Deutsch

FUJITSU

COBOL2000 V1.6

COBOL-Compiler Sprachbeschreibung

Ausgabe Juni 2018

Inhaltsverzeichnis

COBOL Compiler. Sprachbeschreibung.

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an
manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2015

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2015 erfüllt.

Copyright und Handelsmarken

1 Einleitung

In diesem Kapitel werden folgende Themen behandelt:

- Kurzbeschreibung des Produkts
- Zielgruppe und Konzept des Handbuchs
- Änderungen gegenüber der Vorgängerversion
- Anerkennung (Acknowledgment)

1.1 Kurzbeschreibung des Produkts

Bei der Lösung kommerzieller Probleme werden überwiegend große Datenmengen verarbeitet. Dafür eignet sich COBOL besonders. COBOL-Programme sind weitgehend unabhängig von den Eigenheiten einer bestimmten Datenverarbeitungsanlage. Die Sprache ist in einem offiziellen Standard-Dokument vom Normungsinstitut der USA, American National Standards Institute (ANSI), unter folgendem Namen exakt festgelegt:

> „American National Standard for Information Systems
> - Programming Language COBOL
> ANSI X3.23-1985“

Es handelt sich dabei um eine Überarbeitung des Standards von 1974. Die internen Standardfunktionen sind im Addendum „ANSI X3.23a-1989, Intrinsic Function Module“ spezifiziert.

Die deutsche Norm DIN 66028-1986 und die internationale Norm ISO 1989:1985 entsprechen dem American National Standard. Den internen Standardfunktionen nach ANSI entspricht die internationale Norm „ISO/IEC 1989 Amendment 1, Intrinsic Function Module“.

Das Standard-Dokument teilt die COBOL-Sprache zur Beschreibung in einen Nukleus und in elf funktionelle Moduln ein, von denen fünf optional sind (Report Writer, Communication, Debug, Segmentation, Intrinsic Functions). Jeder dieser Moduln enthält wiederum ein bis zwei funktionelle Ausbaustufen, wobei jeweils die untere Stufe eine echte Untermenge der höheren Stufen desselben Moduls darstellt.

Seit Dezember 2002 ist für COBOL die internationale Norm ISO 1989:2002 gültig, in der der Inhalt der Norm ISO 1989:1985 ohne die bisherige Aufteilung in Moduln, das Amendment1 “Intrinsic Function Module“ und zahlreiche neue Sprachmittel zusammengefasst sind.

Der Compiler COBOL2000 (BS2000) unterstützt den COBOL-Sprachumfang „High“ des ANS85. Die optionalen Sprachmoduln Report Writer und Segmentation werden ebenfalls entsprechend dem High-level des ANS85 unterstützt.
Aus der nun gültigen Norm ISO1989:2002 bietet der Compiler COBOL2000 (BS2000) darüber hinaus bereits eine große Teilmenge der Sprachfunktionalität an.

Die im neuen Standard entfallenen optionalen Sprachmoduln Communication und Debug werden nicht unterstützt. Ersatz für das Communication-Modul ist im BS2000 das Produkt openUTM, für das Debug-Modul das Produkt AID.

1.2 Zielgruppe und Konzept des Handbuchs

Dieses Handbuch richtet sich an Programmierer und Schulungskräfte. Es soll als Arbeitsgrundlage zur Programmerstellung und -wartung sowie als Ergänzung zu Schulungsmaterialien dienen.
Es ist weder ein COBOL-Lehrbuch noch ein Benutzerhandbuch.
Allgemeine Programmierkenntnisse und Grundkenntnisse der COBOL-Sprache werden vorausgesetzt.

Wie der Compiler bedient und ein ablauffähiges COBOL-Programm erstellt wird, ist im „COBOL2000 Benutzerhandbuch“ [1] beschrieben.

Das Handbuch enthält alle zum Erstellen von COBOL-Programmen möglichen Sprachelemente, gegliedert nach Funktion, Format, Syntaxregeln, Allgemeinen Regeln und Beispielen:

Funktion gibt eine knappe, allgemeine Beschreibung der einzelnen Sprachelemente. Falls mehrere Formate vorhanden sind, werden deren funktionelle Unterschiede kurz erklärt.

Format definiert die spezifische Art von Zeichenfolgen und Trennsymbolen, damit sie eine zulässige Klausel, Anweisung oder zusammengesetzte Struktur ergeben. Das Auftreten spezifischer Zeichenfolgen und Trennsymbole und die Reihenfolge, wie sie im Format gezeigt werden, ist ausschlaggebend.

Die besondere Notation zur Beschreibung der Formate wird unter „Allgemeines Format“ erklärt.

Ist mehr als eine spezifische Anordnung erlaubt, werden die Formate mit „**Format 1, Format 2**“ usw. bezeichnet.

Syntaxregeln beschreiben die speziellen Anforderungen und Einschränkungen für eine Funktion und bieten zusätzliche Erläuterungen und Anwendungsvorschriften.

Allgemeine Regeln beschreiben die Anwendung der Sprachstruktur innerhalb des Programmkontexts, d.h. in Abhängigkeit von vorausgehenden und nachfolgenden sowie von über- und untergeordneten Strukturen und im Zusammenhang mit Aufrufen und Querverweisen von anderen Sprachelementen, die eigentlich unabhängig von der bezeichneten Struktur sind. Beschränkungen für die Reihenfolge der Wirkungen beim Programmlauf werden erklärt. Alle diese Hinweise befassen sich im Allgemeinen mit Elementen, die nicht direkt im Format erscheinen.

Beispiel zeigt den konkreten Einsatz des beschriebenen Sprachmittels.

Die verwendeten Fachausdrücke entsprechen den in DIN 66028 festgelegten deutschen Übersetzungen der englischen COBOL-Fachausdrücke, z.B. Anweisung für statement usw.

Der Aufbau des Handbuchs orientiert sich an der Struktur des Standard-Dokuments für COBOL.

Einige Sprachmittel sind farbig gekennzeichnet, und zwar mit folgenden Unterscheidungen:

blaugrüne Schrift	Spracherweiterungen des COBOL2000-Compilers gegenüber dem COBOL-Standard 1985. Dazu gehören: • herstellerspezifische Erweiterungen, • Erweiterungen aus dem „Journal of Development“ (JOD) • Erweiterungen aus dem X/OPEN Portability Guide • Erweiterungen aus dem COBOL-Standard 2002
orange Schrift	Sprachmittel, die in neuen Programmen nicht verwendet werden sollen, weil sie von künftigen COBOL-Normen nicht mehr unterstützt werden (obsolete elements). Ihre Entfernung aus alten Programmen ist ratsam.

Das Inhaltsverzeichnis gibt Aufschluss über die gesamte Gliederung des Handbuchs.

Die Stichwörter sichern einen schnellen Zugriff auf die gewünschte Information. Im Abschnitt „Begriffserklärungen“ sind in alphabetischer Reihenfolge die wichtigsten in diesem Handbuch verwendeten Begriffe und Ausdrücke der Sprache COBOL definiert.

Literaturhinweise werden im Text in Kurztiteln angegeben. Der vollständige Titel jeder Druckschrift, auf die verwiesen wird, ist im Literaturverzeichnis aufgeführt.

Der Abschnitt „Begriffserklärungen“ und das Stichwortverzeichnis sind von der Farbkennzeichnung ausgenommen.

1.3 Änderungen gegenüber der Vorgängerversion

Verarbeiten von XML-Dateien

- XML GENERATE-Anweisung

1.4 Anerkennung (Acknowledgment)

Die in diesem Handbuch beschriebene Programmiersprache COBOL basiert auf der im Standarddokument „American National Standard for Information Systems - Programming Language - COBOL X3.23-1985“ festgelegten Sprache. In Anerkennung der Entwicklungs- und Standardisierungsarbeiten für die COBOL-Sprache ist es üblich, einer COBOL-Beschreibung folgenden, im Original wiedergegebenen Text voranzustellen:

„Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication (any organization using a short passage from this document, such as in a book review, is requested to mention ’COBOL’ in acknowledgment of the source, but need not quote the acknowledgment):

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted materials used hereinFLOW-MATIC (trademark of Sperry Rand Corporation), Programming for theUNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translater Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.“

2 Einführung in die COBOL-Sprache

In diesem Kapitel werden folgende Themen behandelt:

2.1 Begriffserklärungen

Mit den nachfolgend definierten Ausdrücken wird in diesem Handbuch die COBOL-Sprache beschrieben. Die Bedeutung der Ausdrücke für COBOL trifft nicht unbedingt auf andere Programmiersprachen zu.
In den Begriffserklärungen sind die wesentlichen Merkmale kurz zusammengefasst. Detaillierte Angaben und syntaktische Regeln sind den nachfolgenden Kapiteln zu entnehmen.

Ablaufeinheit

Run Unit

Eine bestimmte Anzahl von Zielprogrammen, die zur Ausführungszeit als Einheit fungieren.

Absteigender Sortierschlüssel

Descending Key

Ein Schlüssel, nach dessen Werten die Daten sortiert werden, und zwar vom höchsten bis zum niedrigsten Wert des Schlüssels, entsprechend den Regeln für den Vergleich von Datenfeldern.

Adresse

Address

Adressen können sich auf Daten oder Programme beziehen.

Aktueller Datensatz

Current Record

Der Satz, der im Satzbereich einer Datei verfügbar ist.

Alphabetisches Zeichen

Alphabetic Character

Einer der folgenden Buchstaben:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
und das Leerzeichen.

Alphabetname

Alphabet-Name

Ein benutzerdefinierter Name im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION, der einem speziellen Zeichensatz und/oder einer Sortierfolge einen Namen zuweist.

Alphanumerische Datengruppe

Alphanumeric Group Item

Jede Datengruppe mit Ausnahme von nationalen oder stark typisierten Datengruppen.

Alphanumerisches Zeichen

Alphanumeric Character

Ein Zeichen dargestellt mit dem EBCDIC Zeichensatz, unabhängig davon, ob es eine bildliche Darstellung besitzt.

Alternativer Satzschlüssel

Alternate Record Key

Ein zum primären Satzschlüssel unterschiedlicher Schlüssel, mit dem ein Satz aus einer indizierten Datei bezeichnet werden kann.

Angabe

Phrase

Eine geordnete Folge von einer oder mehreren COBOL-Zeichenfolgen, die einen Teil einer COBOL-Anweisung oder -Klausel bilden.

Anweisung

Statement

Eine syntaktisch richtige Kombination von Wörtern und Symbolen, die mit einem Verb beginnt und in der PROCEDURE DIVISION geschrieben wird.

Anzeigenbereich

Indicator Area

Spalte 7 des COBOL-Referenzformates.

Argument

Argument

Ein Bezeichner, ein Literal oder ein arithmetischer Ausdruck zur Angabe eines Wertes, der für die Auswertung einer Funktion verwendet wird.

Arithmetischer Ausdruck

Arithmetic Expression

Ein arithmetischer Ausdruck kann sein:

- ein Bezeichner für ein numerisches Datenelement,
- ein numerisches Literal,
- zwei arithmetische Ausdrücke, die durch einen arithmetischen Operator getrennt sind,
- ein arithmetischer Ausdruck, der in Klammern eingeschlossen ist.

Arithmetischer Operator

Arithmetic Operator

Ein einzelnes Zeichen oder eine Zwei-Zeichen-Kombination der folgenden Art:

Zeichen	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
* *	Potenzierung

Aufgerufenes Ablaufelement

activated runtime element

Ein Programm oder eine Methode, die in einer CALL- bzw. INVOKE-Anweisung bezeichnet werden und zur Ablaufzeit mit dem aufrufenden Programmteil eine Ablaufeinheit bilden.

Aufrufendes Ablaufelement

activating runtime element

Ein Programm oder eine Methode, die die aufrufende Anweisung enthalten.

Aufsteigender Sortierschlüssel

Ascending Key

Ein Schlüssel, nach dessen Werten die Daten sortiert werden, und zwar vom niedrigsten bis zum höchsten Wert des Schlüssels, entsprechend den Regeln für den Vergleich von Datenfeldern.

Ausführungszeit

Execution Time

Die Zeit, während der ein Zielprogramm ausgeführt wird.

Ausgabedatei

Output File

Eine Datei, die entweder im Ausgabemodus oder im Erweiterungsmodus eröffnet wird.

Ausgabemodus

Output Mode

Der Zustand einer Datei nach Ausführung einer OPEN-Anweisung mit OUTPUT- oder EXTEND-Angabe und vor Ausführung einer CLOSE-Anweisung für diese Datei.

Ausgabeprozedur

Output Procedure

Eine Folge von Anweisungen, die während der Ausführung einer SORT-Anweisung jedesmal ausgeführt wird, nachdem ein sortierter Satz an die Sortierdatei übergeben oder während der Ausführung einer MERGE-Anweisung, nachdem der nächste zu mischende Satz ausgewählt worden ist.

Ausnahmesituation

Exception

Eine Situation zur Laufzeit, die anzeigt, dass ein Fehler bzw. eine Abweichung zur normalen Ausführung entstanden ist.

Ausnahmesituationsname

Exception-Name

Wort, das eine Ausnahmesituation benennt.

Ausnahmezustand

Exception-Condition

Eine ausgelöste Ausnahmesituation.

Ausnahmezustand auslösen

Raising of an Exception-Condition

Übergang einer Ausnahmesituation in einen Ausnahmezustand durch:

1. eine durch die TURN-Direktive eingeschaltete Überprüfung der Ausnahmesituation.
2. die Ausführung einer RAISE-Anweisung für diese Ausnahmesituation.

Bedingte Anweisung

Conditional Statement

Eine bedingte Anweisung veranlasst das Prüfen des Wahrheitswertes einer Bedingung und bestimmt, dass die darauffolgende Aktion des Zielprogramms von diesem Wahrheitswert abhängt.

Bedingung

Condition

Ein Zustand eines Programms während der Ablaufzeit, für den ein Wahrheitswert ermittelt werden kann. Der Ausdruck „bedingung“ (bedingung-1, bedingung-2,...) repräsentiert in der vorliegenden Beschreibung entweder eine einfache Bedingung oder eine zusammengesetzte Bedingung, die eine syntaktisch zugelassene Kombination von einfachen Bedingungen, logischen Operatoren und Klammerpaaren enthält, für die ein Wahrheitswert ermittelt werden kann.

Bedingungsausdruck

Conditional Expression

Eine einfache oder komplexe Bedingung, die in einer IF-, PERFORM-, EVALUATE- oder SEARCH-Anweisung vorkommt.

Bedingungsname

Condition-Name

Ein benutzerdefinierter Name, der einem bestimmten Wert, einer Gruppe von Werten oder einer Folge von Werten, die eine Bedingungsvariable annehmen kann, zugeordnet ist, bzw. ein benutzerdefinierter Name, der dem Zustand eines Prozess- oder Benutzerschalters zugeordnet ist.

Bedingungsnamen-Bedingung

Condition-Name Condition

Bewirkt, dass eine Bedingungsvariable geprüft wird, um zu entscheiden, ob ihr Wert gleich einem der Werte ist, die zu einem bestimmten Bedingungsnamen gehören.

Bedingungsvariable

Conditional Variable

Ein Datenfeld, dessen Wert (oder mehreren Werten) ein Bedingungsname zugeordnet ist.

Begrenzer

Delimiter

Ein Zeichen (oder eine Folge von benachbarten Zeichen), das das Ende einer Zeichenfolge anzeigt und das eine Zeichenfolge von weiteren Zeichenfolgen trennt. Ein Begrenzer ist nicht Teil der Zeichenfolgen, die er trennt.

Benutzerdefinierter Name

User-Defined Word

Ein COBOL-Wort, das vom Programmierer entsprechend dem Format einer Klausel oder Anweisung gewählt wird.

Bezeichner

Identifier

Eine syntaktisch richtige Kombination von Zeichen und Trennzeichen, die ein Datenfeld benennt. Der Bezeichner besteht aus einem Datennamen und den entsprechenden Kennzeichnern, Subskripten und Teilfeld-Selektoren, soweit diese für die Eindeutigkeit der Bezugnahme erforderlich sind. Der Bezeichner einer Funktion (Intrinsic Function) ist gesondert unter dem Begriff „Funktionsbezeichner“ definiert.

Bezugsschlüssel

Key of Reference

Der primäre oder alternative Satzschlüssel, über den aktuell auf Datensätze einer indizierten Datei zugegriffen wird.

Bibliotheksname

Library-Name

Ein benutzerdefinierter Name, der eine Übersetzungseinheit-Bibliothek bezeichnet, die mehrere COBOL-Texte mit verschiedenen Namen enthalten kann.

Bibliothekstext

Library-Text

Zeichenfolgen und/oder Trennsymbole in einer COBOL-Bibliothek.

Binäres Suchen

Binary Search

Eine Methode, eine auf- oder absteigend geordnete Tabelle nach einem bestimmten Element zu durchsuchen. Die Suche wird in jeweils halbierten Bereichen vorgenommen. Dabei wird bei jedem Suchschritt geprüft, ob das Element, das sich in der Mitte befindet, größer, kleiner oder gleich dem gesuchten ist. Dieses Halbieren und Vergleichen setzt sich fort, bis das geprüfte Element mit dem gesuchten übereinstimmt.

Block

Block

Eine physische Dateneinheit, die normalerweise aus einem oder mehreren logischen Sätzen oder aus einem Teil eines logischen Satzes besteht. Die Größe eines Blocks hängt nicht unmittelbar mit der Größe der Datei zusammen, in der der Block enthalten ist, oder mit der Größe der logischen Sätze, die entweder im Block enthalten sind oder ihn überlappen. Block ist gleichbedeutend mit „Physischer Satz“.

COBOL-Wort

COBOL Word

siehe „Wort“.

COBOL-Zeichenvorrat

COBOL Character Set

Die Menge von Zeichen, mit denen sich die Syntax einer COBOL-Übersetzungsgruppe schreiben lässt, mit Ausnahme von Kommentaren und dem Inhalt von nicht-hexadezimalen alphanumerischen und nicht-hexadezimalen nationalen Literalen.

COMMON-Programm

Common Program

Ein inneres Programm einer geschachtelten Quelleinheit, dessen Name mit dem COMMON-Attribut versehen ist. Ein solches Programm kann außer vom direkt übergeordneten Programm auch von jedem „Geschwisterprogramm“ und dessen „Abkömmlingen“ aufgerufen werden.

Datei

File

Eine Sammlung von Datensätzen.

Dateierklärung

File Description Entry

Eine Erklärung in der FILE SECTION der DATA DIVISION, die aus der Stufenbezeichnung FD, einem Datennamen und einer Folge von Dateiklauseln besteht.

Dateiklausel

File Clause

Eine Klausel, die als Teil einer der folgenden Erklärungen in der DATA DIVISION vorkommt:

Dateierklärung (FD)

Sortierdateierklärung (SD)

Listenerklärung (RD)

Dateiname

File-Name

Ein benutzerdefinierter Name, der eine Datei bezeichnet, die in einer Dateierklärung oder Sortierdateierklärung in der FILE SECTION der DATA DIVISION beschrieben ist.

Dateiorganisation

File Organization

Eine unveränderliche, logische Dateistruktur, die zum Zeitpunkt der Dateierzeugung festgelegt wird.

Dateisteuerbereich

File Connector

Ein Speicherbereich, der Informationen über eine Datei enthält. Er wird verwendet als Verknüpfung zwischen einem Dateinamen und einer physischen Datei sowie zwischen einem Dateinamen und dem zugeordneten Satzbereich.

Dateipositionsindikator

File Position Indicator

Eine logische Informationseinheit, die eine Positionsbeschreibung des Satzes enthält, auf den bei der nächsten READ-Anweisung zugegriffen wird. Falls kein solcher Satz existiert, zeigt der Dateipositionsindikator an, warum der Satz nicht existiert, d.h. „ungültig“ ist. Der Dateipositionsindikator wird nur durch die Anweisungen CLOSE, OPEN, READ und START verändert.

Datenadresse

Data-Address

Eine Datenadresse ist eine konzeptuelle Dateneinheit, die den Speicherort eines Datums identifiziert. Eine Datenadresse kann in einem Datenzeiger gespeichert werden.

Datenelement

Elementary Item

Ein Datenfeld, das nicht noch weiter logisch unterteilt ist.

Datenerklärung

Data Description Entry

Eine Erklärung in der DATA DIVISION, die aus einer Stufennummer, gegebenenfalls einem Datennamen und einer Folge von Datenklauseln besteht.

Datenfeld

Data Item

Eine Dateneinheit (ausgenommen Literale), die durch ein COBOL-Programm oder durch die Regeln einer Funktionsauswertung definiert ist.

Datenklausel

Data Clause

Eine Klausel, die in einer Datenerklärung der DATA DIVISION vorkommt und die Information für die Beschreibung eines bestimmten Attributs eines Datenfeldes liefert.

Datenname

Data-Name

Ein benutzerdefinierter Name, der ein Datenfeld bezeichnet, das in einer Datenerklärung der DATA DIVISION beschrieben ist. Wenn datenname in den allgemeinen Formaten auftritt, darf er weder indiziert noch gekennzeichnet sein, außer es ist in den Regeln ausdrücklich erlaubt.

Datensatz

Record

Ein Datenfeld auf der höchsten Stufe der Hierarchie, das in keinem anderen Datensatz enthalten ist.

Datensatzbereich

Record Area

Ein Speicherbereich, der zugewiesen wird, um Sätze zu bearbeiten, die in einer Datensatzerklärung in der FILE SECTION beschrieben worden sind.

Datensatzerklärung

Record Description Entry

Die vollständige Folge von Datenerklärungen, die zu einem bestimmten Datensatz gehört.

Datensatzname

Record-Name

Ein benutzerdefinierter Name, der einen Datensatz bezeichnet, der in einer Datensatzerklärung der DATA DIVISION beschrieben ist.

Datensatznummer

Record Number

Die Folgenummer eines Datensatzes in einer sequenziell organisierten Datei.

Datensatzschlüssel

Record Key

Entweder ein primärer oder ein alternativer Datensatzschlüssel, dessen Inhalt einen Datensatz in einer indizierten Datei bezeichnet.

Datenzeiger

Data-pointer

Ein Datenzeiger ist ein Datenelement, in dem eine Datenadresse gespeichert wird.

Deeditieren

De-editing

Die Ermittlung des numerischen Wertes von numerisch druckaufbereiteten Daten.

Direkte Indizierung

Bei direkter Indizierung wird der Index in Form eines direkten Subskripts verwendet. Siehe „Direkte Subskribierung“.

Direkte Subskribierung

Bei der direkten Subskribierung wird das Subskript entweder durch ein ganzzahliges Literal oder durch einen Datennamen angegeben, der als numerisches Datenelement ohne Zeichenstellen rechts vom angenommenen Dezimalpunkt erklärt ist.

Druckaufbereitungszeichen

Editing Character

Ein einzelnes Zeichen oder eine Zwei-Zeichen-Kombination aus der folgenden Liste:

Zeichen	Bedeutung
B	Leerzeichen
0	Null
+	Pluszeichen
-	Minuszeichen
CR	Kredit
DB	Debet
Z	Nullenunterdrückung durch Z (Leerzeichen)

*	Nullenunterdrückung durch Stern (*)
$	Währungszeichen
,	Komma (Dezimalpunkt)
.	Punkt (Dezimalpunkt)
/	Schrägstrich

Druckdezimalpunkt

Actual Decimal Point

Die physische Darstellung der Stelle des Druckdezimalpunkts in einem Datenfeld unter Verwendung der Zeichen . (Punkt) oder , (Komma).

Druckfähige Liste

Printable Group

Eine Liste, die mindestens eine Druckzeile enthält.

Druckfähiges Datenfeld

Printable Item

Ein Datenfeld, dessen Größe und Inhalt in einer Listenerklärung beschrieben ist. Diese Listenerklärung enthält eine COLUMN NUMBER-Klausel, eine PICTURE-Klausel, eine SOURCE-Klausel, SUM-Klausel oder VALUE-Klausel.

Dynamischer Zugriff

Dynamic Access

Die Methode des Wechsels zwischen sequenziellem und wahlfreiem Zugriff. Diese Zugriffsmethode kann nur für relative oder indizierte Dateien angegeben werden.

Ein-/Ausgabe-Datei

Input-Output File

Eine Datei, die im Ein-/Ausgabe-Modus eröffnet ist.

Ein-/Ausgabe-Modus

I-O Mode

Der Zustand einer Datei nach Ausführung einer OPEN-Anweisung mit I-O-Angabe und vor Ausführung einer CLOSE-Anweisung für diese Datei.

Ein-/Ausgabe-Zustand

I-O Status

Der Ein-/Ausgabe-Zustand ist ein Wert, der in ein zwei Zeichen langes Datenfeld übertragen wird, um dem COBOL-Programm den Zustand einer Ein-/Ausgabe-Operation anzuzeigen. Dieser Wert wird nur dann übertragen, wenn die FILE STATUS-Klausel im
FILE-CONTROL-Paragrafen angegeben ist.

Einfache Bedingung

Simple Condition

Eine einzelne der nachfolgenden Bedingungen:

Vergleichsbedingung
Klassenbedingung
Bedingungsnamen-Bedingung
Schalterzustandsbedingung
Vorzeichenbedingung

Eingabedatei

Input File

Eine Datei, die im Eingabemodus eröffnet ist.

Eingabemodus

Input Mode

Der Zustand einer Datei nach Ausführung einer OPEN-Anweisung mit INPUT-Angabe und vor Ausführung einer CLOSE-Anweisung für diese Datei.

Eingabeprozedur

Input Procedure

Eine Folge von Anweisungen, die jedesmal dann ausgeführt wird, wenn ein Satz an die Sortierdatei übergeben wird.

Einstelliger Operator

siehe „Unärer Operator“

Eintragung

Entry

Jede mit einem Punkt abgeschlossene Folge von Klauseln, die in der IDENTIFICATIONDIVISION, ENVIRONMENT DIVISION oder DATA DIVISION einer COBOL-Übersetzungseinheit geschrieben wird.

Element-Positions-Vektor (EPV)

Element Position Vector

Eine logische Informationseinheit, die zu jedem Datenfeld, für das eine IDENTIFIED-Klausel angegeben ist, festhält, welcher Knoten aus der Baumdarstellung eines XML-Dokuments ihm zugeordnet ist.

END PROGRAM-Eintrag

End Program Header

Ein Eintrag, der das Ende eines COBOL-Programms anzeigt. Er besteht aus den Schlüsselwörtern END PROGRAM, dem Programmnamen und dem Abschlusspunkt.

Eröffnungsmodus

Open Mode

Der Zustand einer Datei nach Ausführung einer OPEN-Anweisung und vor Ausführung einer CLOSE-Anweisung für diese Datei.

Der genaue Eröffnungsmodus ist in der OPEN-Anweisung entweder mit INPUT, OUTPUT, I-O oder EXTEND beschrieben.

Erweiterter Zugriff

Ist eine Methode des Wechsels zwischen sequenziellem und wahlfreiem Zugriff. Diese Zugriffsmethode kann nur für indizierte Dateien angegeben werden.

Erweiterungsmodus

Extend Mode

Der Zustand einer Datei nach Ausführung einer OPEN-Anweisung mit EXTEND-Angabe und vor Ausführung einer CLOSE-Anweisung für diese Datei.

Explizit begrenzte Anweisung

Delimited Scope Statement

Jede Anweisung, die einen expliziten Bereichsbegrenzer enthält.

Expliziter Bereichsbegrenzer

Explicit Scope Terminator

Ein reserviertes Wort, das den Gültigkeitsbereich einer einzelnen Anweisung in der PROCEDURE DIVISION begrenzt.

Externes Datenfeld

External Data Item

Ein Datenfeld, das als Teil eines externen Datensatzes in einem oder mehreren Programmen einer Ablaufeinheit beschrieben ist. Auf ein externes Datenfeld kann von jedem Programm, in dem es beschrieben ist, zugegriffen werden.

Externer Datensatz

External Data Record

Ein logischer Datensatz, der in einem oder mehreren Programmen einer Ablaufeinheit beschrieben ist. Auf die Datenfelder eines solchen externen Datensatzes kann von jedem Programm, in dem der Satz beschrieben ist, zugegriffen werden.

Fabrikdefinition

Factory Definition

Die Quelleinheit, die ein Fabrikobjekt beschreibt.

Fabrikobjekt

Factory Object

Jede Klasse besitzt nur ein Fabrikobjekt, das alle anderen Objekte der Klasse erzeugt. Es wird durch die Fabrikdefinition einer Klasse spezifiziert.

Folgenummernbereich

Sequence Number Area

Spalten 1-6 im COBOL-Fixed-Form-Referenzformat.

Format

Format

Eine spezifische Anordnung von Zeichenfolgen und Trennsymbolen einer Anweisung oder Klausel.

Füllzeichen

Padding Character

Ein alphanumerisches Zeichen, mit dem die nicht verwendeten Zeichenpositionen eines physischen Satzes aufgefüllt werden.

Funktion

Function

Ein temporäres Datenfeld, dessen Wert durch einen Auswertungsmechanismus bestimmt wird, der bei Referenzierung der Funktion während der Ausführung einer Anweisung wirksam wird.

Ganze Zahl (Ganzzahl)

Integer

Ein numerisches Literal oder ein numerisches Datenfeld, das keine Zeichenposition rechts vom angenommenen Dezimalpunkt enthält. Wo die Bezeichnung „Ganzzahl“ in den Formaten auftritt, muss „Ganzzahl“ ein ganzzahliges numerisches Literal ohne Vorzeichen und ungleich Null sein, außer die Regeln des Formats lassen ausdrücklich etwas anderes zu.

Ganzzahlige Funktion

Integer Function

Eine Funktion, deren Kategorie numerisch ist und deren Returnwert rechts vom Dezimalpunkt bei jedem möglichen Wert nur die Ziffer 0 enthält.

Gekennzeichneter Datenname

Qualified Data-Name

Ein Bezeichner, der sich aus einem Datennamen, gefolgt von einer oder mehreren Angaben eines der Verknüpfer OF oder IN und einem weiteren Datennamen (Kennzeichner) zusammensetzt.

Globaler Name

Global Name

Ein Name, der in nur einem Programm deklariert ist, aber von jedem Programm referenziert werden kann, das in diesem Programm direkt oder indirekt enthalten ist. Globale Namen können sein: Bedingungsnamen, Datennamen, Dateinamen, Datensatznamen, Listennamen, Typnamen sowie einige Sonderregister.

Geschachteltes Programm

Nested Source Program

Ein COBOL-Programm, das andere Programme enthält, die wiederum weitere Programme enthalten können. Es besteht demnach aus einem äußeren Programm und einem oder mehreren darin enthaltenen (contained) Programmen.

Gruppenbegriff

Control Data Item

Ein Datenfeld, dessen Inhalt für einen Gruppenwechsel ausschlaggebend ist.

Gruppenbegriffsname

Control Data-Name

Ein Datenname in einer CONTROL-Klausel, der sich auf einen Gruppenbegriff bezieht.

Gruppenfuß

Control Footing

Eine Leiste, die am Ende der Gruppenleiste, zu der sie gehört, auftritt.

Gruppenhierarchie

Control Hierarchy

Eine bestimmte Folge von Unterteilungen, definiert durch die positionsgebundene Stellung von FINAL und die Datennamen innerhalb einer CONTROL-Klausel.

Gruppenkopf

Control Heading

Eine Leiste, die zu Beginn des Abschnitts Gruppenbegriffe, dessen Bestandteil sie ist, erscheint.

Gruppenleiste

Control Group

Eine zusammengehörige Folge von Daten, die einem Gruppenbegriff in der Gruppenhierarchie zugeordnet ist. Für einen gegebenen Gruppenbegriff besteht die Gruppenleiste aus der gesamten Folge von Gruppenköpfen, Gruppenfüßen und den zugehörigen Postenleisten.

Gruppenwechsel

Control Break

Eine Änderung im Wert eines Datenfeldes, das in der CONTROL-Klausel bezeichnet ist. Im Allgemeinen eine Änderung im Wert eines Datenfeldes, das benutzt wird, um die hierarchische Struktur einer Liste zu überwachen.

Gruppenwechselstufe

Control Break Level

Die relative Position innerhalb einer Gruppenhierarchie, in der der häufigste Gruppenwechsel stattfindet.

Herstellername

Implementor-Name

Herstellername muss ein Name aus der nachfolgenden Liste sein:

CONSOLE*)	literal
TERMINAL*)	jobvariablenname
SYSIPT*)	TSW-0 bis TSW-31
PRINTER, PRINTER01-PRINTER99	USW-0 bis USW-31
SYSOPT*)	COMPILER-INFO
ARGUMENT-NUMBER*)	CPU-TIME
ARGUMENT-NAME*)	PROCESS-INFO
ENVIRONMENT-NAME*)	TERMINAL-INFO
ENVIRONMENT-VALUE*)	DATE-ISO4
C01 bis C08; C10, C11	

*) reservierte Wörter innerhalb der ENVIRONMENT DIVISION

Hexadezimale Ziffer

Hexadecimal Digit

Zeichen aus dem Wertebereich 0..9, A..F und a..f

Index

Index

Ein spezielles Adressfeld, das mit einer Tabelle verbunden ist und dessen Inhalt die Distanz eines Tabellenelementes zum Tabellenanfang darstellt. Ein Index ist kein Datenfeld.

Indexdatenfeld

Index Data Item

Ein Datenfeld, in welchem der Wert abgespeichert werden kann, der mit dem Indexnamen verbunden ist.

Indexname

Index-Name

Ein benutzerdefinierter Name, der einen Index bezeichnet, der mit einer bestimmten Tabelle verbunden ist.

Indizierter Datenname

Indexed Data-Name

Ein Bezeichner, der sich aus einem Datennamen, gefolgt von einem oder mehreren Indexnamen, die in runde Klammern eingeschlossen sind, zusammensetzt.

Indizierte Datei

Indexed File

Eine Datei mit indizierter Organisation.

Indizierte Organisation

Indexed Organization

Eine unveränderliche, logische Dateistruktur, in der jeder Datensatz durch den Wert eines Schlüssels oder mehrerer Schlüssel innerhalb dieses Datensatzes bezeichnet ist.

Initial-Programm

Initial Program

Ein Programm, das sich bei jedem Aufruf innerhalb einer Ablaufeinheit im Initialzustand befindet.

Initialzustand

Initial State

Der Zustand eines Programms, wenn es zum ersten Mal innerhalb einer Ablaufeinheit aufgerufen wird.

Interne Daten

Internal Data

Die Daten, die in einem Programm beschrieben werden, ausgenommen alle externen Datenfelder und externen Dateien. Datenfelder, die in der LINKAGE SECTION eines Programms definiert sind, werden als interne Daten behandelt.

Internes Datenfeld

Internal Data Item

Ein Datenfeld, das in einem Programm einer Ablaufeinheit beschrieben ist. Ein internes Datenfeld kann einen globalen Namen haben.

Interne Datei

Internal File

Eine Datei, auf die nur ein Programm der Ablaufeinheit zugreifen kann.

Kapitel

Section

Ein Kapitel besteht aus Paragrafen oder Klauseln. Dem Inhalt ist die Kapitelüberschrift vorangestellt. Ein Kapitel kann leer sein oder einen oder mehrere Paragrafen enthalten.

Kapitelname

Section-Name

Ein benutzerdefinierter Name, der ein Kapitel in der PROCEDURE DIVISION bezeichnet.

Kapitelüberschrift

Section Header

Eine Kombination von Wörtern, gefolgt von einem Punkt und einem Leerzeichen. Sie zeigt den Beginn eines Kapitels in der ENVIRONMENT DIVISION, DATA DIVISION und PROCEDURE DIVISION an. In der ENVIRONMENT DIVISION und DATA DIVISION wird die Kapitelüberschrift mit reservierten Wörtern, gefolgt von einem Punkt und einem Leerzeichen, gebildet.

Die zulässigen Kapitelüberschriften sind:

In der ENVIRONMENT DIVISION:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

In der DATA DIVISION:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION
LINKAGE SECTION.
REPORT SECTION.
SUB-SCHEMA SECTION.
```

In der PROCEDURE DIVISION wird eine Kapitelüberschrift mit einem Kapitelnamen, gefolgt von dem Wort SECTION, einer Segmentnummer (wahlweise), einem Punkt und einem Leerzeichen, gebildet.

Kennzeichner

Qualifier

Ein Kennzeichner ist:

1. ein Datenname, der in einer Bezugnahme zusammen mit einem anderen Datennamen benutzt wird, der auf einer niedrigeren Stufe in derselben Hierarchie steht,
2. ein Kapitelname, der in einer Bezugnahme zusammen mit einem Paragrafen benutzt wird, der in diesem Kapitel beschrieben ist,
3. ein Bibliotheksname, der in einer Bezugnahme zusammen mit einem Textnamen benutzt wird, der mit dieser Bibliothek verbunden ist.

Klasse (objektorientiert)

Class

Eine Klasse beschreibt durch die Klassendefinition eine Menge von Objekten mit ihrenAttributen und Methoden.

Klassenbedingung
Class Condition

Mit der Klassenbedingung wird geprüft, ob der Inhalt eines Datenfeldes

- vollständig numerisch ist,
- vollständig alphabetisch ist,
- vollständig aus Großbuchstaben besteht,
- vollständig aus Kleinbuchstaben besteht,
- ausschließlich aus Zeichen besteht, die durch die Definition des Klassennamens im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION festgelegt sind.

Klassendefinition (objektorientiert)

Class Definition

Eine Übersetzungseinheit, die eine Klasse von Objekten definiert.

Klassenname (objektorientiert)

Class-Name

Ein vom Benutzer definierter Name, der eine Klasse bezeichnet.

Klassenname (für Bedingungen)

Class-Name

Ein benutzerdefinierter Name, der im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION festgelegt wird und einen vom Programmierer definierten Zeichenvorrat benennt. Soll der Inhalt eines Datenfeldes daraufhin überprüft werden, ob er nur Zeichen dieses Zeichenvorrats enthält, wird in der Klassenbedingung der Klassenname angegeben.

Klausel

Clause

Eine funktionelle Folge von COBOL-Wörtern, deren Zweck es ist, das Attribut einer Erklärung festzulegen.

Komplexe Bedingung

Complex Condition

Eine Bedingung, in welcher ein logischer Operator oder mehrere logische Operatoren sich auf eine Bedingung oder mehrere Bedingungen auswirken.

Konformität

Conformance

für Objekte:

Die Eigenschaft, die es erlaubt, ein Objekt mit einer gegebenen Schnittstelle A auch dort zu verwenden, wo ein Objekt mit einer anderen Schnittstelle B erwartet wird. Die Konformität der Schnittstellen stellt sicher, dass jede Operation der Schnittstelle B auch von der dazu konformen Schnittstelle A unterstützt wird.

für Parameter:

Die Anforderungen an aktuelle und entsprechende formale Parameter, sowie an Rückgabeparameter in rufenden und gerufenen Ablaufelementen.

Konvertierung

Conversion

Implizite Umwandlung von numerischen Werten von einem Format in ein anderes Format bzw. von Werten von Indizes in Tabellenelementnummern und umgekehrt.

- Wert von Indizes (Zahl in Binärform) -> Tabellenelementnummern Umwandlung erfolgt in beiden Richtungen mit der Formel:

  Wert von Index = (Tabellenelementnummer - 1) * Länge des Tabellenelements

  Die Konvertierung ist also von der Tabelle abhängig.
- Verschiedene USAGEs von numerischen Datenfeldern ineinander.

Leiste

Report Group

Eine 01-Stufenerklärung und die untergeordneten Stufenerklärungen in der REPORT SECTION der DATA DIVISION.

Leistenerklärung

Report Group Description Entry

Ein Eintrag in der REPORT SECTION der DATA DIVISION, der sich aus der Stufennummer 01, dem gewählten Datennamen, einer TYPE-Klausel und einer wahlweisen Folge von REPORT-Klauseln zusammensetzt.

Letzter Ausnahmezustand

Last Exception-Status

Der letzte in einer Ablaufeinheit ausgelöste Ausnahmezustand. Dies kann auch der Zustand „kein Ausnahmezustand ist ausgelöst“ sein.

Linksbündiges Ende

High Order End

Das am weitesten links stehende Zeichen einer Zeichenfolge.

Listendatei

Report File

Eine Ausgabedatei, deren Dateierklärung die REPORT-Klausel enthält. Der Inhalt einer Listendatei besteht aus Datensätzen, die unter der Kontrolle des Listenprogrammkontrollsystems geschrieben werden.

Listenerklärung

Report Description Entry

Eine Erklärung in der REPORT SECTION der DATA DIVISION, die sich aus dem Stufenbezeichner RD, einem Listennamen und REPORT-Klauseln zusammensetzt.

Listenfuß

Report Footing

Eine Leiste, die das Ende einer Liste bezeichnet.

Listenklausel

Report Clause

Eine Klausel in der REPORT SECTION der DATA DIVISION, die in einer Listenerklärung oder Leistenerklärung angegeben wird.

Listenkopf

Report Heading

Eine Leiste, die nur am Anfang einer Liste ausgegeben wird.

Listenname

Report-Name

Ein benutzerdefinierter Name, der eine Liste in einer Listenerklärung in der REPORT SECTION der DATA DIVISION beschreibt.

Listenzeile

Report Line

Teil einer Seite, der eine Reihe von Zeichen darstellt.

Logischer Listensatz

Report Writer Logical Record

Ein Datensatz, der aus der Listenprogrammdruckzeile und der damit zusammenhängenden Information für die Ablaufsteuerung zur Auswahl der Listenprogrammzeile und für die vertikale Positionierung besteht.

Logischer Operator

Logical Operator

Eines der reservierten Wörter AND, OR oder NOT.

Bei der Bildung zusammengesetzter Bedingungen können AND oder OR als logische Verknüpfer, NOT als logische Negation benutzt werden.

Logischer Satz

Logical Record

Ein Datensatz auf der höchsten Stufe der Hierarchie, der in keinem anderen Satz enthalten ist.

Maschineneigene Sortierfolge

Native Collating Sequence

Entsprechend den zur Darstellung von Zeichen verwendeten Zeichensätzen gibt es 2 Sortierfolgen:

- alphanumerische Sortierfolge in 1 Byte
- nationale Sortierfolge in 2 Bytes

In beiden Fällen beruht sie auf der Binärdarstellung der Zeichen.

Maschineneigener Zeichensatz

Native Character Set

Entsprechend den zur Darstellung von Zeichen verwendeten Zeichensätzen gibt es:

- EBCDIC-Zeichensatz zur Darstellung alphanumerischer Zeichen (in 1Byte)
- UTF-16-Zeichensatz zur Darstellung nationaler Zeichen (in 2 Bytes)

Merkname

Mnemonic-Name

Ein festgelegter Name, falls der Programmierer ihn mit einem bestimmten Herstellernamen im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION verknüpft.

Methode

Method

Ablauffähige Anweisungen in der Procedure Division einer Methodendefinition.

Methodendefinition

Method Definition

Die Quelleinheit, die eine Methode definiert.

Methodenname

Method-Name

Ein vom Benutzer definierter Name, der eine Methode bezeichnet.

Mischdatei

Merge File

Eine Sammlung von Sätzen, die auf Grund einer MERGE-Anweisung gemischt werden. Die Mischdatei kann nur mit der MERGE-Funktion erzeugt und verwendet werden.

Nächste ausführbare Anweisung

Next Executable Statement

Die nächste Anweisung, auf die die Ablaufsteuerung nach Ausführung der aktuellen Anweisung übertragen wird.

Nächster ausführbarer Satz

Next Executable Sentence

Der nächste Satz, auf den die Ablaufsteuerung nach Ausführung der aktuellen Anweisung übertragen wird.

Nächster Satz

Next Record

Der Satz, der logisch dem aktuellen Satz einer Datei folgt.

Nationales Zeichen

National Character

Ein Zeichen, das mit dem UTF-16 Zeichensatz dargestellt ist.

Nichtnumerisches Datenfeld

Nonnumeric Item

Ein alphanumerisches oder nationales Datenfeld, dessen Wert aus einer beliebigen Kombination von Zeichen aus dem Zeichensatz der Datenverarbeitungsanlage bestehen kann. Gewisse Kategorien von Datenfeldern können auch aus einem eingeschränkten Zeichensatz gebildet werden.

Nichtnumerisches Literal

Nonnumeric Literal

Ein alphanumerisches oder ein nationales Literal.

Numerische Funktion

Numeric Function

Eine Funktion, deren Klasse und Kategorie numerisch ist.

Null-längiges Datenfeld

Zero-length item

Ein Datenfeld, dessen minimale erlaubte Länge 0 ist, und dessen Länge zur Ablaufzeit 0 ist.

Numerisches Datenfeld

Numeric Item

Ein Datenfeld, dessen Wert mit den Ziffern 0 bis 9 dargestellt wird.
Ist ein Rechenvorzeichen notwendig, so muss dies durch eine erlaubte Darstellung von + oder - erfolgen.

Numerisches Literal

Numeric Literal

Ein Literal, das aus einem oder mehreren numerischen Zeichen besteht, und das außerdem einen Dezimalpunkt oder ein Vorzeichen enthalten darf.
Der Dezimalpunkt darf nicht das äußerste rechte Zeichen sein, das Vorzeichen muss das äußerste linke Zeichen sein.

Numerisches Zeichen

Numeric Character

Jede der folgenden Ziffern:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Oberklasse

superclass

Eine Klasse, die an eine andere Klasse vererbt.

Objekt

Object

Ein Objekt wird durch eine Klasse definiert. Es besteht aus einer Kombination von Daten und Methoden, die damit arbeiten.

Objektdefinition

Object Definition

Die Quelleinheit, die ein Objekt definiert.

Objektreferenz

Object Reference

Ein implizit oder explizit definiertes Datenelement, dessen Inhalt eindeutig auf ein Objekt verweist.

Objektsicht

Object-view

Eine Objektsicht veranlasst den Compiler, eine Objektrefererenz so zu behandeln, als ob sie in der angegebenen Form definiert wäre.

Operand

Operand

Die allgemeine Definition eines Operanden ist: der Teil, der bearbeitet werden muss. In dieser Beschreibung bedeutet Operand: jedes der kleingeschriebenen Wörter, das im Format einer Erklärung, eines Paragrafen, einer Klausel oder einer Anweisung vorkommt.

Optionale Datei

Optional File

Eine Datei, die zur Programmablaufzeit nicht unbedingt vorhanden sein muss. Das Programm, das auf eine solche als optional deklarierte Datei zugreift, verursacht eine Abfrage, ob die Datei vorhanden ist oder nicht.

Paragraf

Paragraph

In der PROCEDURE DIVISION: ein Paragrafenname, dem ein Punkt und ein Leerzeichen folgen bzw. dem ein oder mehrere Sätze folgen können.
In der IDENTIFICATION DIVISION und ENVIRONMENT DIVISION: eine Paragrafenüberschrift, der eine oder mehrere Erklärungen folgen können.

Paragrafenname

Paragraph-Name

Ein Wort, das zur Benennung eines Paragrafen in der PROCEDURE DIVISION verwendet wird.

Paragrafenüberschrift

Paragraph Header

Ein reserviertes Wort, das zur Identifikation vor allen Paragrafen in der IDENTIFICATIONDIVISION und in der ENVIRONMENT DIVISION steht. Die zugelassenen Paragrafenüberschriften lauten:

IDENTIFICATION DIVISION:

CLASS-ID.	Klassenname
METHOD-ID.	Methodenname
OBJECT.	Objektdefinition

FACTORY.	Fabrikdefinition
INTERFACE-ID.	Schnittstellenname
PROGRAM-ID.	Programmname

ENVIRONMENT DIVISION:

SOURCE-COMPUTER.	Übersetzungsrechner
OBJECT-COMPUTER.	Ablaufrechner
SPECIAL-NAMES.	Sondernamen
REPOSITORY.	Klassen- /Schnittstellendefinition
FILE-CONTROL.	Dateizuordnung
I-O-CONTROL.	Ein-/Ausgabe-Steuerung

Physischer Satz

Physical Record

siehe „Block“

Plattenspeicher

Mass Storage

Ein Speichermedium, auf welchem Daten in sequenzieller und nichtsequenzieller Weise organisiert und gewartet werden können.

Plattenspeicherdatei

Mass Storage File

Eine Sammlung von Datensätzen, die auf Platten gespeichert ist.

Primärer Satzschlüssel

Prime Record Key

Ein Schlüssel zur eindeutigen Kennzeichnung eines Satzes innerhalb einer indizierten Datei.

Programmablaufzeit

Object Time

Die Zeit, während der ein Zielprogramm ausgeführt wird.

Programmadresse

Program-Address

Eine Programmadresse identifiziert den Speicherort eines Programms. Eine Programmadresse kann in einem Programmzeiger gespeichert werden.

(Programm)ausführungseinheit

Run Unit

siehe „Ablaufeinheit“

Programmidentifikationsbereich

Program-Identification-Area

Spalten 73 bis 80 im COBOL-Fixed-Form-Referenzformat.

Programmname

Program-Name

Ein vom Benutzer definierter Name, der ein COBOL-Programm bezeichnet.

Programmsatz

Sentence

Eine Anweisung oder eine Folge von Anweisungen, deren letzte durch einen Punkt, gefolgt von einem Leerzeichen, abgeschlossen wird.

Programmteil

Division

Kapitel oder Paragrafen, die in Übereinstimmung mit den entsprechenden Regeln aufgebaut und zusammengestellt sind. In einem COBOL-Programm gibt es vier Programmteile:

IDENTIFICATION DIVISION	Erkennungsteil
ENVIRONMENT DIVISION	Maschinenteil
DATA DIVISION	Datenteil
PROCEDURE DIVISION	Prozedurteil

Programmteilüberschrift

Division Header

Eine Kombination von Wörtern, gefolgt von einem Punkt und einem Leerzeichen, die den Anfang einer DIVISION bezeichnet. Die Programmteilüberschriften lauten:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING {datenname-1}...].
```

Programmtext-Bereich

Program-text Area

Ab Spalte 8 bis Spalte 72 (einschließlich) im COBOL-Fixed-Form-Referenzformat.

Programmzeiger

Program-pointer

Ein Programmzeiger ist ein Datenelement, in dem die Adresse eines Programms gespeichert werden kann.

Prototyp

Prototype

Die Definition der Schnittstelle einer Methode bzw. eines Programms allein, dh. ohne ausführbare Anweisungen und nur mit Definitionen, die in der Schnittstelle gebraucht werden.

Prozedur

Procedure

Ein Paragraf, eine Gruppe von logisch aufeinanderfolgenden Paragrafen, ein Kapitel oder eine Gruppe von logisch aufeinanderfolgenden Kapiteln innerhalb der PROCEDURE DIVISION.

Prozedurname

Procedure-Name

Ein benutzerdefinierter Name, der zum Aufruf eines Paragrafen oder eines Kapitels in der PROCEDURE DIVISION benutzt wird und das aus einem Paragrafennamen, einem gekennzeichneten Paragrafennamen oder einem Kapitelnamen besteht.

Prozedurvereinbarungen

Declaratives

Ein Kapitel oder eine Folge von Kapiteln, die am Anfang der PROCEDURE DIVISION erklärt werden. Das erste Kapitel wird eingeleitet mit dem Schlüsselwort DECLARATIVES. Das letzte Kapitel wird abgeschlossen mit dem Schlüsselwort END DECLARATIVES. Eine Prozedurvereinbarung ist zusammengesetzt aus einer Kapitelüberschrift, gefolgt von einem USE-Übersetzungssatz, von einem, keinem oder mehreren Paragrafen.

Prozedurvereinbarungssatz

Declarative Sentence

Eine Übersetzungssteueranweisung, die eine einzelne USE-Anweisung enthält und durch einen Punkt abgeschlossen wird.

Pseudotext

Pseudo-Text

Eine Folge von Textwörtern, Kommentarzeilen oder Leerzeichen in einer Übersetzungseinheit, die von Pseudotext-Begrenzern eingeschlossen ist. Die Begrenzer gehören nicht zum Pseudotext.

Pseudotext-Begrenzer

Pseudo-Text-Delimiter

Zwei unmittelbar aufeinanderfolgende Gleichheitszeichen (==), die einen Pseudotext links und rechts begrenzen.

Quelleinheit

source unit

Eine Anweisungsfolge, die mit einer Identification Division beginnt und mit einem zugehörigen END-Eintrag schließt (kann geschachtelt sein, siehe auch „Übersetzungseinheit“).

Rechenanlagenbezeichnung

Computer-Name

Ein Systemname, der die Datenverarbeitungsanlage bezeichnet, in der das Programm übersetzt wird oder abläuft.

Rechendezimalpunkt

Assumed Decimal Point

Die Stellung des Dezimalpunktes in einem Datenfeld. Der Rechendezimalpunkt hat eine logische arithmetische Bedeutung. Die Existenz eines Druckdezimalpunktes ist nicht eingeschlossen.

Rechenvorzeichen

Operational Sign

Ein algebraisches Vorzeichen, welches mit dem Inhalt eines numerischen Datenfeldes oder mit einem numerischen Literal verbunden ist und anzeigt, ob es sich um einen positiven oder negativen Wert handelt.

Rechtsbündiges Ende

Low-Order End

Das am weitesten rechts stehende Zeichen einer Zeichenfolge.

Referenzformate

Reference Formats

Standarddarstellung der beiden möglichen Anweisungsformate in einer COBOL-Übersetzungseinheit, nämlich Fixed-Form-Referenzformat und Free-Form-Referenzformat, wobei das Fixed-Form-Referenzformat dem traditionellen COBOL-Referenzformat entspricht.

Relative Datei

Relative File

Eine Datei mit relativer Organisation.

Relative Indizierung

Bei der relativen Indizierung folgt dem Namen des Tabellenelementes ein Index in der Form von (indexname +|- ganzzahl).

Relative Organisation

Relative Organization

Eine logische Dateistruktur, in welcher jeder Satz eindeutig mit einem ganzzahligen Wert größer Null festgelegt ist, der die relative Lage des Datensatzes innerhalb der logischen Reihenfolge der Datei angibt.

Relative Satznummer

Relative Record Number

Die Nummer eines Satzes in einer relativ organisierten Datei. Sie besteht aus einem ganzzahligen, numerischen Literal.

Relative Subskribierung

Bei der relativen Subskribierung folgt dem Namen des Tabellenelements ein Subskript in Form von

```
(datenname + ganzzahl) bzw.
```

```
(datenname - ganzzahl).
```

Relativer Schlüssel

Relative Key

Ein Schlüssel, dessen Inhalt einen logischen Satz in einer relativen Datei bestimmt.

Repository

external Repository

Eine externe Bibliothek, die die Beschreibung der Schnittstellen von Programmen, Klassen und Interfaces enthält.

Reserviertes Wort

Reserved Word

Ein COBOL-Wort, das in der Liste der reservierten Wörter enthalten ist und in der COBOL-Übersetzungseinheit entsprechend den Formaten und Regeln verwendet werden kann. Es darf in den Programmen nicht als benutzerdefinierter Name oder Systemname auftreten.

Rumpfleiste

Body Group

Allgemeiner Name für eine Postenleiste, einen Gruppenkopf oder einen Gruppenfuß.

Satznummer

Record Number

siehe „Datensatznummer“

Satzzeichen

Punctuation Character

Zeichen	Bedeutung
,	Komma
;	Semikolon
.	Punkt
:	Doppelpunkt
"	Doppelhochkomma
’	Hochkomma
(	öffnende Klammer
)	schließende Klammer
'BLANK'	Leerzeichen
=	Gleichheitszeichen

Schalterzustandsbedingung

Switch-Status Condition

Eine Bedingung, die angibt, ob ein Benutzer- oder Prozessschalter in einen Ein- oder Auszustand gesetzt worden ist. Das Ergebnis eines Tests ist wahr, wenn der Schalter auf der dem Bedingungsnamen entsprechenden Stellung steht.

Schlüssel

Key

Ein Datenfeld, dessen Inhalt die Position eines Datenfeldes darstellt, oder mehrere Datenfelder, die die Reihenfolge von Daten festlegen.

Schlüsselwort

Key Word

Ein reserviertes Wort oder ein Funktionsname, dessen Anwesenheit erforderlich ist, wenn das Format, in dem das Wort vorkommt, in einer Übersetzungseinheit verwendet wird.

Schnittstelle (für Methode bzw. Programm)

Interface

Die Informationen, die zum Aufruf einer Methode bzw. eines Programms nötig sind, d.h. der Name der Methode bzw. des Programms, die definierte Reihenfolge der Parameter mit der jeweiligen Definition und Übergabeart, sowie dem Rückgabeparameter und seiner Definitionen (sofern vorhanden).

Schnittstelle (Sprachmittel)

Interface

Schnittstellendefinition

Interface Definition

Die Übersetzungseinheit, die ein Interface definiert.

Segmentnummer

Segment-Number

Ein benutzerdefinierter Name, der zur Einteilung der Kapitel in der PROCEDURE DIVISION für die Segmentierung nötig ist. Segmentnummern dürfen nur die Zeichen 0, 1, ..., 9 enthalten. Eine Segmentnummer wird durch eine einstellige oder zweistellige Zahl ausgedrückt.

Seite

Page

Ein Längsabschnitt einer Liste, die eine physische Trennung der fortlaufenden Listendaten darstellt, wobei diese Trennung auf Grund von internen Listenbedingungen und/oder externen Gegebenheiten des Listenmediums vorgenommen wird.

Seitenfuß

Page Footing

Eine Leiste, die am Ende einer Listenseite erscheint und die vor jedem Seitenwechsel, der auf Grund einer Seitenbedingung stattfindet, ausgegeben wird.

Seitenkopf

Page Heading

Eine Leiste, die zu Beginn einer Listenseite erscheint und die nach jedem Seitenwechsel, der auf Grund einer Seitenbedingung stattfindet, ausgegeben wird.

Seitenrumpf

Page Body

Der Teil einer logischen Seite, in der Zeilen beschrieben und/oder freigelassen werden können.

Sequenzielle Datei

Sequential File

Eine Datei mit sequenzieller Organisation.

Sequenzielle Organisation

Sequential Organization

Eine unveränderliche, logische Dateistruktur, in welcher jeder Satz in der Reihenfolge der Erzeugung sequenziell angeordnet ist. Die Sätze werden sequenziell in der Erzeugungsreihenfolge gelesen.

Sequenzieller Zugriff

Sequential Access

Die Methode des Lesens und Schreibens von Datensätzen einer Datei in serieller Reihenfolge; die Reihenfolge der Abarbeitung ist implizit durch die Anordnung der Datensätze in der Datei bestimmt.

Sonderregister

Special Register

Vom Compiler generierte Speicherbereiche, die bei der Verwendung bestimmter Bestandteile von COBOL die dort anfallende Information enthalten.

Sonderzeichen

Special Character

Zeichen	Bedeutung	Zeichen	Bedeutung
+	Pluszeichen	.	Punkt (Dezimalpunkt)
-	Minuszeichen	:	Doppelpunkt
*	Stern	"	Doppelhochkomma
/	Schrägstrich	’	Hochkomma
=	Gleichheitszeichen	(	öffnende Klammer
$	Währungszeichen	)	schließende Klammer
,	Komma (als Dezimalpunkt)	>	Größerzeichen
;	Semikolon	<	Kleinerzeichen

Sonderzeichenwort

Special Character Word

Ein reserviertes Wort, das ein arithmetischer Operator oder ein Vergleichszeichen ist.

Sortierdatei

Sort File

Eine Folge von Datensätzen, die mit Hilfe einer SORT-Anweisung sortiert werden. Die Sortierdatei kann nur mit der SORT-Funktion erzeugt und verwendet werden.

Sortierfolge

Collating Sequence

Eine definierte Reihenfolge der Zeichen, die in einer Datenverarbeitungsanlage zum Sortieren, Mischen und Vergleichen zugelassen sind.

Sortier-Misch-Dateierklärung

Sort-Merge File Description Entry

Eine Dateierklärung in der FILE SECTION der DATA DIVISION, die aus der Stufenbezeichnung SD, einem Dateinamen und einer Folge von Dateiklauseln besteht.

Spalte

Column

Eine Zeichenposition innerhalb einer Druckzeile. Die Spalten sind von 1 an aufwärts nummeriert, angefangen bei der am weitesten links stehenden bis zu der am weitesten rechts stehenden Zeichenposition des druckfähigen Feldes der Druckzeile.

Stark typisiert

Strongly Typed

In der Definition des zugehörigen Typs ist das Wort STRONG angegeben.

Strukturabhängige Datenfelder

Contiguous Items

Datenfelder, die durch aufeinanderfolgende Erklärungen in der DATA DIVISION beschrieben sind und in einer bestimmten hierarchischen Beziehung zueinander stehen.

Strukturunabhängige Datenfelder

Noncontiguous Items

Datenelemente in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION, die keine hierarchische Beziehung zu anderen Datenfeldern haben.

Stufenbezeichner

Level Indicator

Zwei alphabetische Zeichen, die einen bestimmten Typ einer Datei angeben. Mögliche Stufenbezeichner sind: FD, RD, SD und DB.

Stufennummern

Level-Numbers

Ein- oder zweistellige Ziffern, die im Falle 1 bis 49 die hierarchische Struktur eines logischen Satzes angeben, im Falle der Stufennummern 66, 77, 88 besondere Eigenschaften einer Datenerklärung bezeichnen.

Subskribierter Datenname

Subscripted Data-Name

Ein Bezeichner, der aus einem Datennamen, gefolgt von einem oder mehreren Subskripten, die in runde Klammern eingeschlossen sind, besteht.

Subskript

Subscript

Das Subskript ist eine ganze Zahl, ein Datenname oder ein arithmetischer Ausdruck. Der Wert der Zahl bzw. des Datennamens bzw. des arithmetischen Ausdrucks gibt die Nummer eines Tabellenelements oder eines der Datenfelder, die diesem Tabellenelement untergeordnet sind, an. Ein Subskript kann auch das Wort ALL sein, wenn der subskribierte Bezeichner als Funktionsargument verwendet wird.

Summenzähler

Sum Counter

Ein mit Vorzeichen versehenes numerisches Datenfeld einer SUM-Klausel in der REPORT SECTION der DATA DIVISION. Dieses Feld, d.h. der Summenzähler, wird vom Listenprogrammsteuersystem zur Summierung benutzt.

Symbolisches Zeichen

Symbolic Character

Ein benutzerdefinierter Name, der eine vom Benutzer definierte figurative Konstante bezeichnet.

Systemname

System-Name

Ein COBOL-Wort, das als Schnittstelle zum Betriebssystem verwendet wird.

Tabelle

Table

Logisch aufeinanderfolgende Datenelemente, die in der DATA DIVISION mit der OCCURS-Klausel definiert werden.

Tabellenelement

Table Element

Ein Datenfeld, das zu einer Folge sich wiederholender Datenfelder gehört, die eine Tabelle bilden.

Teilfeldselektion

Reference Modification

Definierung eines Datenfeldes durch die Angaben der Position des Anfangszeichens und der Länge des Datenfeldes.

Teilfeldselektor

Reference-Modifier

Eine syntaktisch korrekte Kombination von Zeichen und Trennzeichen, die ein eindeutiges Datenfeld definiert. Der Teilfeldselektor besteht aus

- der begrenzenden linken Klammer,
- der Angabe, ab welcher Zeichenposition des Datenfeldes die Teilfeldselektion beginnen soll,
- dem Trennzeichen Doppelpunkt,
- der Angabe, wie lang das Teilfeld sein soll und
- der begrenzenden rechten Klammer.

Testhilfezeile

Debugging Line

Jede Zeile, die durch ein D im Anzeigenbereich (Spalte 7 des COBOL Fixed-Formats) gekennzeichnet ist.

Textname

Text-Name

Ein benutzerdefinierter Name, der einen Bibliothekstext bezeichnet.

Textwort

Text-Word

Ein Zeichen oder eine Zeichenfolge im Programmtext-Bereich eines COPY-Bibliothekstextes oder einer Übersetzungseinheit. Jedes der folgenden ist ein Textwort:

1. Trennsymbole, außer: Leerzeichen, Pseudotext-Begrenzern und öffnende bzw. schließende Literalbegrenzer. Der Doppelpunkt und die öffnende bzw. schließende runde Klammer werden bei Verwendung außerhalb von alphanumerischen oder nationalen Literalen immer als Trennsymbole behandelt.
2. Alphanumerische und nationale Literale einschließlich ihrer Literalbegrenzer.
3. Jede andere von Trennsymbolen begrenzte Zeichenfolge, außer Kommentaren und dem Wort „COPY“.

Trennsymbol

Separator

Ein Zeichen, mit dem Zeichenfolgen von einander getrennt werden.

Typ (für Typvereinbarung)

Type

Eine Vorlage, die alle Eigenschaften der Datenbeschreibung und ihrer untergeordneten Datenfelder enthält.

Typbezogener Zeiger

Restricted Data Pointer

Ein Zeigerdatenfeld, das auf Daten eines bestimmten Typs eingeschränkt ist.

Typname

Type-Name

Ein benutzerdefinierter Name, der einen Typ kennzeichnet, der in einer Datenerklärung der DATA DIVISION beschrieben ist.

Übersetzungsgruppe

Compilation Group

Eine Folge von Übersetzungseinheiten, die zusammen übersetzt werden.

Übersetzungseinheit

Compilation Unit

Eine Quelleinheit, die nicht in anderen Quelleinheiten geschachtelt ist (Programm-Prototyp, Progamm-Definition, Klassen-Definition und Interface-Definition). Dies sind die Elemente einer Übersetzungsgruppe. Sie sind separat übersetzbar.

Übersetzungssteueranweisung

Compiler Directing Statement

Eine Anweisung, die den Compiler veranlasst, eine bestimmte Aktion auszuführen. Die COPY-, REPLACE- und USE-Anweisung sind Übersetzungssteueranweisungen.

Übersetzungszeit

Compile Time

Der Zeitraum, in dem eine Übersetzungseinheit vom Compiler übersetzt wird.

Unbedingte Anweisung

Imperative Statement

Eine Anweisung, die mit einem unbedingten Verb beginnt und angibt, dass eine Aktion unbedingt ausgeführt werden muss, oder eine bedingte Anweisung, die durch ihren expliziten Bereichsbegrenzer begrenzt ist. Eine unbedingte Anweisung kann aus einer Folge von unbedingten Anweisungen bestehen.

Unterklasse

Subclass

Eine Klasse, die von einer anderen Klasse erbt.

Unterprogramm

Subprogram

Ein durch eine CALL-Anweisung aufgerufenes Ablaufelement.

Unärer Operator

Unary Operator

Ein Plus- (+) oder Minuszeichen (-), das einer Variablen oder einer öffnenden runden Klammer in einem arithmetischen Ausdruck vorausgehen muss. Die Wirkung des Operators ist so, als ob der Ausdruck mit +1 oder -1 multipliziert würde.

Variable

Variable

Ein Datenfeld, dessen Wert während der Ausführung des Zielprogramms geändert werden kann. Eine Variable in einem arithmetischen Ausdruck muss ein numerisches Datenelement sein.

Verb

Verb

Ein COBOL-Wort, mit dem der Compiler und das Zielprogramm zu einer Aktion veranlasst werden können.

Vererbung

Inheritance

für Klassen:

Ein Konzept, bei dem die Schnittstelle und Realisierung (Code) einer oder mehrerer Klassen als Basis für eine weitere Klasse dient. Diese Unterklasse erbt dann von einer oder mehreren Oberklassen. Die Schnittstelle einer erbenden Klasse ist konform zu den Schnittstellen der geerbten Klassen.

für Interfaces:

Ein Konzept, bei dem die Beschreibung einer oder mehrerer Interfaces als Basis für ein weiteres Interface dient. Ein erbendes Interface ist konform zum geerbten Interface.

Vergleich

Relation

siehe „Vergleichsoperator“

Vergleichsbedingung

Relation Condition

Eine Bedingung, für die ein Wahrheitswert ermittelt werden kann. Sie bewirkt, dass zwei Operanden miteinander verglichen werden. Jeder dieser Operanden kann ein Bezeichner, ein Literal oder ein arithmetischer Ausdruck sein.

Vergleichsoperator

Relational Operator

Ein reserviertes Wort, ein Vergleichszeichen, eine Gruppe aufeinanderfolgender reservierter Wörter oder eine Gruppe von aufeinanderfolgenden Wörtern und Vergleichszeichen, die zur Bildung von Vergleichsausdrücken verwendet werden. Die zugelassenen Operatoren und ihre Bedeutung sind:

Vergleichsoperator	Bedeutung
IS [NOT] GREATER THAN IS [NOT] >	größer als oder nicht größer als
IS [NOT] LESS THAN IS [NOT] <	kleiner als oder nicht kleiner als
IS [NOT] EQUAL TO IS [NOT] =	gleich oder nicht gleich
IS GREATER THAN OR EQUAL TO IS >=	größer als oder gleich
IS LESS THAN OR EQUAL TO IS <=	kleiner als oder gleich

Vergleichszeichen

Relation Character

Eines der folgenden Zeichen:

Zeichen	Bedeutung
>	größer als
<	kleiner als
=	gleich
>=	größer als oder gleich
<=	kleiner als oder gleich

Verknüpfer

Connective

Ein reserviertes Wort, das folgenden Zwecken dient:

- einen Datennamen, Paragrafennamen, Bedingungsnamen oder Textnamen mit seinem Kennzeichner zu verbinden.

- zwei oder mehrere in einer Seite geschriebene Operanden zu ketten.
- bedingte Ausdrücke zu bilden (logische Bindewörter), siehe „Logischer Operator“.

Verneinte einfache Bedingung

Negated Simple Condition

Eine einfache Bedingung, die unmittelbar dem logischen Operator NOT folgt.

Verneinte zusammengesetzte Bedingung

Negated Combined Condition

Eine in Klammern eingeschlossene, zusammengesetzte Bedingung, die unmittelbar dem logischen Operator NOT folgt.

Vordefinierte Objektreferenz

predefined object reference

Ein implizit erzeugtes Datenelement, auf das einer der Bezeichner NULL, SELF oder SUPER verweist.

Vorzeichenbedingung

Sign Condition

Mit der Vorzeichenbedingung wird geprüft, ob der algebraische Wert eines Datenfeldes oder eines arithmetischen Ausdrucks kleiner als, größer als oder gleich Null ist.

Währungszeichen

Currency Symbol

Das Zeichen, das in der CURRENCY SIGN-Klausel im SPECIAL-NAMES-Paragrafen definiert ist. Ist keine CURRENCY SIGN-Klausel in der COBOL-Übersetzungseinheit vorhanden, wird das Dollarzeichen ($) als Währungszeichen verwendet.

Wahlfreier Zugriff

Random Access

Die Methode des Lesens und Schreibens von Datensätzen einer Datei in einer vom Programmierer vorgegebenen Weise.

Die Reihenfolge der Abarbeitung von Datensätzen der Datei ist durch besonders festgelegte Schlüssel, die vom Anwender zur Verfügung gestellt werden, gegeben.

Wahlwort

Optional Word

Ein reserviertes Wort, das in einem bestimmten Format allein der Verbesserung der Lesbarkeit dient. Es braucht nicht vorhanden zu sein, wenn das Format, in dem das Wort vorkommt, in einer Übersetzungseinheit verwendet wird.

Wahrheitswert

Truth Value

Darstellung eines Auswertungsergebnisses (aus einer Bedingung) in Einheiten einer der beiden Werte „wahr“ oder „falsch“.

Wort

Word

Eine Folge von maximal 31 Zeichen, die einen benutzerdefinierten Namen, einen Systemnamen, ein reserviertes Wort oder einen Funktionsnamen bildet.

XML-Datei

XML File

Eine Datei oder ein Speicherbereich, der ein XML-Dokument enthält.

Zähler

Counter

Ein Datenfeld, in dem Zahlen gespeichert oder dargestellt werden, und zwar dergestalt, dass diese Zahlen noch um den Wert einer anderen Zahl erhöht oder vermindert, geändert, auf Null gesetzt oder auf einen beliebigen positiven oder negativen Wert gebracht werden können.

Zeichen

Character

Die unteilbare Grundeinheit der Sprache.

Zeichenfolge

Character-String

Aufeinanderfolgende Zeichen, die ein COBOL-Wort, ein Literal, eine Maskenzeichenfolge oder eine Kommentareintragung bilden.

Zeiger

Pointer

Zeiger können sich auf Daten oder Programme beziehen.

Zeile

Line

siehe „Listenzeile“.

Zeilennummer

Line Number

Eine ganze Zahl, die die vertikale Position einer Listenzeile auf einer Seite anzeigt.

Zeilensequenzielle Organisation

Line Sequential Organization

Eine sequenzielle Dateiorganisation aus dem X/OPEN-Standard.

Zielprogramm

Object Program

Das maschinensprachliche Ergebnis der Übersetzung einer COBOL-Übersetzungseinheit und eines Bindelaufs.

Zugriffsart

Access Mode

Die Methode, wie auf Sätze in einer Datei zugegriffen wird.

Zusammengesetzte Bedingung

Combined Condition

Eine Bedingung, die durch Verbinden von zwei oder mehr Bedingungen mit den logischen Operatoren AND oder OR hergestellt wird.

Paragrafenüberschrift

2.2 COBOL-Notation

1. Definition eines Formates

Die bestimmte Anordnung der Elemente einer Klausel oder einer Anweisung wird als allgemeines Format bezeichnet. Eine Klausel oder eine Anweisung kann sich aus verschiedenen Elementtypen zusammensetzen.

Falls mehrere Anordnungstypen bei einer Klausel oder bei einer Anweisung erlaubt sind, wird das allgemeine Format in entsprechend durchnummerierte Formate zerlegt. Dabei ist zu beachten, dass die Klauseln in der Reihenfolge verwendet werden müssen, wie sie im Allgemeinen Format spezifiziert sind. In bestimmten Ausnahmefällen kann von dieser Regel abgewichen werden. Diese Fälle sind aber ausgewiesen.

Die richtige Benutzung, die erforderlichen Voraussetzungen für die Verwendung und die Einschränkungen werden in Regeln ausgedrückt.

2. Elemente

Die Klauseln oder Anweisungen können aus folgenden Elementtypen aufgebaut werden:

- durch Großbuchstaben dargestellte Wörter
- durch Kleinbuchstaben dargestellte Wörter
- durch Klein- und Großbuchstaben dargestellte Wörter
- Stufennummern
- eckige Klammern
- geschweifte Klammern
- Bindewörter
- Sonderzeichen

3. Wörter

Schreibweise	Bedeutung
in Großbuchstaben	ein spezielles, für COBOL reserviertes Wort.
in Großbuchstaben, unterstrichen	dieses Wort muss vom Programmierer so angegeben werden, wie es im Format auftritt; es ist ein COBOL-Schlüsselwort.
in Großbuchstaben, nicht unterstrichen	dieses Wort kann durch den Programmierer an der Stelle, wo es im Format auftritt, angegeben oder weggelassen werden; es ist ein COBOL-Wahlwort.
in Kleinbuchstaben	allgemeiner Ausdruck, um COBOL-Wörter, Literale, Maskenzeichenfolgen, Kommentare oder eine komplette syntaktische Einheit darzustellen, die vom Programmierer an der im Format gezeigten Stelle eingesetzt werden müssen. Wenn mehrere allgemeine Ausdrücke derselben Art in einem Format erscheinen, dient eine angehängte Nummer oder ein angehängter Buchstabe der eindeutigen Kennzeichnung dieses Ausdrucks für die Erklärungen.

Tabelle 1: Schreibweise der COBOL-Wörter

Ein Eintrag, der aus einem Wort oder mehreren Wörtern in Großbuchstaben besteht, die von den Wörtern „Klausel“ oder „Anweisung“ gefolgt sind, bezeichnet eine Klausel bzw. Anweisung, die an anderer Stelle im Handbuch beschrieben ist. Alle COBOL-Wörter können sowohl in Groß-/Kleinschreibung als auch nur in Kleinschreibung im Programm erscheinen.

4. Trennsymbole

Die in der folgenden Tabelle aufgeführten Trennsymbole müssen wie im Format angegeben benutzt werden.

Zeichen	Bedeutung	Zeichen	Bedeutung
'BLANK'	Leerzeichen	"	Doppelhochkomma
,	Komma	’	Hochkomma
;	Semikolon	(	runde Klammer auf
.	Punkt	)	runde Klammer zu
:	Doppelpunkt	==	Pseudotext-Begrenzer

Tabelle 2: Trennsymbole

Die Regeln zu den Trennsymbolen sind im Abschnitt „Trennsymbole" beschrieben.

5. Stufenbezeichnungen und Stufennummern

Im Format auftretende Stufen und Stufennummern müssen an entsprechender Stelle in der COBOL-Übersetzungseinheit angegeben werden. In diesem Handbuch wird die Form 01, 02, ..., 09 benutzt, um die Stufennummern 1, 2, ..., 9 anzuzeigen.

6. Eckige Klammern [ ]

Eine in eckige Klammern gesetzte Angabe eines Formats kann nach Wahl des Benutzers angegeben oder weggelassen werden. Stehen mehrere Angaben untereinander in eckigen Klammern, kann eine der Angaben ausgewählt oder es können alle weggelassen werden.

7. Geschweifte Klammern { }

Die geschweiften Klammern werden mit unterschiedlicher Bedeutung verwendet:

- Stehen mehrere Angaben innerhalb geschweifter Klammern untereinander, muss eine dieser Angaben ausgewählt werden.
- Bei einer Angabe hat die geschweifte Klammer nur die Funktion der Zusammenfassung für ein nachfolgendes Wiederholungssymbol.

8. Senkrechte Balken | |

Die senkrechten Balken sind entweder von eckigen oder geschweiften Klammern umgeben und haben folgende Bedeutung:

- Innerhalb geschweiften Klammern schließen sie Wahlangaben ein, wobei mindestens eine Angabe ausgewählt werden muss oder auch mehrere Angaben ausgewählt werden können. Die Reihenfolge der ausgewählte Angaben ist beliebig. Jede einzelne Angabe darf aber höchstens einmal verwendet werden.
- Innerhalb eckiger Klammern schließen sie Wahlangaben ein, wobei die Angaben weggelassen werden können oder auch mehrere Angaben augewählt werden können. Die Reihenfolge der ausgewählten Angaben ist beliebig. Jede einzelne Angabe darf aber höchstens einmal verwendet werden.

9. Runde Klammern ( )

In runde Klammern gesetzte Angaben eines Formats bezeichnen Tabellenelementnummern (Indizes), die zur Unterscheidung der Tabellenelemente angegeben werden müssen.

10. Wiederholungssymbol ...

Ein Wiederholungssymbol im Text zeigt an, dass es sich um Auslassung eines Wortes oder mehrerer Wörter handelt, ohne dass dadurch der Sinn verändert wird.

Ein im Format angegebenes Wiederholungssymbol zeigt an, dass die unmittelbar vorausgehende Einheit einmal oder beliebig oft wiederholt werden kann. Eine wiederholbare Einheit ist entweder ein einziges Wort oder mehrere Wörter, die durch eckige oder geschweifte Klammern zusammengefasst sind. Im letzten Fall folgt das Wiederholungssymbol unmittelbar auf eine schließende Klammer; die dazugehörige öffnende Klammer bestimmt den Anfang der zu wiederholenden Einheit.

11. Leerzeichensymbol 'BLANK'

Tritt dieses Zeichen in Beispielen und Tabellen auf, so steht es für ein Leerzeichen.

12. Sonderzeichen in Formaten

Treten die Zeichen +, -, >, <, =, >= und <= in Formaten auf, so müssen sie auch gesetzt werden, wenn dieses Format benutzt wird. Dies gilt auch, wenn diese Sonderzeichen nicht unterstrichen sind.

Beispiel 2-1

für die Anwendung der Notation

```
ADD (1)   { (2) bezeichner-1 | literal-1}(2)...(3)

      TO {bezeichner-2(4) [(5) ROUNDED](5) }...
  [ON(6) SIZE ERROR unbedingte(7)-anweisung-1]
  [NOT ON SIZE ERROR unbedingte-anweisung-2]
  [END-ADD]
```

(1) COBOL-Schlüsselwort: Muss in dieser Form angegeben werden.

(2) Geschweifte Klammern: Eine dieser Angaben muss ausgewählt werden.

(3) Wiederholungssymbol: Die vorhergehende Einheit kann einmal oder beliebig oft wiederholt werden.

(4) Eindeutige Kennzeichnung: durch angehängte Nummer oder angehängten Buchstaben.

(5) Eckige Klammern: Diese Angaben können gewählt oder weggelassen werden.

(6) Wahlwort: Kann weggelassen oder zur besseren Lesbarkeit angegeben werden.

(7) Wörter in Kleinbuchstaben: Müssen vom Programmierer eingesetzt werden.

Die folgenden Sprachelemente sind aus diesem Format abgeleitete, gültige ADD-Anweisungen; Kommas und Semikolons sind zur besseren Lesbarkeit eingefügt:

```
ADD I TO J
ADD I-1, I-2, I-3 TO I-4 ROUNDED
ADD 1 TO I-1, I-2 ROUNDED, I-3
ADD I-1 TO I-2; SIZE ERROR PERFORM ADD-ERR END-ADD
```

2.3 Referenzformate

Der aktuelle COBOL-Standard erlaubt zwei Referenzformate. Das traditionelle Referenzformat bezeichnen wir im Folgenden mit „Fixed-Form-Referenzformat“ oder kürzer „Fixed-Format“. Das neu hinzugekommene Format ist das „Free-Form-Referenzformat“, oder kürzer „Free-Format“. Das Umschalten zwischen diesen beiden Referenzformaten erfolgt mit der Direktive >>SOURCE FORMAT.

Für beide Referenzformate gilt:
Nicht getrennt, sondern jeweils in einer eigenen Zeile müssen geschrieben werden:

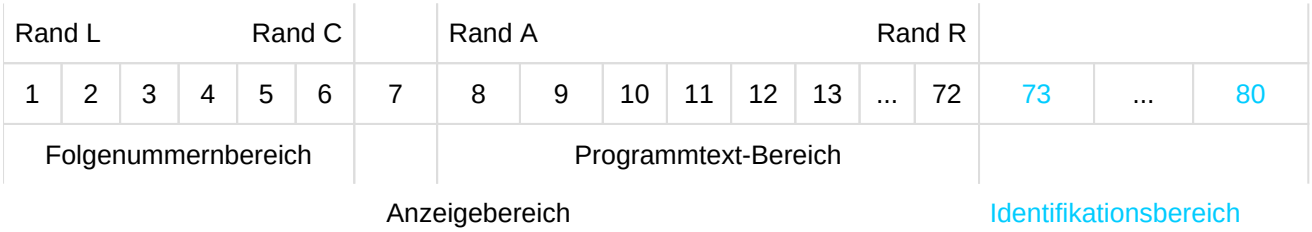
- Programmteilüberschriften
  - IDENTIFICATION DIVISION,
  - ENVIRONMENT DIVISION,
  - DATA DIVISION,
  - PROCEDURE DIVISION,
- Kapitelüberschriften
  - REPORT SECTION,
  - SUB-SCHEMA SECTION,
- PROGRAM-ID,
- CLASS-ID,
- INTERFACE-ID,
- METHOD-ID,
- Paragrafen
  - FACTORY,
  - OBJECT,
- END-Einträge.

Ein COBOL-Programm muss im maschineneigenen alphanumerischen Zeichensatz geschrieben werden.

2.3.1 Allgemeine Beschreibung des Fixed-Form-Referenzformats

Das standardisierte Fixed-Format für das Schreiben von COBOL-Programmen lässt sich anhand der Spalten (80 Zeichenstellen) einer Zeile beschreiben. Der Compiler akzeptiert in diesem Format nur COBOL-Programme, die im Referenzformat geschrieben sind, und erzeugt ein Protokoll des Programms im gleichen Format.

Eine Zeile ist wie folgt eingeteilt:



Rand L
befindet sich links von der äußersten linken Zeichenposition einer Zeile.

Rand C
befindet sich zwischen der 6. und 7. Zeichenposition einer Zeile.

Rand A
befindet sich zwischen der 7. und 8. Zeichenposition einer Zeile.

Rand R
befindet sich rechts von der äußersten rechten Zeichenposition einer Zeile, die für den Compiler noch relevant ist.

Folgenummernbereich
enthält 6 Zeichenpositionen (1-6) und befindet sich zwischen Rand L und Rand C.

Anzeigenbereich
ist die 7. Zeichenposition einer Zeile.

Programmtext-Bereich
enthält die Zeichenpositionen 8-72 und befindet sich zwischen Rand A und Rand R.

2.3.2 Regeln für die Anwendung des Fixed-Form-Referenzformats

- **Folgenummernbereich (Spalten 1-6)**

  Dieser Bereich ist für die Markierung der Zeilen eines COBOL-Programms vorgesehen.

  Der Inhalt des Folgenummernbereichs wird vom Benutzer festgelegt und darf jedes Zeichen aus dem Zeichensatz der Rechenanlage enthalten.

- **Anzeigenbereich (Spalte 7)**

  Dieser Bereich ist für die Kennzeichnung von Fortsetzungs-, Kommentar- und Testhilfezeilen vorgesehen.

  Ein Bindestrich (-) in diesem Bereich bedeutet, dass diese Zeile eine **Fortsetzungszeile** und die vorhergehende Zeile eine fortgesetzte Zeile ist (siehe unten, „Fortsetzung von Zeilen“). Falls der Anzeigenbereich keinen Bindestrich enthält, wird angenommen, dass auf das letzte Zeichen im Programmtext-Bereich (siehe unten) der vorhergehenden Zeile ein Leerzeichen folgt. Ein Stern (*) in diesem Bereich zeigt eine **Kommentarzeile** an (siehe unten, „Kommentarzeile“).

  Ein Schrägstrich (/) in diesem Bereich zeigt eine spezielle Kommentarzeile an, die einen Formularvorschub im Programm vor dem Drucken dieser Zeile auslöst.

  Ein Buchstabe D in diesem Bereich kennzeichnet eine **Testhilfezeile** (siehe unter Abschnitt „Testhilfen").

- **Programmtext-Bereich (Spalten 8-72)**

  Dieser Bereich ist reserviert für die Überschriften der vier Teile eines COBOL-Programms, der Kapitel- und der Paragrafenüberschriften, für Stufenbezeichnungen und Stufennummern. Er dient der Aufnahme aller Klauseln und Anweisungen.

- **Identifikationsbereich (Spalten 73-80)**

  Dieser Bereich kann benutzt werden, um Zeilen eines COBOL-Programms einen Namen zu geben. Dieser Bereich kann beliebige Zeichen aus dem Zeichensatz der Datenverarbeitungsanlage enthalten oder leer sein. Der Inhalt wird vom Compiler nicht ausgewertet.

- **Fortsetzung von Zeilen**

  Wenn ein Programmsatz oder eine Eintragung mehr als eine Zeile erfordert, kann eine Fortsetzung auf nachfolgenden Zeilen im Programmtext-Bereich erfolgen. Die erste Zeile heißt **fortgesetzte Zeile**, nachfolgende Zeilen heißen **Fortsetzungszeilen**. Falls ein Programmsatz oder eine Eintragung mehr als zwei Zeilen beansprucht, sind alle Zeilen, außer der ersten und letzten, sowohl fortgesetzte als auch Fortsetzungszeilen.

  Ein Wort, eine Picture-Maske oder ein Literal darf in der nächsten Zeile fortgesetzt werden. In diesen Fällen gilt:

  - Fortsetzung nichtnumerischer Literale
    Wird ein nichtnumerisches Literal auf der nächsten Zeile fortgesetzt, ist ein Bindestrich im Anzeigenbereich (Spalte 7) der Fortsetzungszeile zu setzen.
    Die Fortsetzung des Literals darf an beliebiger Stelle im Programmtext-Bereich beginnen und muss mit dem selben Anführungszeichen eingeleitet werden, das auch im öffnenden Literalbegrenzer verwendet wurde.
    Alle Leerzeichen, die am Ende der fortgesetzten Zeile oder nach dem Literalbegrenzer der Fortsetzungszeile oder vor dem das Literal abschließenden Literalbegrenzer stehen, werden als Teil des Literals betrachtet.
  - Fortsetzung von Wörtern und numerischen Literalen
    Wird ein Wort oder ein numerisches Literal auf der nachfolgenden Zeile fortgesetzt, muss ein Bindestrich im Anzeigenbereich (Spalte 7) der Fortsetzungszeile gesetzt werden, um anzuzeigen, dass das erste von Leerzeichen verschiedene Zeichen im Programmtext-Bereich der Fortsetzungszeile dem letzten von Leerzeichen verschiedenen Zeichen der fortgesetzten Zeile unmittelbar, d.h. ohne trennende Leerzeichen, folgen soll.

- **Leerzeilen**

Eine Leerzeile besteht ausschließlich aus Leerzeichen in den Spalten 7 bis einschließlich 72 oder aus gar keinen Zeichen. Eine Leerzeile darf überall innerhalb einer Übersetzungsgruppe stehen.

- **Kommentarzeilen**

  Erklärende Kommentare können an jeder Stelle einer COBOL-Quelleinheit in Form von Kommentarzeilen eingefügt werden, indem im Anzeigenbereich (Spalte 7) ein Stern oder ein Schrägstrich gesetzt wird. Jede Kombination von Zeichen aus dem Zeichensatz der Datenverarbeitungsanlage kann im Programmtext-Bereich dieser Zeilen benutzt werden. Der Inhalt der Kommentarzeilen erscheint im Protokoll des Programms, im Falle des Schrägstrichs im Anzeigenbereich zu Beginn einer neuen Seite, und hat keine Auswirkung auf das Programm.

- **Testhilfezeilen**

  Testhilfezeilen sind Zeilen, die durch ein D im Anzeigenbereich (Spalte 7) angezeigt sind (siehe unter Abschnitt „Testhilfen").

- **Pseudotext**

  Der aus Zeichenfolgen und Trennzeichen bestehende Pseudotext steht im Programmtext-Bereich. Steht ein Bindestrich im Anzeigenbereich einer Zeile, die auf einen öffnenden Pseudotextbegrenzer folgt, kann an einer beliebigen Position der ProgrammtextZeile weitergeschrieben werden. Für die Fortsetzung von Textwörtern gelten die üblichen Regeln der Fortsetzung von Zeilen.

- **Zeilenendkommentare** (Inline Comments)

  Die zwei aufeinanderfolgenden Zeichen '*>' im Programmtext-Bereich leiten einen Kommentar ein, der sich bis zum Ende der aktuellen Zeile erstreckt. Ein solcher Kommentar kann allein auf einer Zeile stehen, ihm kann aber auch beliebiger Programmtext vorangehen. Im diesem Fall muss der Zeichenfolge '*>' ein Trennsymbol vorangehen.

  Die Quelltextmanipulation behandelt solche Kommentare wie Leerzeichen.

  Zeilen mit Zeilenendkommentar dürfen nicht fortgesetzt werden.

2.3.3 Allgemeine Beschreibung des Free-Form-Referenzformats

Im Free-Format darf der Programmtext beliebig auf eine Zeile geschrieben werden. Es sind lediglich einige Regeln für Kommentare und Fortsetzungszeilen zu beachten. Insofern stellt die gesamte Zeile den Programmtext-Bereich dar.
Die maximale Länge einer Zeile ist auf 248 Zeichen begrenzt.

2.3.4 Regeln für die Anwendung des Free-Form-Referenzformats

- **Fortsetzung von Zeilen**

  Nichtnumerische Literale können über mehrere Zeilen hinweg fortgesetzt werden. Die Zeile, die so fortgesetzt wird, wird „fortgesetzte Zeile“ genannt, die darauf folgenden Zeilen sind „Fortsetzungszeilen“.

  Das fortzusetzende Literal muss mit einem schließenden Literalbegrenzer abgeschlossen werden, unmittelbar gefolgt von einem Bindestrich. Anschließend dürfen nur Leerzeichen folgen. Das erste Zeichen auf der Fortsetzungszeile, das kein Leerzeichen ist, muss dasselbe Anführungszeichen sein, das auch im öffnenden Literalbegrenzer verwendet wurde. Das folgende Zeichen wird als nächstes Zeichen des fortgesetzten Literals interpretiert.
  Wenigstens ein Zeichen des eigentlichen Literals muss jeweils auf der fortgesetzten und auf der Fortsetzungszeile zu finden sein.

  Alle Zeichen, die zu einem Indikator oder Separator gehören, der aus mehreren Zeichen besteht, müssen auf derselben Zeile zu finden sein. Sie dürfen auch nicht durch Leerzeichen voneinander getrennt sein. Ein verdoppelter Literalbegrenzer, der innerhalb eines Literals einen einzelnen Literalbegrenzer darstellt, muss ebenfalls auf einer Zeile zu finden sein.

- **Leerzeilen**
  Eine Leerzeile ist eine Zeile, die nur Leerzeichen oder gar keine Zeichen enthält. Eine Leerzeile darf überall in einer Übersetzungssgruppe stehen.

- **Kommentare**
  Ein Kommentar besteht aus dem Kommentar-Indikator („*>“), gefolgt von beliebigem Text. Alle Zeichen nach dem Kommentar-Indikator bis zum Ende der Zeile sind Kommentar.
  Jedes Zeichen aus dem Zeichensatz des Computers darf im Kommentartext verwendet werden.
  Kommentare dienen nur der Dokumentation und haben keine Auswirkung auf die Bedeutung der Übersetzungsgruppe.
  Ein Kommentar kann sowohl eine Kommentarzeile als auch ein Zeilenendkommentar sein.

  - **Kommentarzeilen**

    Eine Kommentarzeile hat als erstes Zeichen auf einer Zeile einen Kommentar-Indikator. Eine Kommentarzeile darf überall in einer Übersetzungsgruppe auftauchen.

  - **Zeilenendkommentare** (Inline Comments)

    Ein Kommentar-Indikator, dem andere Zeichen als Leerzeichen vorausgehen, bildet einen Zeilenendkommentar. Ein solcher Zeilenendkommentar darf in jeder Zeile einer Übersetzungsgruppe außer in fortgesetzten Zeilen stehen.

2.4 Sprachkonzept

In diesem Kapitel werden folgende Themen behandelt:

- COBOL-Zeichenvorrat
- Trennsymbole
- COBOL-Wörter
- Literale
- Maskenzeichenfolge
- Typen
- Null-längige Datenfelder
- Konzept der maschinenunabhängigen Datenbeschreibung
- Herstellerabhängige Darstellung und Ausrichtung von Daten

2.4.1 COBOL-Zeichenvorrat

Die grundlegende Einheit der Sprache ist das Zeichen. Der Zeichenvorrat von COBOL besteht aus 26 Großbuchstaben, 26 Kleinbuchstaben, 10 Ziffern, das Leerzeichen und Sonderzeichen.

Zeichenvorrat	Bedeutung
0 bis 9	Ziffern
A bis Z, a bis z	Buchstaben
'BLANK'	Leerzeichen
+	Pluszeichen
-	Minuszeichen (Bindestrich)
*	Stern
/	Schrägstrich
=	Gleichheitszeichen
$ bzw. €;	Währungszeichen (Dollarzeichen/Eurozeichen)
,	Komma (Dezimalpunkt)
;	Semikolon
.	Punkt (Dezimalpunkt)
:	Doppelpunkt
"	Doppelhochkomma
’	Hochkomma
(	öffnende runde Klammer
)	schließende runde Klammer
>	größer als
<	kleiner als

Tabelle 3: COBOL-Zeichenvorrat

Bei nichtnumerischen Literalen, Kommentaren und Kommentarzeilen ist der Zeichenvorrat erweitert. Er enthält dann den gesamten alphanumerischen Zeichensatz der Datenverarbeitungsanlage.

Die in Zeichenfolgen und als Trennsymbole erlaubten Zeichen werden in den nachfolgenden Abschnitten beschrieben.

2.4.2 Trennsymbole

Ein Trennsymbol besteht aus einem oder mehreren Zeichen. Jedes der folgenden ist ein Trennsymbol, außer es tritt in einem Literal oder einer PICTURE-Zeichenfolge auf:

1. Das Leerzeichen ist ein Trennsymbol. Überall, wo ein Leerzeichen als Trennsymbol oder als Teil eines Trennsymbols eingesetzt werden kann, kann auch mehr als ein Leerzeichen verwendet werden. Alle Leerzeichen, die den Trennsymbolen Komma, Semikolon oder Punkt folgen, sind Teil dieser Trennsymbole (das heißt, sie werden nicht als Trennsymbol Leerzeichen behandelt).
2. Komma und Semikolon sind nur in Verbindung mit einem unmittelbar folgenden Leerzeichen Trennsymbole. Sie können überall da zum Zwecke der besseren Lesbarkeit des Programms eingesetzt werden, wo auch das Trennsymbol Leerzeichen verwendet werden kann.
3. Der Punkt ist nur in Verbindung mit einem unmittelbar folgenden Leerzeichen ein Trennsymbol. Er darf nur verwendet werden, um einen Programmsatz abzuschließen, bzw. so, wie er im Format dargestellt ist.
4. Rechte und linke Klammern sind Trennsymbole. Sie dürfen außerhalb von Pseudotext nur paarweise als linke und rechte Klammer verwendet werden, um Subskripte, Listen von Funktionsargumenten, Teilfelder, arithmetische Ausdrücke, Bedingungen oder den Wiederholungsfaktor eines PICTURE-Symbols einzuschließen.
5. Öffnende und schließende Literalbegrenzer sind Trennsymbole. Als Anführungszeichen im Literalbegrenzer kann entweder ein Hochkomma (') oder ein Doppelhochkomma (“) verwendet werden.

   Öffnende Literalbegrenzer sind:

   - ein Anführungszeichen
   - die 2 Zeichen N', N“, X' und X“
   - die 3 Zeichen NX' und NX“

   Schließende Literalbegrenzer sind:

   - ein Hochkomma, wenn der öffnende Literalbegrenzer ein Hochkomma verwendet hat
   - ein Doppelhochkomma, wenn der öffnende Literalbegrenzer ein Doppelhochkomma verwendet hat

   Unmittelbar vor einem öffnenden Literalbegrenzer muss entweder ein Leerzeichen, eine linke Klammer oder ein öffnender Pseudotext-Begrenzer stehen; unmittelbar nach einem schließenden Literalbegrenzer muss eines der Trennsymbole Leerzeichen, Komma, Semikolon, Punkt, rechte Klammer oder schließender Pseudotext-Begrenzer stehen. Trennsymbole, die dem öffnenden Literalbegrenzer unmittelbar vorausgehen bzw. dem schließenden Literalbegrenzer unmittelbar nachfolgen, sind kein Bestandteil des Trennsymbols.
6. Pseudotext-Begrenzer sind Trennsymbole. Unmittelbar vor einem öffnenden Pseudotext-Begrenzer muss ein Leerzeichen stehen; unmittelbar nach einem schließenden Pseudotext-Begrenzer muss eines der Trennsymbole Leerzeichen, Komma, Semikolon oder Punkt stehen.
   Pseudotext-Begrenzer dürfen nur paarweise zur Begrenzung von Pseudotext verwendet werden.
7. Der Doppelpunkt ist ein Trennsymbol und muss angegeben werden, falls er in den allgemeinen Formaten verlangt ist.
8. Das Trennsymbol Leerzeichen kann unmittelbar vor allen Trennsymbolen stehen, außer
   - Die Regeln des Referenzformats lassen kein voranstehendes Trennsymbol zu.
   - Vor dem schließenden Literalbegrenzer. Ein ihm vorhergehendes Leerzeichen wird als Teil des nichtnumerischen Literals und nicht als Trennsymbol betrachtet.
9. Das Trennsymbol Leerzeichen kann unmittelbar jedem Trennsymbol außer dem öffnenden Literalbegrenzer folgen. Ein Leerzeichen nach dem öffnenden Literalbegrenzer wird als Teil des nichtnumerischen Literals und nicht als Trennsymbol betrachtet.

2.4.3 COBOL-Wörter

Ein Wort besteht aus 1-31 Zeichen des folgenden Zeichenvorrats:

A-Z, a-z, 0-9, - (Bindestrich)
Groß- und Kleinbuchstaben werden als gleich angesehen.

Ein Wort darf nicht mit einem Bindestrich beginnen oder enden. Es darf keine Leerzeichen enthalten und muss mindestens einen Buchstaben enthalten.

Wörter werden in vier Kategorien eingeteilt:

- Programmiererwörter
- Systemnamen
- Reservierte Wörter
- Funktionsnamen

1. Benutzerdefinierte Namen

Ein benutzerdefinierter Name ist ein COBOL-Wort, das vom Programmierer entsprechend

dem Format einer Klausel oder Anweisung anzugeben ist. Es bezieht sich auf einzelne Einheiten von Daten während der Verarbeitung des Programmes. Nachfolgend werden die Arten der in COBOL-Programmen verwendeten benutzerdefinierten Namen beschrieben und die Regeln zum Schreiben dieser Namen angegeben.

Die Arten von benutzerdefinierten Namen werden in Tabelle 4 aufgeführt und erklärt.

Alle benutzerdefinierten Namen, ausgenommen Segmentnummern und Stufennummern, müssen eindeutig sein, entweder dadurch, dass keine anderen benutzerdefinierten Namen in der selben Übersetzungseinheit die gleiche Buchstaben- bzw. Satzzeichenanordnung haben, oder dadurch, dass sie gekennzeichnet sind.

Mit Ausnahme von Paragrafenname, Kapitelname, Stufennummer und Segmentnummer müssen alle benutzerdefinierten Namen mindestens ein alphabetisches Zeichen enthalten. Segmentnummern und Stufennummern können identisch sein mit anderen Segmentnummern oder Stufennummern und auch mit Paragrafennamen und Kapitelnamen.

alphabetname	Name eines Alphabets im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION, der mit einem Zeichensatz und/oder Sortierfolge verknüpft ist.
bedingungsname	Ein Name, dem ein Wert, eine Gruppe von Werten oder ein Wertebereich, die ein Datenfeld annehmen kann, zugewiesen ist (also eine Bedingung des Datenfeldes). Ein Bedingungsname wird durch den Eintrag einer Stufennummer 88 in der FILE SECTION, LINKAGE SECTION, WORKING-STORAGE SECTION oder LOCAL-STORAGE SECTION erklärt.
bibliotheksname	Name einer COBOL-Übersetzungseinheit-Bibliotheksdatei, die mehrere Texte mit verschiedenen Namen enthalten kann.
dateiname	Ein Name, der einer oder mehreren Gruppen von Ein- oder Ausgabedateien zugewiesen ist. Ein Dateiname wird durch sein Auftreten in der SELECT-Klausel des FILE CONTROL-Paragrafen erklärt und als Name eines FD-Eintrages verwendet.
	Ein besonderer Dateiname ist ein Sortierdateiname, der eine Sortierdatei bezeichnet. Ein Sortierdateiname wird durch sein Auftreten in der SELECT-Klausel des FILE CONTROL-Paragrafen erklärt und als Name eines SD-Eintrages verwendet.
datenname	

	Ein Name, der ein Datenfeld in der DATA DIVISION bezeichnet. Ein Datenname wird durch sein Auftreten in einer Datenerklärung definiert.
datensatzname	Der Name eines Datensatzes. Ein Datensatz wird durch einen 01-Stufeneintrag in der FILE SECTION, LINKAGE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder in der SUB-SCHEMA SECTION erklärt.
indexname	Ein Name eines Index für eine bestimmte Tabelle. Ein Indexname wird durch sein Auftreten in der INDEXED BY-Angabe der OCCURS-Klausel erklärt.
interfacename	Ein vom Benutzer definierter Name, der eine Schnittstelle bezeichnet.
kapitelname	Ein Kapitelname wird zur Benennung eines Kapitels in der PROCEDURE DIVISION verwendet. Kapitelnamen werden in den Programmtext-Bereich geschrieben, gefolgt vom Wort SECTION.
klassenname (objektorientiert)	Ein vom Benutzer definierter Name, der eine Klasse bezeichnet.
klassenname	Ein Name, durch den in der CLASS-Klausel des SPECIAL-NAMES-Paragrafen in der ENVIRONMENT DIVISION ein vom Programmierer festgelegter Zeichenvorrat benannt wird. In der Klassenbedingung kann auf diesen Klassennamen Bezug genommen werden.
listenname	Der Name einer Liste. Ein Listenname wird durch sein Auftreten in der REPORT-Klausel eines FD-Eintrags erklärt und zur Benennung eines RD-Eintrags in der REPORT SECTION verwendet.
merkname	Ein festgelegter Name, falls der Programmierer ihn mit einem bestimmten Herstellernamen im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION verknüpft.
methodenname	Ein vom Benutzer definierter Name, der eine Methode kennzeichnet. Der Methodenname wird durch seine Verwendung im METHOD-ID- Paragrafen der IDENTIFICATION DIVISION erklärt.
paragrafenname	Ein Paragrafenname wird zur Benennung eines Paragrafen in der PROCEDURE DIVISION verwendet. Paragrafennamen müssen im Programmtext-Bereich stehen.
parametername	Ein Parametername bezeichnet einen formalen Parameter einer parametrisierten Klasse oder eines parametrisierten Interface.
programmname	Der Name zur Bezeichnung des Programms. Der Programmname wird durch seine Verwendung im PROGRAM-ID-Paragrafen der IDENTIFICATION DIVISION erklärt. Er kann auch in einer CALL-Anweisung des entsprechenden aufrufenden Programms auftreten.
programmprototypname	Ein vom Benutzer definierter Name, der einen Programmprototypen bezeichnet.
segmentnummer	Eine Nummer zum Einordnen von Kapiteln in der PROCEDURE DIVISION für die Segmentierung. Sie ist durch Verwendung in der Kapitelüberschrift erklärt.
symbolisches-zeichen	Ein Name für eine figurative Konstante, die der Benutzer in der SYMBOLIC-CHARACTERS-Klausel des SPECIAL-NAMES-Paragrafen definiert.
stufennummer	Eine Stufennummer bezeichnet die Lage eines Datenfeldes in der Strukturhierarchie eines Datensatzes oder beschreibt besondere Eigenschaften

	einer Datenerklärung. Stufennummern sind durch ihr Auftreten in einer Datenerklärung definiert.
textname	Name eines Eintrags in der COBOL-Übersetzungseinheit-Bibliothek. Der Eintrag wird durch die COPY-Anweisung aus der Bibliothek übernommen.
typname	Ein benutzerdefinierter Name, der einen Typ bezeichnet, der in einer Datenerklärung der DATA DIVISION beschrieben ist.

Tabelle 4: COBOL benutzerdefinierte Namen

2. Systemnamen

Ein Systemname ist ein COBOL-Wort, das als Schnittstelle zum Betriebssystem verwendet

wird. Systemnamen werden durch den Hersteller festgelegt und können zwischen den Übersetzern der verschiedenen Hersteller abweichend sein. Für den Programmierer sehen Systemnamen eines bestimmten Compilers aus wie reservierte Wörter.

Die Systemnamen für den COBOL2000 sind:

Rechenanlagenbezeichnung	in den Paragrafen SOURCE-COMPUTER und OBJECT-COMPUTER,
Herstellername	im SPECIAL-NAMES-Paragrafen und in der ASSIGN-Klausel.
Aufruf-Konvention	in der CALL-CONVENTION-Direktive

3. Reservierte Wörter

Die COBOL-Sprache umfasst eine feste Anzahl reservierter Wörter, die COBOL-Wörter.

Ein reserviertes Wort hat einen besonderen Zweck und darf nur in dem in den Formaten angegebenen Zusammenhang verwendet werden; es darf nicht als benutzerdefinierter Name oder Systemname in der Übersetzungseinheit auftreten.

Eine vollständige Liste der reservierten Wörter ist in nachfolgenden Seiten angegeben. Alle reservierten Wörter in dieser Liste, die mit einem * versehen sind, werden nur im Falle einer Übersetzung mit DML-Anweisungen (DML = Data Manipulation Language) als reservierte Wörter behandelt, ansonsten können sie als Programmiererwörter verwendet werden. Um eine Übersetzung mit DML-Anweisungen handelt es sich, wenn in der DATA DIVISION eines Programms eine

SUB-SCHEMA SECTION

vorkommt (siehe UDS/SQL-Beschreibung [6]).
Es gibt drei Arten von reservierten Wörtern:

- Pflichtwörter (required words)
- Wahlwörter (optional words)
- Sonderzweckwörter (special purpose words)

- **Pflichtwörter**

  Ein Pflichtwort ist ein reserviertes Wort, das vorhanden sein muss, wenn das Format, in dem das Wort vorkommt, in der Übersetzungseinheit verwendet ist.

  Es gibt zwei Typen von Pflichtwörtern:

  Schlüsselwörter

Innerhalb eines jeden Formates sind solche Wörter in Großbuchstaben geschrieben und unterstrichen. Schlüsselwörter dürfen nur in den genannten Formaten verwendet werden. Die Schlüsselwörter lassen sich wiederum unterteilen in

- Verben wie ADD, READ und CALL,
- notwendige Wörter, die in Anweisungs- und Eintragungsformaten vorkommen,
- Wörter, die eine besondere funktionelle Bedeutung haben, wie z.B. NEGATIVE, SECTION etc.

Einige Schlüsselwörter können abgekürzt werden (z.B. PIC für PICTURE).

Sonderzeichenwörter

sind arithmetische Operatoren und Vergleichszeichen (siehe Abschnitt „Begriffserklä-rungen").

- **Wahlwörter**
  Innerhalb eines jeden Formates werden Wörter in Großbuchstaben, die nicht unterstri-chen sind, Wahlwörter genannt. Diese kann der Benutzer nach Belieben verwenden. Die Bedeutung der COBOL-Anweisung ändert sich nicht, unabhängig davon, ob ein Wahlwort in der Anweisung angegeben ist oder nicht. Ein Wahlwort darf jedoch nicht falsch geschrieben oder durch ein anderes Wort ersetzt werden.
- **Sonderzweckwörter**
  Es gibt zwei Typen von Sonderzweckwörtern:
  - Sonderregister
  - Figurative Konstanten

  Sonderregister

  Sonderregister sind Datenfelder, in denen bei Verwendung bestimmter Bestandteile von COBOL die dort angefallene Information abgelegt wird. Die Attribute dieser Sonderregister sind vorgegeben. Jedes Register hat einen festen Namen. Deshalb braucht der Programmierer diese Register in der DATA DIVISION nicht zu erklären. Die Sonderregister sind in Tabelle 5 aufgezeigt.

Sonderregistername	Beschreibung	Verwendung
TALLY	5-ziffriges Datenfeld ohne Vorzei-chen mit COMPUTATIONAL-Angabe (siehe Abschnitt „USAGE-Klausel")	TALLY kann überall dort verwendet werden, wo ein Datenfeld mit ganzzahligen Wert auftreten kann. Falls z.B. der aktuelle Wert von TALLY 3 ist, dann sind die folgenden Anweisungen gleichwertig:ADD 3 TO ALPHA.ADD TALLY TO ALPHA.
LINE-COUNTER PAGE-COUNTER PRINT-SWITCHCBL-CTR	Verwendet vom Listenprogramm-Teil (siehe Kapitel „Listenprogramm (Report-Writer)").	Siehe Abschnitt „Sonderregister des Listenprogramms".
LINAGE-COUNTER	4 Byte langes Datenfeld, das eine vorzeichenlose Ganzzahl enthält, die maximal den Wert von ganzzahl-1 oder datenname-	Für jede Datei, deren Dateierklärung eine LINAGE-Klausel enthält, wird vom Compiler ein LINAGE-COUNTER-Sonderregister generiert (siehe Abschnitt „LINAGE-Klausel").

	1 in der LINAGE-Klausel annehmen kann.	
RETURN-CODE	8-ziffriges Datenfeld mit Vorzeichen mit COMPUTATIONAL-Angabe und SYNCHRONIZED-Angabe (entspricht PIC S9(8) COMP-5 SYNC)	Dieses Datenfeld existiert nur einmal pro Ablaufeinheit. Der Anwender kann das Feld zum Informationsaustausch zwischen getrennt übersetzten, aber zum Zielprogramm gebundenen COBOL-Moduln nutzen. Außerdem kann in diesem Feld der Returnwert eines fremdsprachigen Unterprogramms hinterlegt werden. Bei Beendigung eines COBOL-Unterprogramms kann der Inhalt des Feldes dem aufrufenden fremdsprachigen Programm als Funktionswert zur Verfügung gestellt werden. Ist nach Ausführung von STOP RUN der Inhalt des RETURN-CODE-Sonderregisters ungleich Null, wird das Betriebssystem über die abnormale Beendigung des Programms informiert.
SORT-RETURNSORT-FILE-SIZE SORT-CORE-SIZESORT-MODE-SIZESORT-EOWSORT-CCSN	Verwendet vom Sortierteil (siehe Abschnitt „Sortieren von Datensätzen").	Siehe Abschnitt „Sonderregister für Dateien-SORT".
XML-EVENT XML-CODE XML-TEXT XML-NTEXT XML-NAMESPACE XML-NNAMESPACE XML-NAMESPACE-PREFIX XML-NNAMESPACE-PREFIX	Verwendet vom XML-Teil (siehe Abschnitt „Sprachmittel zur Verarbeitung von XML").	Siehe Abschnitt „Sonderregister für XML PARSE-Anweisung".

Tabelle 5: COBOL-Sonderregister

Reservierte Wörter

Die folgende Übersicht enthält alle **reservierten Wörter.**

Ein '*' vor dem Wort bedeutet, dass dieses Wort nur im Falle einer Übersetzung mit DML-Anweisungen (DML = Data Manipulation Language) als reserviertes Wort behandelt wird. Ansonsten kann es als benutzerdefinierter Name verwendet werden. Um eine Übersetzung mit DML-Anweisungen handelt es sich, wenn die SUB-SCHEMA SECTION angegeben ist.

Ein '#' vor dem Wort bedeutet, dass dieses Wort nicht als reserviertes Wort behandelt wird, wenn bei der SOURCE-PROPERTIES-Option ENABLE-KEYWORDS = *COBOL85 gesetzt ist.

<	#B-AND	COMMUNICATION	DEBUG-CONTENTS
<=	#B-NOT	COMP	DEBUG-ITEM
+	#B-OR	COMP-1	DEBUG-LINE
*	#B-XOR	COMP-2	DEBUG-NAME
**	#BASED	COMP-3	DEBUG-SUB-1
-	BEFORE	COMP-5	DEBUG-SUB-2
/	BEGINNING	COMPUTATIONAL	DEBUG-SUB-3

```
>
>=
:
=
ACCEPT
ACCESS
#ACTIVE-CLASS
ADD
#ADDRESS
ADVANCING
AFTER
ALL
#ALLOCATE
ALPHABET
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTER
ALTERNATE
AND
ANY
#ANYCASE
ARE
AREA
AREAS
#AS
ASCENDING
ASSIGN
AT
AUTHOR

END-IF
#END-INVOKE
END-MULTIPLY
END-OF-PAGE
#END-OPEN
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING

BINARY
#BINARY-CHAR
#BINARY-DOUBLE
#BINARY-LONG
#BINARY-SHORT
#BIT
BLANK
BLOCK
#BOOLEAN
BOTTOM
BY
BYTE-LENGTH

CALL
CANCEL
*CASE
CBL-CTR
CF
CH
CHARACTER
CHARACTERS
CHECKING
CLASS
#CLASS-ID
CLOCK-UNITS
CLOSE
CODE
CODE-SET
#COL
COLLATING
#COLS
COLUMN
#COLUMNS
COMMA
COMMIT
COMMON

FOR
#FORMAT
FREE
FROM
FUNCTION
#FUNCTION-ID

GENERATE
GET
GIVING
GLOBAL
GO
GOBACK

COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-5
COMPUTE
#CONDITION
CONFIGURATION
*CONNECT
#CONSTANT
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONVERTING
COPY
CORR
CORRESPONDING
COUNT
CREATING
#CRT
CURRENCY
*CURRENT
#CURSOR

DATA
#DATA-POINTER
*DATABASE-EXCEPTION
DATABASE-KEY
DATABASE-KEY-LONG
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
*DB
DE

*KEEP
KEY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
*LIMITED
LIMITS
LINAGE

DEBUGGING
DECIMAL-POINT
DECLARATIVES
#DEFAULT
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DETAIL
DISABLE
DISC
*DISCONNECT
DISPLAY
DIVIDE
DIVISION
#DOCUMENT
DOWN
*DUPLICATE
DUPLICATES
DYNAMIC

EBCDIC
#EC
ELSE
*EMPTY
ENABLE
END
END-ACCEPT
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DISPLAY
END-DIVIDE
END-EVALUATE

#NULL
NUMBER
NUMERIC
NUMERIC-EDITED

#OBJECT
OBJECT-COMPUTER
#OBJECT-REFERENCE
*OCCURENCE
OCCURS
OF
OFF
OMITTED
```

END-SUBTRACT
END-UNSTRING
END-WRITE
#END-XML
ENDING
ENTRY
ENVIRONMENT
#EO
EOP
EQUAL
ERASE
ERROR
*ESCAPE
EVALUATE
EVERY
EXCEPTION
#EXCEPTION-OBJECT
*EXCLUSIVE
EXIT
EXTEND
EXTENDED
EXTERNAL

#FACTORY
FALSE
FD
*FETCH
FILE
FILE-CONTROL
FILLER
FINAL
*FIND
*FINISH
FIRST
#FLOAT-EXTENDED
#FLOAT-LONG
#FLOAT-SHORT
FOOTING

#PROPERTY
*PROTECTED
#PROTOTYPE
PURGE

QUOTE
QUOTES

#RAISE
#RAISING
RANDOM
RD

GREATER
GROUP
#GROUP-USAGE

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
ID
IDENTIFICATION
#IDENTIFIED
IF
IGNORING
IN
*INCLUDING
INDEX
INDEXED
INDICATE
#INHERITS
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
#INTERFACE
#INTERFACE-ID
INTO
INVALID
#INVOKE
IS

JUST
JUSTIFIED

RETURN
#RETURNING
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROLLBACK
ROUNDED
RUN

LINE
LINE-COUNTER
LINES
LINKAGE
#LOCAL-STORAGE
#LOCALE
LOCK
LOW-VALUE
LOW-VALUES

*MASK
*MATCHING
*MEMBER
*MEMBERS
*MEMBERSHIP
MEMORY
MERGE
MESSAGE
#METHOD
#METHOD-ID
#MINUS
MODE
*MODIFY
MODULES
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY

#NATIONAL
#NATIONAL-EDITED
NATIVE
NEGATIVE
#NESTED
NEXT
NO
NOT

SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STATUS
STOP
*STORE
STRING
*SUB-SCHEMA
SUBTRACT
SUM

ON
OPEN
OPTIONAL
#OPTIONS
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
#OVERRIDE
*OWNER

PACKED-DECIMAL
PADDING
PAGE
PAGE-COUNTER
PERFORM
*PERMANENT
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
#PRESENT
PRINT-SWITCH
PRINTING
*PRIOR
PROCEDURE
PROCEED
PROGRAM
PROGRAM-ID
#PROGRAM-POINTER

UNIT
UNITS
#UNIVERSAL
#UNLOCK
UNSTRING
UNTIL
UP
*UPDATE
UPON
USAGE
*USAGE-MODE
USE

READ
*READY
*REALM
*REALM-NAME
RECEIVE
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REFERENCE
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPEATED
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
#REPOSITORY
RERUN
RESERVE
RESET
*RESULT
#RESUME
*RETAINING
*RETRIEVAL
#RETRY
SAME
#SCREEN
SD
SEARCH
SECTION
SECURITY
SEGMENT-LIMIT
SELECT
*SELECTIVE
#SELF
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
*SET-SELECTION
*SETS
#SHARING
SIGN
SIZE
SORT
SORT-MERGE
SORT-TAPE
SORT-TAPES
*SORTED
SOURCE
SOURCE-COMPUTER
#SOURCES
SPACE
SPACES
#SUPER
SUPPRESS
*SUPPRESSING
SYMBOLIC
SYNC
SYNCHRONIZED
*SYSTEM
#SYSTEM-DEFAULT

TABLE
TALLY
TALLYNG
TAPE
TAPES
*TENANT
TERMINAL
TERMINATE
TEST
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TRUE
TRY
TYPE
#TYPEDEF
#USER-DEFAULT
USING

#VAL-STATUS
#VALID
#VALIDATE
#VALIDATE-STATUS
VALUE
VALUES
VARYING
#VERSION-XML
*VIA

WHEN
WITH
*WITHIN
WORDS
WORKING-STORAGE
WRITE

#XML

ZERO
ZEROES
ZEROS

Reservierte Wörter für Compiler-Direktiven

ALL
AND
AS

B-AND
B-NOT
B-OR
B-XOR
BYTE-LENGTH

CALL-CONVENTION
CHECKING
COBOL

DE-EDITING
DEFINE
DEFINED
LEAP-SECOND
LESS
LISTING
LOCATION

MOVE

NOT
NUMVAL

OFF
ON
OR
OTHER
OVERRIDE

PAGE

```
DIVIDE

ELSE
END-IF
END-EVALUATE
EQUAL
EVALUATE

FIXED
FLAG-85
FLAG-NATIVE-ARITHMETIC
FORMAT
FREE
FUNCTION-ARGUMENT

GREATER

IF
IMP
IS

PARAMETER
PROPAGATE

SET
SIZE
SOURCE

THAN
THROUGH
THRU
TO
TRUE
TURN

WHEN

ZERO-LENGTH
```

Zusätzlich sind alle Ausnahmesituationsnamen aus der Tabelle 45 im Abschnitt "Ausnahmesituationen und Ausnahmezustände" reservierte Wörter für Compiler-Direktiven.

Kontext-sensitive Wörter

```
ALIGNED
ARITHMETIC
ATTRIBUTE
AUTO
AUTOMATIC

BACKGROUND-COLOR
BELL
BLINK

CENTER
CHECK
CLASSIFICATION
CYCLE

DISCARD
DTD

ELEMENT
ENTRY-CONVENTION
EOL
EOS
EXPANDS

FOREGROUND-COLOR

ONLY

PARAGRAPH
PARSE
PREVIOUS
PROCESSING
RAW
RECURSIVE
RELATION
REQUIRED
RETURN-CODE
REVERSE-VIDEO

SCHEMA
SECONDS
SECURE
SIGNED
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
STACK
STATEMENT
```

```
FOREVER
FULL

HIGHLIGHT

IMPLEMENTS
INITIALIZED
INTRINSIC

LINAGE-COUNTER
LOCALIZE
LOWLIGHT

MANUAL

NAMESPACE
NONE
NORMAL
NUMBERS

STEP
STRONG
SYMBOL

UCS-2
UCS-4
UNDERLINE
UNSIGNED
UTF-16
UTF-8

VALIDITY

YYYYDDD
YYYYMMDD
```

4. Funktionsnamen

Ein Funktionsname ist ein Wort aus einer besonderen Liste von Wörtern, die in einem COBOL-Programm verwendet werden dürfen. Dasselbe Wort darf, wenn es nicht als Funktionsname gebraucht wird, als benutzerdefinierter Name verwendet werden (siehe Kapitel „Interne Standard-Funktionen", „Funktionsname" im Abschnitt "Allgemeines").

5. Ausnahmesituationsnamen

Ein Ausnahmesituationsname ist ein COBOL-Wort, das eine Ausnahmesituation bezeichnet (siehe Tabelle 45 im Abschnitt "Ausnahmesituationen und Ausnahmezustände").

In anderen Zusammenhängen können diese Worte als benutzerdefinierter Name verwendet werden.

2.4.4 Literale

Ein Literal ist eine Zeichenfolge, deren Wert durch die Zeichen bestimmt wird, aus denen sich das Literal zusammensetzt, oder durch ein reserviertes Wort festgelegt ist, das einer figurativen Konstanten entspricht. Jedes Literal gehört zu einer der Klassen und Kategorien: alphanumerisch, national oder numerisch.

Alphanumerische und nationale Literale bestehen aus einer Zeichenfolge, die in Literalbegrenzer eingeschlossen ist. Anführungszeichen im Literalbegrenzer können Doppelhochkommas ("...") oder Hochkommas (’...’) sein. Dabei ist zu beachten, dass im schließenden Literalbegrenzer das gleiche Anführungszeichen wie im öffnenden Literalbegrenzer verwendet wird. In einer Quelleinheit dürfen beide Arten Anführungszeichen zum Bilden von Literalen eingesetzt werden.

Hexadezimale Literale werden aus hexadezimalen Ziffern gebildet. Dies sind die Ziffern 0..9 sowie die Buchstaben A..F und a..f. Hierbei sind die korrespondierenden Klein- und Großbuchstaben (z.B. a, A) als äquivalent zu betrachten.

1. Alphanumerische Literale

Alphanumerische Literale sind von der Klasse und Kategorie alphanumerisch.

Format 1 (alphanumerisch)

```
{"zeichen-1..." | 'zeichen-1...'}
```

Syntaxregeln

1. Ein alphanumerisches Literal muss (ohne Literalbegrenzer) mindestens 1 Zeichen enthalten und darf maximal 180 Zeichen enthalten.
   Wird ein alphanumerisches Literal im Zusammenhang mit nationalen Datenfeldern verwendet, darf es maximal 90 Zeichen enthalten.
2. Wenn zeichen-1 dem Anführungszeichen aus dem Literalbegrenzer entspricht, muss zeichen-1 doppelt angegeben werden, um im Literal (einfach) dargestellt zu werden. Das verdoppelte Anführungszeichen zählt in der Länge des Literals als ein Zeichen.

Allgemeine Regel

1. Ein alphanumerisches Literal kann alle Zeichen aus dem EBCDIC-Zeichensatz enthalten. Der Wert des alphanumerischen Literals ist die Folge der einzelnen Zeichen selbst, ohne Literalbegrenzer.

Beispiel 2-2

```
’ZEICHEN’
"153.78"
"ADAM ""BDAM"" CDAM"
```

Format 2 (hexadezimal-alphanumerisch)

```
{X | x}  {"hexadezimalziffer..." | 'hexadezimalziffer...'}
```

Syntaxregeln

1. Ein hexadezimal-alphanumerisches Literal muss (ohne Literalbegrenzer) mindestens 2 hexadezimale Ziffern enthalten und darf maximal 360 hexadezimale Ziffern enthalten.Wird ein hexadezimal-alphanumerisches Literal im Zusammenhang mit nationalen Datenfeldern verwendet, darf es maximal 180 hexadezimale Ziffern enthalten.
2. Die Anzahl der hexadezimalen Ziffern muss gerade sein.

Allgemeine Regel

1. Der Wert des hexadezimalen Literals ist das Bitmuster entsprechend der Folge der hexadezimalen Ziffern.

Beispiel 2-3

```
X"12ab"
```

2. Numerische Literale

Numerische Literale sind von der Klasse und Kategorie numerisch. Es gibt zwei Arten von numerischen Literalen: Festpunktliterale und Gleitpunktliterale.

- **Numerische Festpunktliterale**

  Ein numerisches Festpunktliteral ist eine Folge von Zeichen, bestehend aus den Ziffern 0 bis 9, dem Pluszeichen, dem Minuszeichen und dem Dezimalpunkt.

  Numerische Festpunktliterale müssen entsprechend den folgenden Regeln gebildet werden:

  1. Das Literal darf 1 bis 31 Ziffern enthalten;
  2. das Literal darf nur ein Vorzeichen enthalten. Wird ein Vorzeichen verwendet, so muss es das am weitesten links stehende Zeichen des Literals (linksbündiges Ende) sein. Ein vorzeichenloses Literal wird als positiv betrachtet;
  3. Das Literal darf nur einen Dezimalpunkt enthalten. Der Dezimalpunkt kann an beliebiger Stelle im Literal auftreten, außer als das am weitesten rechts stehende Zeichen (rechtsbündiges Ende). Ein Dezimalpunkt bezeichnet die Stelle des Rechendezimalpunktes (der Rechendezimalpunkt in irgendwelchen numerischen Literalen oder Datenfeldern ist die Stelle, an der der Compiler im generierten Programm den Dezimalpunkt annimmt, ohne dafür eine Internspeicherstelle freizuhalten). Ein Literal ohne Dezimalpunkt ist ganzzahlig.

  Der Begriff **ganzzahlig** wird zur Beschreibung eines numerischen Literals verwendet, das vorzeichenlos und größer als Null ist und keine Zeichenstellen rechts vom Rechendezimalpunkt enthält.

  Beispiel 2-4

  (Das Zeichen V beschreibt hier den Rechendezimalpunkt.)

Literal	Stelle des Rechendezimalpunktes	Internes Vorzeichen	Anzahl der zugewiesenen Ziffernstellen
+123	123V	+	3
3.765	3V765	+	4
-45.7	45V7	-	3

- **Numerische Gleitpunktliterale**

  Ein numerisches Gleitpunktliteral muss das folgende Format haben:

  Mantisse Exponent

  Die Mantisse besteht aus einem wahlweise anzugebenden Vorzeichen, gefolgt von 1 bis 16 Ziffern mit einem anzugebenden Dezimalpunkt; der Dezimalpunkt kann innerhalb der Mantisse an jeder Stelle angegeben werden.

  Der Exponent besteht aus dem Symbol E, gefolgt von einem wahlweise anzugebenden Vorzeichen, gefolgt von einer oder zwei Ziffern (der Exponent Null kann als 0 oder 00 geschrieben werden).

  Das Literal darf keine Leerzeichen enthalten. Der Exponent muss unmittelbar rechts neben der Mantisse angegeben werden.

  Die Vorzeichen sind die einzigen wahlweisen Zeichen im Format. Eine vorzeichenlose Mantisse oder ein vorzeichenloser Exponent wird als positiv betrachtet.

Der Wert des Literals ist das Produkt aus der Mantisse und der 10er-Potenz, die durch den Exponenten gegeben ist.

Der Absolutwert einer Zahl, dargestellt durch ein Gleitpunktliteral, darf 7.210^{75} nicht überschreiten.

Beispiel 2-5

```
+1.5E-2 = 1.5  10-2
```

3. Nationale Literale

Nationale Literale sind von der Klasse und Kategorie national.

Format 1 (national)

```
{N | n} {"zeichen-1..." | 'zeichen-1...'}
```

Syntaxregeln

1. Ein nationales Literal muss (ohne Literalbegrenzer) mindestens 1 Zeichen enthalten und darf maximal 90 Zeichen enthalten.
2. Wenn zeichen-1 dem Anführungszeichen aus dem Literalbegrenzer entspricht, muss zeichen-1 doppelt angegeben werden, um im Literal (einfach) dargestellt zu werden. Das verdoppelte Anführungszeichen zählt in der Länge des Literals als ein Zeichen.
3. zeichen-1 muss ein EBCDIC-Zeichen sein, das ein UTF-16 Gegenstück hat.

Allgemeine Regel

1. Der Wert des nationalen Literals ist die Folge nationaler Zeichen (ohne Literalbegrenzer), die sich aus der Konvertierung der einzelnen Zeichen nach UTF-16-Darstellung ergibt.

Beispiel 2-6

```
N’TSV’
n"1860"
```

Format 2 (hexadezimal-national)

```
{N | n} {X | x} {"hexadezimalziffer-1..." | 'hexadezimalziffer-1...'}
```

Syntaxregeln

1. Ein hexadezimal-nationales Literal muss (ohne Literalbegrenzer) mindestens 4 hexadezimale Ziffern enthalten und darf maximal 360 hexadezimale Ziffern enthalten.
2. Die Anzahl der hexadezimalen Ziffern muss ein Vielfaches von 4 sein.

Allgemeine Regel

1. Der Wert des hexadezimalen Literals ist das Bitmuster entsprechend der Folge der hexadezimalen Ziffern.

Beispiel 2-7

```
NX"005400530056"
nx’0031003800360030’
```

Damit werden die gleichen Literale dargestellt wie im vorhergehenden Beispiel 2-6.

4. Figurative Konstanten

Die Werte figurativer Konstanten werden vom Compiler erzeugt und durch die in Tabelle 6 aufgeführten reservierten Wörter bezeichnet.
Syntaxregeln:

1. Die Einzahl- und Mehrzahlform der figurativen Konstanten ist gleichwertig und kann wahlweise verwendet werden.
2. Außer in der figurativen Konstanten ALL literal hat das Wort ALL keine Funkion, es dient nur der besseren Lesbarkeit.
3. Eine figurative Konstante kann überall dort verwendet werden, wo literal in einem Format vorkommt, mit den folgenden Einschränkungen:
   - Wenn literal eingeschränkt ist auf numerische Literale, ist die einzig zulässige figurative Konstante ZERO (ZEROS, ZEROES) ohne die Angabe von ALL.
   - Wenn eine Syntaxregel die Angabe figurativer Konstanten verbietet.
4. literal muss ein alphanumerisches oder nationales Literal sein, aber nicht selber wieder eine figurative Konstante.
5. symbolisches-zeichen muss in der SYMBOLIC-CHARACTERS-Klausel des SPECIAL-NAMES-Paragrafen angegeben sein.

Allgemeine Regeln

1. Wird eine der figurativen Konstanten ZERO, SPACE, HIGH-VALUE, LOW-VALUE oder QUOTE in einem Kontext verwendet, der nationale Zeichen verlangt, dann stellt die figurative Konstante einen nationalen Wert dar. In allen anderen Fällen, wo die figurative Konstante einen Zeichenwert darstellt, steht die figurative Konstante für einen alphanumerischen Wert.
2. Stellt eine figurative Konstante eine Folge von einem oder mehreren Zeichen dar, bestimmt der Compiler die Länge der Zeichenfolge durch Anwendung der folgenden Regeln der Reihe nach:
   - Wird eine figurative Konstante in einer VALUE-Klausel angegeben oder mit einem anderen Datenfeld in Verbindung gebracht (z.B. in ein anderes Datenfeld übertragen), so wird sie zunächst nach rechts so oft wiederholt, bis die so entstehende Zeichenfolge mindestens ebensoviele Zeichenpositionen enthält wie das andere Datenfeld.
     Hat diese Zeichenfolge nach Abschluss der Vervielfältigungsoperation mehr Zeichenpositionen als das andere Datenfeld, werden die überzähligen von rechts abgeschnitten.
     Hat das Datenfeld eine Länge von Null, so wird bis auf 1 Zeichen abgeschnitten. Das Erweitern bzw. Abschneiden der Zeichenfolge der figurativen Konstanten geschieht vor und unabhängig von der etwaigen Anwendung einer JUSTIFIED-Klausel auf das andere Datenfeld.
   - Wenn die figurative Konstante nicht ALL literal ist, dann hat die Zeichenfolge immer die Länge 1. Dies gilt insbesondere dann, wenn solche figurativen Konstanten in einer DISPLAY-, STOP-, STRING- oder UNSTRING-Anweisung auftreten.
   - Die Länge der Zeichenfolge ist gleich der Länge von literal.
3. Werden die figurativen Konstanten HIGH-VALUE(S) oder LOW-VALUE(S) in einer Übersetzungseinheit verwendet (außer in der ALPHABET-Klausel), hängt das aktuelle Zeichen, das mit jeder dieser Konstanten verknüpft ist, von der für das Programm angegebenen Sortierfolge des Zeichensatzes ab (siehe Abschnitt „OBJECT-COMPUTER-Paragraf", und „SPECIAL-NAMES-Paragraf").
4. Die figurative Konstante [ALL] symbolisches-zeichen stellt eines oder mehrere der Zei-chen dar, die als Wert von symbolisches-zeichen in der SYMBOLIC-CHARACTERS-Klausel des SPECIAL-NAMES-Paragrafen angegeben sind.
5. Die figurative Konstante ZERO stellt den numerischen Wert 0 dar oder ein oder mehrere Zeichen „0“, je nach Kontext, in dem sie verwendet wird.

Tabelle 6 zeigt die figurativen Konstanten und gibt Hinweise auf die Werte, die sie darstellen.

Figurative Konstante	**entsprechender Wert**	**Beispiel** [1)]
[ALL] ZERO oder [ALL] ZEROS oder [ALL] ZEROES	eine oder mehrere Wiederholungen des Zeichens 0 (X‘F0‘ bzw. X‘0030‘) oder binär Null (X‘00‘), abhängig von der Definition des Datenfeldes.	Anweisung: MOVE ZEROS TO FELD. Inhalt von FELD: • Falls FELD ein binäres Datenfeld ist: X ‘00000000‘ • Falls FELD ein externes dezimales Datenfeld ist: X‘F0F0F0F0‘ (= C‘0000‘) • Falls FELD ein internes dezimales Datenfeld ist: X‘0000000F‘.
[ALL] SPACE oder [ALL] SPACES	eine oder mehrere Wiederholungen des Zeichens Leerzeichen (X‘40‘ bzw. X‘0020‘).	Anweisung: MOVE SPACE TO FELD. Inhalt von FELD: X‘40404040‘(= C‘ËËËË‘)
[ALL] HIGH-VALUE oder [ALL] HIGH-VALUES	Ist COLLATING SEQUENCE nicht angegeben: eine oder mehrere Wiederholungen des Zeichens, das den höchsten Wert in der EBCDIC bzw. UTF-16 Sortierfolge hat (X‘FF‘ bzw. X‘FFFF‘).	Anweisung: MOVE HIGH-VALUE TO FELD. Inhalt von FELD: X‘FFFFFFFF‘(=C‘~~~~‘)
	Ist COLLATING SEQUENCE angegeben: das Zeichen, das die höchste Position in der Sortierfolge des Programms hat.	Angabe SPECIAL-NAMES-Paragraf: ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 194. Die höchste Position hat das Zeichen, das an 194. Stelle des EBCDIC-Zeichensatz steht, das Zeichen A. A wird HIGH-VALUE zugeordnet.
[ALL] LOW-VALUE oder [ALL] LOW-VALUES	Ist COLLATING SEQUENCE nicht angegeben: eine oder mehrere Wiederholungen des Zeichens, das den kleinsten Wert in der EBCDIC bzw. UTF-16 Sortierfolge hat (X‘00‘ bzw. X ‘0000‘).	Anweisung: MOVE LOW-VALUE TO FELD. Inhalt von FELD: X‘00000000‘
	Ist COLLATING SEQUENCE angegeben: das Zeichen, das die niedrigste Position in der Sortierfolge des Programms hat.	Angabe SPECIAL-NAMES-Paragraf: ALPHABET ALPHATAB IS „0“ „1“ „2“. Die niedrigste Position hat das Zeichen 0. 0 wird LOW-VALUE zugeordnet.
[ALL] QUOTE oder [ALL] QUOTES	eine oder mehrere Wiederholungen des Doppelhochkommas. Anmerkung: Das Wort QUOTE (oder QUOTES) kann nicht an Stelle eines	Anweisung: MOVE QUOTE TO FELD. Inhalt von FELD: X‘7F7F7F7F‘[2)]

	Anführungszeichens zum Begrenzen eines nichtnumerischen Literals verwendet werden.	
ALL literal	eine oder mehrere Wiederholungen der das Literal bildenden Zeichenfolge. Das Literal muss nichtnumerisch sein.	Anweisung: MOVE ALL "A" TO ALPHA. Inhalt von ALPHA: C‘AAAA‘ Anweisung: MOVE ALL "12" TO ALPHA. Inhalt von ALPHA: C‘1212‘ Anweisung: MOVE ALL "ABC" TO ALPHA. Inhalt von ALPHA: C‘ABCA‘
[ALL] symbolisches-zeichen	eine oder mehrere Wiederholungen des Zeichens, das als Wert von symbolisches-zeichen in der SYMBOLIC-CHARACTER-Klausel des SPECIAL-NAMES-Paragrafen angegeben ist.	Definition: SYMBOLIC C0 IS 193 Anweisung: MOVE ALL C0 TO ALPHA. Inhalt von ALPHA:X‘C0C0C0C0‘

Tabelle 6: COBOL figurative Konstanten und ihre Werte

[1)] In diesen Beispielen wird, wenn nichts anderes angegeben ist, angenommen, dass ALPHA und FELD jeweils als PIC X(4) definiert sind.

[2)] Kann durch Option auf X’7D’ geändert werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

2.4.5 Maskenzeichenfolge

Eine Maskenzeichenfolge enthält bestimmte Kombinationen von Zeichen aus dem COBOL-Zeichenvorrat, die als Symbole benutzt werden (siehe Abschnitt „PICTURE-Klausel").

Jedes Satzzeichen innerhalb einer Maskenzeichenfolge wird nicht als Satzzeichen gewertet, sondern als Symbol einer Maskenzeichenfolge.

2.4.6 Typen

Ein Typ ist eine Vorlage, die alle Eigenschaften der Datenbeschreibung und ihrer untergeordneten Datenfelder enthält. Die wesentlichen Eigenschaften eines Typs, der durch seinen Typnamen identifiziert wird, sind die relativen Positionen und Längen ihrer Datenelemente, die in der Typdeklaration definiert sind, sowie BLANK WHEN ZERO-Klausel, JUSTIFIED-Klausel, PICTURE-Klausel, SIGN-Klausel, SYNCHRONIZED-Klausel und USAGE-Klausel, die für diese Datenelemente angegeben sind (Einzelheiten zu den Klauseln siehe Abschnitt „Datenerklärung").

Ein Typ wird durch Angabe der TYPEDEF-Klausel definiert.
Ein Typ wird referenziert, indem in einer Datenbeschreibung eine TYPE-Klausel angegeben wird. Das typisierte Datenfeld, das damit definiert wird, hat alle Eigenschaften des angesprochenen Typs.

Datengruppen können stark oder schwach typisiert sein. Eine Datengruppe ist stark typisiert, wenn:

- entweder ihre Datenbeschreibung eine TYPE-Klausel enthält, die auf eine Typdefinition mit der Angabe STRONG verweist oder
- sie einer Datengruppe untergeordnet ist, die eine TYPE-Klausel enthält, die auf eine Typdefinition mit der Angabe STRONG verweist.

Zwei typisierte Datengruppen sind vom **gleichen Typ**, wenn

- entweder die Datenbeschreibungen TYPE-Klauseln enthalten, die äquivalente Typerklärungen referenzieren oder
- die Datenbeschreibungen untergeordnete Datengruppen in äquivalenten Typerklärungen sind, die an der gleichen Position innerhalb der Typerklärung beginnen und die gleiche Länge haben.

Zwei Typerklärungen werden als äquivalent betrachtet, wenn sie den gleichen Typnamen haben und für jedes Datenelement in der einen Typerklärung ein korrespondierendes Datenelement in der anderen Typerklärung existiert, das an der gleichen Position beginnt und die gleiche Länge hat. Jedes Paar korrespondierender Datenelemente muss dieselben BLANK WHEN ZERO-Klauseln, JUSTIFIED-Klauseln, PICTURE-Klauseln, SIGN-Klauseln, SYNCHRONIZED-Klauseln und USAGE-Klauseln haben.
Dabei gilt folgende Ausnahme:
Das Zeichen *Punkt* in der zugehörigen PICTURE Zeichenkette ist äquivalent, wenn die Angabe DECIMAL-POINT IS COMMA für beide Typdeklarationen gilt oder für keine von beiden. Dasselbe gilt für das Zeichen *Komma* in der zugehörigen PICTURE Zeichenkette.

Schwach typisierte Datenbeschreibungen

Schwach typisierte Datenbeschreibungen übernehmen die Eigenschaften ihrer zugehörigen Typbeschreibungen. Diese Eigenschaften können nicht durch Festlegungen übergeordneter Datengruppen geändert werden.

Schwach typisierte Datenbeschreibungen können genau wie Datenbeschreibungen ohne zugehörige Typbeschreibung genutzt werden.

Stark typisierte Datenbeschreibungen

Stark typisierte Datenbeschreibungen übernehmen ebenfalls die Eigenschaften ihrer zugehörigen Typbeschreibungen. Zusätzlich gibt es Einschränkungen bei ihrer Nutzung, um die Integrität der Daten zu schützen.

Es können nur Datengruppen, jedoch keine Datenelemente, stark typisiert sein.

2.4.7 Null-längige Datenfelder

Ein null-längiges Datenfeld ist ein Datenfeld, dessen minimale erlaubte Länge 0 ist, und dessen Länge zur Ablaufzeit 0 ist. Ein null-längiges Datenfeld ist eines der folgenden:

- Eine Datengruppe, die nur Datenfelder enthält, die mit OCCURS DEPENDING ON beschrieben sind und die Anzahl der Wiederholungen ist 0.
- Eine Datengruppe, der nur null-längige Datenfelder untergeordnet sind.
- Ein Datenfeld, das mit ANY LENGTH definiert ist und einem Argument bzw. Rückgabeelement entspricht, das ein null-längiges Datenfeld ist.
- Ein Datensatz einer Datei mit variablem Satzformat, in dem die Anzahl der Zeichenpositionen 0 ist.
- Eine interne Standardfunktion, die ein null-längiges Ergebnis liefert.
- Die Sonderregister XML-TEXT, XML-NAMESPACE, XML-NAMESPACE-PREFIX bzw. XML-NTEXT, XML-NNAMESPACE, XML-NNAMESPACE-PREFIX, wenn sie nicht mit einem Wert versorgt sind (siehe Abschnitt „Sonderregister für XML PARSE-Anweisung").

2.4.8 Konzept der maschinenunabhängigen Datenbeschreibung

Daten sollten so maschinenunabhängig wie möglich beschrieben sein. Um dies zu erreichen, werden die Charakteristika und Eigenschaften der Daten in einem Format beschrieben, das weitgehend unabhängig ist von der konkreten Darstellung im Computer oder auf dem externen Datenträger.

Für Literale gilt:

- Der Inhalt alphanumerischer nicht-hexadezimaler Literale wird für die Ablaufzeit unverändert übernommen.
- Der Inhalt nationaler nicht-hexadezimaler Literale wird für die Ablaufzeit immer zu UTF-16-Darstellung konvertiert.
- Hexadezimale Literale geben immer das endgültige, zur Ablaufzeit zu verwendende Bitmuster an.

1. Konzept des logischen Satzes und der Datei

Die logischen Eigenschaften eines Satzes oder einer Datei unterscheiden sich von der Art und Weise, wie Daten in der Datenverarbeitungsanlage physisch abgespeichert werden.

- **Physische Eigenschaften einer Datei**

  Die physischen Eigenschaften einer Datei ergeben sich aus der Art und Weise, wie die Daten auf dem Eingabe- oder Ausgabemedium gespeichert sind, und umfassen Angaben wie

  - die Gruppierung der logischen Sätze unter Berücksichtigung der physischen Grenzen des Speichermediums,
  - die Art, wie eine Datei identifiziert werden kann.

- **Logische Eigenschaften einer Datei**

  Die logischen Eigenschaften einer Datei ergeben sich aus den Strukturen, die der Benutzer durch Datendefinitionen festlegt. Die Ein-/Ausgabe-Anweisungen in einem COBOL-Programm beziehen sich auf logische Sätze.

  Es ist äußerst wichtig, zwischen einem physischen Satz und einem logischen Satz zu unterscheiden.

  - Ein **physischer Datensatz** (oder **Block**) ist eine Informationseinheit, deren Größe und Satzmodus zum Speichern von Daten auf einem Ein- oder Ausgabedatenträger für eine bestimmte Datenverarbeitungsanlage vorteilhaft ist. Die Größe eines physischen Datensatzes ist maschinenorientiert und gestattet keinen direkten Vergleich zu der Größe der logischen Dateiinformation.
  - Ein **logischer Datensatz** (oder nur **Datensatz**) ist eine Gruppe von zusammengehörigen Daten, die eindeutig bezeichnet und als Einheit behandelt werden kann sowie aus einer Datei gelesen oder in eine Datei geschrieben werden kann. Mehrere Datensätze können in einem einzelnen Block enthalten sein.

  Der Begriff Datensatz ist nicht beschränkt auf Daten, die auf einem externen Datenträger gespeichert sind, sondern ist übertragbar auf die Definition des Arbeitsspeichers für die Daten, die intern während des Programmablaufs erzeugt werden.

  In dieser Beschreibung bedeuten Bezüge auf Datensätze immer Bezüge auf logische Datensätze.

2. Stufenkonzept

Das Stufenkonzept erlaubt die Strukturierung eines logischen Datensatzes. Daten, die von einem COBOL-Programm verarbeitet werden müssen, werden als Datenelemente, Datengruppen, Datensätze und Dateien beschrieben.

- **Datenelemente**

  Ein Datenelement ist die kleinste benannte Dateneinheit; d.h. ihm sind keine weiteren Datenelemente untergeordnet. Ein Datenelement wird meistens mit der PICTURE-Klausel beschrieben.
  Zu Ausnahmen siehe Abschnitt „Formate der Datenerklärung".

  Die Länge eines Datenelements darf 131 071 Bytes nicht überschreiten.

- **Datengruppen**
  Mehrere Datenelemente werden zu einer Datengruppe zusammengefasst. So kann eine Anzahl von Datenelementen unter dem Namen der Datengruppe auf einmal angesprochen werden. Eine Datengruppe besteht somit aus einem oder mehreren Datenelementen. Sie darf aber auch weitere Datengruppen enthalten. Infolgedessen kann ein Datenelement zu mehr als einer Datengruppe gehören (siehe Bild 1 und Bild 2). Der Name einer Datengruppe darf nicht mit der PICTURE-Klausel beschrieben sein.
- **Datensätze**
  Ein Datensatz ist ein Datenfeld, das keinem anderen untergeordnet ist. Ein Datensatz besteht aus einer Datengruppe oder ist selbst ein Datenelement. Die Beschreibung eines Datensatzes steht im Programmtext-Bereich.
- **Stufennummern**

  Daten werden in verschiedene Stufen gegliedert. Diese Stufen werden durch Stufennummern bezeichnet. Zugelassen sind die Stufennumern 01 bis 49. Daneben gibt es die speziellen Stufennummern 66, 77 und 88. In einer Übersetzungseinheit muss für jede Stufennummer eine gesonderte Eintragung erfolgen.

  Da ein Datensatz die größte Ordnungseinheit darstellt, beginnen die Stufennummern für Datensätze mit 01. Den hierarchisch untergeordneten Feldern werden zahlenmäßig höhere Stufennummern (02 bis 49) zugewiesen. Die Stufennummer eines untergeordneten Datenfeldes muss um eine oder mehrere Einheiten größer als die des übergeordneten Datenfeldes sein. Nach der Beschreibung eines Datenelementes ist nur eine Stufennummer zulässig, die in derselben Datensatzbeschreibung schon einmal aufgetreten ist.

  Beispiel 2-8

  Richtig:

```
01 DATENSATZ.
   05 DATENGRUPPE-1.
      10 DATENELEMENT-11
...
      10 DATENELEMENT-12
...
      10 DATENELEMENT-13
...
   05 DATENGRUPPE-2.
      10 DATENELEMENT-21
...
      10 DATENELEMENT-22
...
```

  Falsch:

```
01 DATENSATZ.
   05 DATENGRUPPE-1.
      10 DATENELEMENT-11
...
      10 DATENELEMENT-12
...
      10 DATENELEMENT-13
...
   03 DATENGRUPPE-2.
      10 DATENELEMENT-21
...
      10 DATENELEMENT-22
...
```

  Drei Arten von Daten, für die kein Stufenkonzept vorhanden ist, werden den Stufennummern 66, 77 und 88 zugeordnet:

  - Die Stufennummer 66 für Namen der Datenfelder, die mit der RENAMES-Klausel beschrieben sind (siehe Abschnitt „RENAMES-Klausel").
  - Die Stufennummer 77 für strukturunabhängige Datenfelder der WORKING-STORAGE SECTION bzw. LINKAGE SECTION (siehe Abschnitt „Stufennummer").
  - Die Stufennummer 88 für die Erklärung von Bedingungsnamen (siehe Abschnitt „VALUE-Klausel")

  Weitere Regeln sind unter „Stufennummer“ (Abschnitt „Stufennummer") beschrieben.

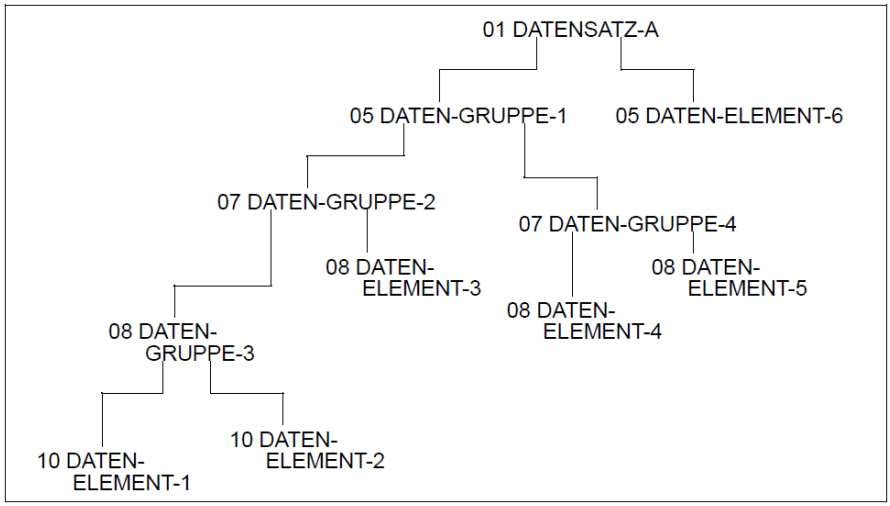


Bild 1: Beziehungen von Datengruppen und Datenelementen in einem Datensatz

```
01 DATENSATZ-A.
   05 DATEN-GRUPPE-1.
      07 DATEN-GRUPPE-2.
         08 DATEN-GRUPPE-3.
            10 DATEN-ELEMENT-1...
            10 DATEN-ELEMENT-2...
         08 DATEN-ELEMENT-3...
      07 DATEN-GRUPPE-4.
         08 DATEN-ELEMENT-4...
         08 DATEN-ELEMENT-5...
   05 DATEN-ELEMENT-6...
```

Bild 2: Datengruppen und Datenelemente in einem Datensatz

Bild 1 zeigt die Struktur eines Musterdatensatzes und Bild 2 veranschaulicht, wie die Stufennummern zu verwenden sind, um diese Struktur in der Beschreibung des Datensatzes wiederzugeben. In diesem Beispiel sind DATEN-GRUPPE-3 und DATEN-ELEMENT-3 Teil von DATEN-GRUPPE-2 sowie DATEN-GRUPPE-2 und DATEN-GRUPPE-4 Teil von DATEN-GRUPPE-1.

3. Klassen und Kategorien von Daten und Literalen

Jedem Datenfeld und Literal ist eine Klasse und Kategorie zugeordnet.

Die Klasse und Kategorie einer stark typisierten Datengruppe ist der Typname, der in der TYPE-Klausel der Datengruppe angegeben ist.

Die Klasse und Kategorie einer nicht stark typisierten Datengruppe sind

- alphanumerisch für eine alphanumerische Gruppe
- national für eine nationale Gruppe

Eine alphanumerische Gruppe wird behandelt, als ob sie USAGE DISPLAY hat.

Die Klasse und Kategorie einer Funktion wird durch den Typ der Funktion festgelegt (siehe „Funktionstypen" im Abschnitt "Allgemeines").
Die Kategorie eines Datenelements hängt von seiner Beschreibung ab.
Die Klassen und Kategorien von Literalen sind im Abschnitt „Literale" definiert.

Die nachfolgende Tabelle 7 veranschaulicht die Beziehungen von Klassen und Kategorien von Datenelementen.

Klasse	Kategorie

alphabetisch	alphabetisch
alphanumerisch	numerisch druckaufbereitetalphanumerisch druckaufbereitetalphanumerisch
index	index
national	national
numerisch	numerisch
objekt	objektreferenz
zeiger	datenzeigerprogrammzeiger

Tabelle 7: Klassen und Kategorien von Datenelementen

Die folgenden Abschnitte beschreiben die Datenfelder in den verschiedenen Kategorien.

- **Alphabetische Datenfelder**

  Ein alphabetisches Datenfeld ist ein Datenfeld, dessen Inhalt eine beliebige Kombination der 52 Groß- und Kleinbuchstaben des Alphabets und des Leerzeichens sein kann. Jedes alphabetische Zeichen ist in einem eigenen Byte des Arbeitsspeichers gespeichert.

  Die Maskenzeichenfolge für alphabetische Datenfelder enthält nur das Symbol A.Das Datenformat von alphabetischen Datenfeldern ist immer DISPLAY.

  Ein alphabetisches Datenfeld darf überall dort angegeben werden, wo ein alphanumerisches Datenelement als Operand zulässig ist. Sofern nicht anders angegeben, wird es behandelt, als ob es ein alphanumerisches Datenfeld wäre.

- **Alphanumerisch druckaufbereitete Datenfelder**

  Ein alphanumerisch druckaufbereitetes Datenfeld beschreibt die Form der Druckaufbereitung eines alphanumerischen Wertes. Falls ein alphanumerisch druckaufbereitetes Datenfeld ein Empfangsdatenfeld einer MOVE-Anweisung ist, werden die in das Datenfeld übertragenen Daten entsprechend der angegebenen Maskenzeichenfolge druckaufbereitet.

  Seine Maskenzeichenfolge ist auf bestimmte Kombinationen der Zeichen A, X, 9,0 (Null), B und / (Schrägstrich) beschränkt (siehe Abschnitt „PICTURE-Klausel").

  Die Inhalte eines alphanumerisch druckaufbereiteten Datenfeldes sind aus dem EBCDIC-Zeichensatz ausgewählte, zulässige Zeichen.Das Datenformat eines alphanumerisch druckaufbereiteten Datenfeldes ist immer DISPLAY.

- **Alphanumerische Datenfelder**

  Ein alphanumerisches Datenfeld ist ein Datenfeld, dessen Inhalt ein beliebiges Zeichen aus dem EBCDIC-Zeichensatz sein kann.

  Die Maskenzeichenfolge eines alphanumerischen Datenfeldes ist auf eine Mischung der Zeichen A, X und 9 beschränkt. Das Datenfeld wird behandelt, als ob in seiner Maskenzeichenfolge nur die Zeichen X wären.

  Eine Maskenzeichenfolge, die nur die Zeichen A oder nur die Zeichen 9 enthält, erklärt kein alphanumerisches Datenfeld. Das Datenformat eines alphanumerischen Datenfeldes ist immer DISPLAY.

- **Datenzeiger**

  Ein Datenzeiger (auch kurz nur Zeiger) ist ein Datenelement, in dem eine Datenadresse gespeichert werden kann.

- **Index Datenfelder**

  Jedes der folgenden ist ein Index Datenfeld:

  - ein Datenelement, das explizit oder implizit mit USAGE INDEX beschrieben ist
  - eine index Funktion

- **Nationale Datenfelder**

  Jedes der folgenden ist ein nationales Datenfeld:

  - ein Datenelement, das durch seine PICTURE Maskenzeichenfolge als national beschrieben ist
  - ein Datenelement ohne PICTURE-Klausel, das mit einer VALUE-Klausel des Formats 1 und einem nationalen Literal beschrieben ist
  - ein Datenelement, das explizit oder implizit mit USAGE NATIONAL beschrieben ist
  - eine Datengruppe, die explizit oder implizit mit einer GROUP-USAGE NATIONAL Klausel beschrieben ist
  - eine nationale Funktion

- **Numerisch druckaufbereitete Datenfelder**

  Ein numerisch druckaufbereitetes Datenfeld beschreibt die Form der Druckaufbereitung eines numerischen Wertes. Falls ein numerisch druckaufbereitetes Datenfeld ein Empfangsdatenfeld einer MOVE-Anweisung ist, werden die in das Datenfeld übertragenen Daten entsprechend der angegebenen Maskenzeichenfolge druckaufbereitet.

  Die Maskenzeichenfolge ist auf bestimmte Kombinationen der Zeichen:B, / (Schrägstrich), P, V, Z, 0 (Null), 9, , (Komma), . (Dezimalpunkt), *, +, -, CR, DB und $ (Währungszeichen) beschränkt. Die zulässigen Kombinationen werden von Druckaufbereitungsregeln und der Präzedenzreihenfolge der Zeichen bestimmt (siehe Abschnitt „PICTURE-Klausel"). Die maximale Zahl von Ziffern in einer Maskenzeichenfolge für ein numerisch druckaufbereitetes Datenfeld ist 31.

  Daten werden mit einem Zeichen pro Byte gespeichert. Der Inhalt einer Zeichenstelle, die eine Ziffer darstellt, muss eine der Ziffern 0 bis 9 sein.

  Das Datenformat eines numerisch druckaufbereiteten Datenfeldes ist immer DISPLAY.

- **Numerische Datenfelder**

  Es gibt zwei Arten der Darstellung von numerischen Datenfeldern, Festpunktdatenfelder und Gleitpunktdatenfelder. Für die interne Darstellung von numerischen Datenfeldern siehe Tabelle 17 im Abschnitt "COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe".

Festpunktdatenfelder

Ein Festpunktdatenfeld ist ein numerisches Datenfeld, in dem der Rechendezimalpunkt in jedem Wert immer als vorhanden angenommen oder an einer festen Stelle relativ zum Beginn oder Ende des Speicherbereichs festgehalten wird, der für das Datenfeld belegt ist. Der Inhalt eines Festpunktdatenfeldes muss sich, falls die SIGN-Klausel nicht angegeben ist, aus den Ziffern 0 bis 9 zusammensetzen. Ist die SIGN-Klausel angegeben, darf der Inhalt zusätzlich zu den obigen Ziffern noch +, - oder andere Darstellungen des Vorzeichens enthalten. Falls die Maskenzeichenfolge für ein Festpunktdatenfeld ein S enthält, wird der Inhalt des Datenfeldes als positiver oder negativer Wert behandelt, abhängig vom Rechenvorzeichen. Falls die Maskenzeichenfolge kein S enthält, wird der Inhalt des Datenfeldes als absoluter Wert behandelt.

Maskenzeichenfolgen für Festpunktdatenfelder dürfen nur die symbolischen Zeichen 9, P, S und V enthalten.

COBOL kennt drei Arten von Festpunktzahlen:

extern dezimal	(USAGE IS DISPLAY)
binär	(USAGE IS COMPUTATIONAL oder COMPUTATIONAL-5 oder USAGE IS BINARY)
intern dezimal	(USAGE IS COMPUTATIONAL-3 oder USAGE IS PACKED-DECIMAL)

Die Beschreibung der Unterschiede ist unter Abschnitt „USAGE-Klausel" aufgeführt.

Gleitpunktdatenfelder

Ein Gleitpunktdatenfeld ist ein numerisches Datenfeld, dessen Dezimalpunkt verschiebbar ist, d.h. der Dezimalpunkt kann an verschiedenen Stellen stehen und wird durch die Angabe eines Exponenten der Basis

(10) festgelegt. Die Basis, damit exponentiert, dient als Koeffizient einer Festpunktzahl (Mantisse); damit ist die Gleitpunktzahl dargestellt.

Gleitpunktdatenfelder werden nur für Daten benutzt, deren potenzieller Wertebereich zur Festpunktdarstellung zu groß ist.

Es gibt zwei Arten von Gleitpunktdatenfeldern: externe Gleitpunktdatenfelder und interne Gleitpunktdatenfelder.

Externe Gleitpunktdatenfelder

Die Maskenzeichenfolge für ein externes Gleitpunktdatenfeld kann die Zeichen. (Dezimalpunkt), V, E, + und - enthalten. Jedes Zeichen, außer V, belegt ein Byte des Speichers und ist in jeder Druckausgabe enthalten (siehe „PICTURE-Klausel“, und „USAGE-Klausel“, für weitere Details).

Das Datenformat eines externen Gleitpunktdatenfeldes ist immer DISPLAY.

Interne Gleitpunktdatenfelde

Es gibt zwei Arten von internen Gleitpunktdatenfeldern:

- einfach genaue Datenfelder (USAGE IS COMPUTATIONAL-1) mit einer Länge von vier Bytes und
- doppelt genaue Datenfelder (USAGE IS COMPUTATIONAL-2) mit einer Länge von acht Bytes.

Beide umfassen denselben Wertebereich. Das einfach genaue Datenfeld erlaubt sieben Dezimalziffern Genauigkeit. Das doppelt genaue Datenfeld erlaubt sechzehn Dezimalziffern Genauigkeit.

Eine PICTURE-Klausel ist für interne Gleitpunktzahlen nicht anzugeben; die Länge eines solchen Datenfeldes ist durch seine USAGE-Klausel bestimmt.

- **Objekt und Objektreferenz**

  Eine Objektreferenz ist ein implizit oder explizit definiertes Datenelement. Der Inhalt der Objektreferenz verweist eindeutig auf ein Objekt und seine zugehörigen Informationen.

- **Programmzeiger**

  Ein Programmzeiger ist ein Datenelement, in dem die Adresse eines Programms gespeichert werden kann.

4. Algebraische Vorzeichen

Es gibt zwei Kategorien algebraischer Vorzeichen:

- Rechenvorzeichen, die mit vorzeichenbehafteten numerischen Datenfeldern und vorzeichenbehafteten numerischen Literalen zusammenhängen, um ihre algebraischen Merkmale aufzuzeigen,
- Druckaufbereitungsvorzeichen, die z.B. in druckaufbereiteten Listen zum Bezeichnen des Vorzeichens des Datenfeldes auftreten.
  Druckaufbereitungsvorzeichen werden in ein Datenfeld durch Verwendung des Vorzeichen-Steuerzeichens der jeweiligen Maskenzeichenfolge (siehe Abschnitt „PICTURE-Klausel") eingefügt.

5. Ausrichtung von Daten

Die Ausrichtung von Daten innerhalb von Datenelementen hängt von der Kategorie des Empfangsfeldes ab.

- **Numerische Datenfelder**

  Falls das Empfangsfeld als numerisch beschrieben ist, wird das Sendedatum am Dezimalpunkt ausgerichtet in die Ziffernstellen des Empfangsfeldes übertragen. Ist das Sendedatum kürzer als das Empfangsfeld, werden die nicht verwendeten Ziffernstellen mit Nullen aufgefüllt. Ist das Sendedatum länger als das Empfangsfeld, wird es von links bzw. von rechts abgeschnitten.
  Wenn ein Rechendezimalpunkt nicht ausdrücklich angegeben ist, wird das Empfangsfeld so behandelt, als hätte es einen Rechendezimalpunkt unmittelbar nach der am weitesten rechts stehenden Ziffer. Ausrichtung und Übertragung erfolgen wie oben beschrieben.

- **Numerisch druckaufbereitete Datenfelder**

Falls das Empfangsfeld ein numerisch druckaufbereitetes Datenfeld ist, erfolgt die Ausrichtung und Übertragung des Sendedatums wie bei numerischen Empfangsfeldern, wobei durch spezielle Druckaufbereitungsangaben führende Nullen durch andere Zeichen ersetzt werden können.

- **Alphanumerische, alphanumerisch druckaufbereitete, alphabetische und nationale Datenfelder**

  Falls das Empfangsfeld alphanumerisch (anders als ein numerisch druckaufbereitetes Datenfeld), alphanumerisch druckaufbereitet, alphabetisch oder national ist, wird das Sendedatum von links nach rechts in die Zeichenstellen des Empfangsfeldes übertragen. Ist das Sendedatum kürzer als das Empfangsfeld, werden die nicht verwendeten Zeichenstellen mit Leerzeichen entsprechend der Klasse des Empfangsfeldes aufgefüllt. Ist das Sendedatum länger als das Empfangsfeld, werden die überschüssigen Zeichen des Sendedatums abgeschnitten.Falls die JUSTIFIED-Klausel für das Empfangsfeld angegeben ist, siehe Beschreibung der JUSTIFIED-Klausel (Abschnitt „JUSTIFIED-Klausel").

- **Datenfeldausrichtung für schnelleren Ablauf des Programms**

  Bestimmte Daten (in arithmetischen Operationen oder in Subskribierungen) können schneller verarbeitet werden, wenn die Daten an „natürlichen“ Grenzen (Halbwort, Wort, Doppelwort) ausgerichtet sind.
  Es sind zusätzliche Maschinenbefehle für den Zugriff und das Abspeichern von Daten notwendig, falls Teile von zwei oder mehr Datenfeldern zwischen zwei benachbarten natürlichen Grenzen vorkommen oder falls bestimmte natürliche Grenzen ein einzelnes Datenfeld aufteilen.
  Datenfelder, die an diesen „natürlichen“ Grenzen so ausgerichtet sind, dass sie zusätzliche Maschinenbefehle vermeiden, sind als **ausgerichtet** („synchronized“) definiert. Immer ausgerichtet werden Datenfelder der Klassen index, objekt und zeiger (siehe Tabelle 16 „Ausrichtung von Datenfeldern" im Abschnitt "SYNCHRONIZED-Klausel").

  Für diese Form des Ausrichtens hat der Benutzer zwei Möglichkeiten:

  - Angabe der SYNCHRONIZED-Klausel (siehe Abschnitt „SYNCHRONIZED-Klausel"),
  - geeignetes Organisieren der Daten unter Berücksichtigung der entsprechenden natürlichen Grenzen. Näheres hierzu siehe nächster Abschnitt.

2.4.9 Herstellerabhängige Darstellung und Ausrichtung von Daten

Ausrichtung durch Einfügen von Füllfeld-Bytes

Es gibt zwei Arten von Füllfeld-Bytes:

- Füllfeld-Bytes *innerhalb* von Datensätzen sind nicht verwendete Zeichenstellen, die jedem ausgerichteten Datenfeld im Datensatz vorangehen.
- Füllfeld-Bytes *zwischen* Datensätzen sind nicht verwendete Zeichenstellen, die zwischen geblockten logischen Datensätzen eingefügt sind.

- **Füllfeld-Bytes innerhalb von Datensätzen**

  Für eine Ausgabedatei oder in der WORKING-STORAGE SECTION und LOCAL-STORAGE SECTION fügt der Compiler Füllfeld-Bytes innerhalb von Datensätzen ein, um sicherzustellen, dass sich alle ausgerichteten Datenfelder an den geeigneten Grenzen befinden. Für eine Eingabedatei oder in der LINKAGE SECTION erwartet der Compiler, dass eventuell notwendige Füllfeld-Bytes vorhanden sind, um die richtige Ausrichtung eines mit SYNCHRONIZED erklärten Datenfeldes zu garantieren.

  Da es für die Benutzer sehr wichtig ist, die Länge eines Datensatzes in einer Datei zu kennen, ist der Algorithmus beschrieben, den der Compiler verwendet, um zu bestimmen, ob Füllfeld-Bytes notwendig sind und, falls sie notwendig sind, die entsprechende Anzahl von Füllfeld-Bytes hinzuzufügen:

  Die Zahl von belegten Bytes aller Datenelemente, die einem Datenfeld in einem Datensatz vorausgehen, werden zusammengezählt, einschließlich eventuell vorher dazugezählter Füllfeld-Bytes.

  Diese Summe ist durch m zu dividieren, wobei gilt:

m = 2	für COMPUTATIONAL- bzw. COMPUTATIONAL-5 bzw. BINARY-Daten-felder von 4 Ziffern Länge oder weniger
m = 4	für COMPUTATIONAL- bzw. COMPUTATIONAL-5 bzw. BINARY-Datenfelder von 5 Ziffern Länge oder mehr
m = 4	für COMPUTATIONAL-1-Datenfelder
m = 8	für COMPUTATIONAL-2-Datenfelder
m = 4	für Indexdatenfelder
m = 8	für Objektreferenzen
m = 4	für Zeiger
m = 8	für typisierte Datengruppen

  Falls der Divisionsrest r dieser Division gleich Null ist, werden keine Füllfeld-Bytes benötigt. Falls der Divisionsrest nicht gleich Null ist, so ist die Zahl von Füllfeld-Bytes, die hinzugefügt werden müssen, gleich m - r.

  Diese Füllfeld-Bytes werden jedem Datensatz unmittelbar nach dem Datenelement eingefügt, das dem BINARY-, COMPUTATIONAL-, COMPUTATIONAL-1-, COMPUTATIONAL-2-, COMPUTATIONAL-5-, dem INDEX-Datenfeld, der Objektreferenz, dem Zeiger oder dem typisierten Datenfeld vorausgeht. Sie sind so erklärt, als wären sie ein Datenfeld mit einer Stufennummer gleich der des Datenfeldes, das dem ausgerichteten Datenfeld unmittelbar vorausgeht, und sie sind mit eingerechnet in die Größe der sie enthaltenden Gruppe.

  > **i** Legen sie in Unterstrukturen solche Ausrichtung verlangenden Felder an den Anfang. Dadurch ist sichergestellt, dass immer gleich viele Füllfeld-Bytes enthalten sind.

Beispiel 2-9

Füllfeld-Bytes innerhalb von Datensätzen

```
01  A.
    02   B  PICTURE X(5).
    02   C.
         03 D  PICTURE XX.
        [03 Füllfeld-Byte PICTURE X. Eingefügt vom Compiler.]
         03 E  PICTURE S9(6) COMP SYNCHRONIZED.
```

Füllfeld-Bytes werden auch dann vom Compiler hinzugefügt, wenn eine Datengruppe mit OCCURS-Klausel beschrieben ist und ein ausgerichtetes Datenfeld mit USAGE definiert als BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-5 oder INDEX enthält. Um zu entscheiden, ob Füllfeld-Bytes dazuzufügen sind, werden folgende Schritte durchgeführt:

- Der Compiler berechnet die Größe der Gruppe, einschließlich aller notwendigen Füllfeld-Bytes innerhalb des Datensatzes.
- Diese Summe wird durch das größte, von irgendeinem Datenfeld innerhalb der Gruppe benötigtes m dividiert.
- Falls der Divisionsrest r dieser Division gleich Null ist, werden keine Füllfeld-Bytes benötigt. Falls r ungleich Null ist, müssen m - r Füllfeld-Bytes hinzugezählt werden.

Die Füllfeld-Bytes werden am Ende jeder Wiederholung der Datengruppe eingefügt, die die OCCURS-Klausel enthält, um sicherzustellen, dass alle Wiederholungen von Tabellendatenfeldern an derselben Art von Grenze beginnen. Im Beispiel 2-10 beginnen alle Wiederholungen von D ein Byte hinter einer Doppelwortgrenze.

Beispiel 2-10

Wiederholungen von Füllfeld-Bytes in Tabellen

```
01 A.
   02 B  PICTURE X.
   02 C  OCCURS 10 TIMES.
      03 D PICTURE X.
     [03 Füllfeld-Bytes PICTURE XX. Eingefügt vom Compiler.]
      03 E PICTURE S9(4)V99 COMP SYNC.
      03 F PICTURE S9(4) COMP SYNC.
      03 G PICTURE X(5).
     [03 Füllfeld-Bytes PICTURE XX. Eingefügt vom Compiler.]
```

Falls ausgerichtete Datenfelder, definiert als BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-5 oder INDEX, einem Eintrag mit einer OCCURS DEPENDING-Klausel folgen, so werden Füllfeld-Bytes auf der Basis des Feldes dazugezählt, das mit der Maximalzahl wiederholt wird. Ist die Länge dieses Feldes nicht durch das von den Daten benötigte m teilbar, dann ergeben nur bestimmte Werte des bei der DEPENDING-Angabe verwendeten Datennamens eine richtige Ausrichtung der Felder. Der Programmierer sollte sich dieser Situation bewusst sein und sie zu vermeiden versuchen. Diese Werte sind solche, bei denen die Länge des Datenfeldes, multipliziert mit der Anzahl der Wiederholungen plus der Zahl der Füllfeld-Bytes, die auf der Basis der maximalen Wiederholungszahl berechnet wurden, durch m ohne Rest teilbar ist.

Beispiel 2-11

Wiederholungen von Füllfeld-Bytes in Tabellen mit DEPENDING-Angabe

```
01  A.
    02  B   PICTURE 99.
    02  C   PICTURE X OCCURS 50 TO 99 TIMES
             DEPENDING ON B.
   [02  Füllfeld-Byte PICTURE X. Eingefügt vom Compiler.]
    02  D   PICTURE S99 COMP SYNC.
```

Sind in diesem Beispiel Bezugnahmen zu D notwendig, so ist B auf ungerade Werte beschränkt.

```
01  A.
    02  B   PICTURE 999.
    02  C   PICTURE XX OCCURS 20 TO 99 TIMES
             DEPENDING ON B.
   [02  Füllfeld-Bytes PICTURE X. Eingefügt vom Compiler.]
    02  D   PICTURE S99 COMP SYNC.
```

In diesem Beispiel ergeben alle Werte von B richtige Bezugnahmen auf D.

- **Ausrichtung von Datensätzen**

  Alle Satzbeschreibungen (01-Stufen) in allen Kapiteln des Datenteils beginnen an der Doppelwortgrenze.

- **Füllfeld-Bytes zwischen Datensätzen**

  Wenn Datensätze, die ausgerichtete Datenfelder enthalten, zu blocken sind, so muss der Programmierer sicherstellen, dass alle Datensätze nach dem ersten Datensatz im Ein-/Ausgabe-Speicherbereich die richtige Grenzenausrichtung haben. Dies ist jedoch nur nötig, wenn die Daten blockweise zu verarbeiten sind (locate mode). COBOL2000 verwendet diesen Modus nicht.

2.5 Eindeutigkeit von Bezugnahmen

In diesem Kapitel werden folgende Themen behandelt:

- Kennzeichnung
- Subskribierung
- Indizierung
- Funktionsbezeichner
- Teilfeldselektion
- Bezeichner
- Objektsicht (object-view)
- Vordefinierte Objektreferenzen
  - NULL
  - SELF und SUPER
- Vordefinierte Adresse NULL
- Datenadressbezeichner
- Programmadressbezeichner
- BYTE-LENGTH OF
- LENGTH OF
- Bedingungsname

2.5.1 Kennzeichnung

Funktion

Jeder vom Benutzer angegebene Name, der in einer COBOL-Übersetzungseinheit explizit referenziert wird, muss eindeutig sein. Ein Name ist eindeutig, wenn kein anderer Name aus der gleichen Folge von Zeichen und Bindestrichen besteht oder der Name in einer Hierarchie von Namen vorkommt, so dass eindeutig auf ihn Bezug genommen werden kann. Dies geschieht, indem ein oder mehrere Namen einer höheren Stufe der Hierarchie angegeben werden. Die höheren Stufen heißen Kennzeichner, und der Vorgang, der die Eindeutigkeit von Namen bewirkt, heißt Kennzeichnung. Ein Name muss ausreichend gekennzeichnet sein, um eindeutig zu sein; jedoch ist es nicht unbedingt nötig, alle Stufen der Hierarchie anzugeben. Innerhalb der DATA DIVISION müssen alle Datennamen, die zur Kennzeichnung benutzt werden, mit einer Stufennummer oder einer Stufenbezeichnung versehen werden. Deshalb dürfen zwei gleiche Datennamen nicht untergeordnete Elemente einer Datengruppe sein, es sei denn, sie können eindeutig gekennzeichnet werden. In der PROCEDURE DIVISION dürfen zwei gleiche Paragrafennamen nur dann im gleichen Kapitel auftreten, wenn auf sie nicht Bezug genommen wird. Wird ein Paragrafenname referenziert, muss er eindeutig sein, d.h. er muss, wenn er in mehreren Kapiteln vorkommt, gekennzeichnet werden.

Im Zusammenhang mit Typen ist die Eindeutigkeit von Namen gegeben, wenn der Name nur innerhalb einer Typdefinition auftritt und diese Typdefinition nicht mehr als einmal zur Definition von Daten genutzt wird. In allen anderen Fällen ist die Kennzeichnung notwendig.

In der Hierarchie der Kennzeichnung sind die zu einer Stufenbezeichnung gehörenden Namen am wichtigsten, danach die Namen, die zur Stufennummer 01 gehören, danach Namen mit Stufennummern 02 bis 49. Ein Kapitelname ist der einzige Kennzeichner, der für Paragrafennamen zur Verfügung steht. Der oberste Name in der Hierarchie muss eindeutig sein und darf nicht gekennzeichnet werden. Subskribierte oder indizierte Datennamen und Bedingungsvariable sowie Prozedurnamen und Datennamen können durch Kennzeichnung eindeutig gemacht werden. Der Name einer Bedingungsvariablen kann als Kennzeichner für jeden seiner Bedingungsnamen verwendet werden.

Format 1

```
{datenname-1 | bedingungsname} {  {IN | OF}... [{IN | OF} dateiname]
                                  | {IN | OF}  dateiname
                                }
```

Format 2

```
paragrafenname {IN | OF} kapitelname
```

Format 3

```
textname {IN | OF} bibliotheksname
```

Format 4

```
LINAGE-COUNTER {IN | OF} dateiname
```

Format 5

```
{PAGE-COUNTER | LINE-COUNTER} {IN | OF} listenname
```

Format 6

```
datenname-1  {  {IN | OF} datenname-2 [{IN | OF} listenname]
             | {IN | OF} listenname
            }
```

Syntaxregeln

1. Jeder Kennzeichner muss auf höherer Stufe und innerhalb der gleichen Hierarchie auftreten wie der Name, den er kennzeichnet.
2. Der gleiche Name darf nicht auf zwei Stufen in einer Hierarchie erscheinen.
3. Ein Datenname darf nicht subskribiert oder indiziert sein, wenn er als Kennzeichner benutzt wird.

Allgemeine Regeln

1. Falls ein Datenname oder ein Bedingungsname mehr als einem Datenfeld innerhalb der Übersetzungseinheit zugeordnet ist, muss der Datenname oder der Bedingungsname jedesmal gekennzeichnet werden, wenn in der PROCEDURE DIVISION, ENVIRONMENT DIVISION und DATA DIVISION darauf Bezug genommen wird (außer in der REDEFINES-Klausel, wo Kennzeichnung unnötig ist und nicht verwendet werden darf).
2. Ein Paragrafenname darf innerhalb eines Kapitels nur dann mehrfach auftreten, wenn nicht auf ihn Bezug genommen wird. Wird auf ihn Bezug genommen, darf er innerhalb eines Kapitels nur einmal vorkommen bzw. muss, wenn er in mehreren Kapiteln auftritt, gekennzeichnet werden. Wenn ein Paragrafenname durch einen Kapitelnamen gekennzeichnet wird, darf das Wort SECTION nicht verwendet werden.
   Ein Paragrafenname braucht nicht gekennzeichnet zu werden, wenn auf ihn innerhalb des gleichen Kapitels Bezug genommen wird.
3. Ein Name kann gekennzeichnet werden, auch wenn keine Kennzeichnung nötig ist; falls es mehrere Kombinationen von Kennzeichnern gibt, die Eindeutigkeit garantieren, so darf jede dieser Kombinationen verwendet werden. Die Gesamtmenge der Kennzeichner für einen Datennamen darf nicht die gleiche sein wie eine Teilmenge von Kennzeichnern für einen anderen Datennamen.
4. Ist mehr als eine COBOL-Bibliothek während der Übersetzungszeit für den Compiler verfügbar, muss der Textname, jedesmal wenn er angesprochen wird, durch bibliotheksname gekennzeichnet sein.
5. Wird datenname in einem inneren oder äußeren Programm eines Schachtelprogramms gekennzeichnet, darf der gleiche Datenname nicht für ein Datum (Datensatz oder Datenfeld) verwendet werden, das in einem der Programme des Schachtelprogramms als extern oder global deklariert ist.

2.5.2 Subskribierung

Funktion

Subskripte werden verwendet, wenn innerhalb einer Tabelle auf ein einzelnes Element zugegriffen werden soll (siehe Abschnitt „OCCURS-Klausel").

Format 1

beschreibt Subskribierung ohne Kennzeichnung.

```
{datenname-1 | bedingungsname} ({subskript-1}...)
```

Format 2

beschreibt Subskribierung mit Kennzeichnung.

```
{datenname-1 | bedingungsname} {IN | OF} datenname -2 [{IN | OF} datenname -3 ]
... ({subskript-1}...)
```

Erklärung und Regeln der Kennzeichnung siehe Abschnitt „Kennzeichnung".

Syntaxregeln für beide Formate

1. datenname-1 ist der Name des Tabellenelements. Seine Beschreibung muss entweder eine OCCURS-Klausel enthalten, oder er muss einem Datenfeld untergeordnet sein, das eine OCCURS-Klausel enthält.
2. subskript-1... kann dargestellt werden durch
   - ein ganzzahliges Literal,
   - einen Datennamen mit dem Wert einer positiven Ganzzahl,
   - relative Subskribierung,
   - einen arithmetischen Ausdruck mit dem Wert einer positiven Ganzzahl,
   - das Wort ALL.

   Der Datenname selbst darf gekennzeichnet, aber nicht indiziert werden. ALL darf nur angegeben werden, wenn der subskribierte Bezeichner als Funktionsargument angegeben ist.
3. Für jede datenname übergeordnete OCCURS-Klausel muss ein Subskript angegeben sein. Da eine Tabelle bis zu sieben Dimensionen haben kann, kann eine Bezugnahme auf ein Tabellenelement bis zu sieben Subskripte erfordern.
4. Das Subskript ist eingeschlossen in Klammern. Die öffnende Klammer folgt unmittelbar den Leerzeichen nach dem Namen des Tabellenelements (datenname). Tritt mehr als ein Subskript in einem Klammerpaar auf, können die Subskripte entweder durch Kommas getrennt werden, denen mindestens ein Leerzeichen folgt, oder nur durch Leerzeichen. Bei relativer Subskribierung müssen auch die Rechenzeichen zwischen datenname und ganzzahl durch Leerzeichen abgegrenzt werden.
5. Das Subskript bzw. eine Folge von Subskripten bezeichnet das Tabellenelement, auf das Bezug genommen werden soll. Ein Datenname, dem ein oder mehrere Subskripte beigefügt sind, heißt subskribierter Datenname oder Bezeichner.
6. Mehrere Subskripte werden in der Reihenfolge von der äußersten zur innersten Tabelle angegeben.

Allgemeine Regel für beide Formate

1. Das **Subskript** kann ein positives Vorzeichen enthalten. Der niedrigste erlaubte Wert für ein Subskript ist 1. Demzufolge ist Null oder eine negative Zahl als Wert für ein Subskript nicht zulässig. Der höchste erlaubte Wert für ein Subskript ist in jedem einzelnen Fall die Maximalzahl der Wiederholungen des Datenfeldes, die in der OCCURS-Klausel festgelegt ist.

2.5.3 Indizierung

Funktion

Indizes werden verwendet, wenn innerhalb einer Tabelle auf ein einzelnes Element zugegriffen werden soll (siehe Abschnitt „OCCURS-Klausel").

Format 1

beschreibt Indizierung ohne Kennzeichnung.

```
{datenname-1 | bedingungsname}  ({index-1 [{+ | -} ganzzahl]}... )
```

Format 2

beschreibt Indizierung mit Kennzeichnung.

```
{datenname-1 | bedingungsname} [{IN | OF} datenname-2] [{IN | OF} dateiname]
        ({index-1 [{+ | -} ganzzahl]}... )
```

Erklärung und Regeln der Kennzeichnung siehe Abschnitt „Kennzeichnung".

Syntaxregeln für beide Formate

1. datenname-1 ist der Name eines Tabellenelements.

   Ist bei datenname-1 index angegeben, muss entweder die Datenerklärung von datenname-1 eine OCCURS-Klausel mit INDEXED BY index enthalten, oder datenname-1 muss einer Gruppe untergeordnet sein, die mit der OCCURS-Klausel mit INDEXED BY index beschrieben ist.

   Zum Beispiel besagt die Bezugnahme

   `TOTAL (INDEXA, INDEXB),`

   dass TOTAL zu einer Struktur mit zwei Stufen von OCCURS-Klauseln gehört, deren jede eine INDEXED BY-Angabe enthält.
2. Der Index ist eingeschlossen in Klammern. Die öffnende Klammer folgt unmittelbar den Leerzeichen nach dem Namen des Tabellenelements (datenname). Tritt mehr als ein Index in einem Klammerpaar auf, können die Indizes entweder durch Kommas getrennt werden, denen mindestens ein Leerzeichen folgt, oder nur durch Leerzeichen.
3. Verwendet man die + ganzzahl-Angabe oder die - ganzzahl-Angabe, müssen vor und hinter den Zeichen + oder - Leerzeichen vorhanden sein.
4. Die Indizes werden in der Reihenfolge von der äußersten zur innersten Tabelle geschrieben.
5. Die niedrigste erlaubte Tabellenelementnummer für einen Index ist 1, die höchste ist in jedem einzelnen Fall die Maximalzahl der Wiederholungen des Datenfeldes. Die Maximalzahl ist in der OCCURS-Klausel festgelegt. Dies gilt auch für relative Indizierung.
6. Durch Bezugnahme auf ein Tabellenelement oder auf ein Feld innerhalb eines Tabellenelements wird der mit dieser Tabelle verbundene Index nicht verändert.
7. Durch Einsetzen der relativen Indizierung werden die Werte der Indizes im Zielprogramm nicht verändert.

Allgemeine Regeln für beide Formate

1. Die Werte von Indizes können ohne Konvertierung in Datenfelder gespeichert werden (SET-Anweisung), deren Erklärung die USAGE-Klausel mit INDEX-Angabe enthält. Diese Datenfelder heißen dann Indexdatenfelder (siehe „USAGE-Klausel“ und „SET-Anweisung“).
2. Ein Index kann nur durch eine SET-, SEARCH- oder PERFORM-Anweisung verändert werden (siehe Beschreibungen dieser Anweisungen).

2.5.4 Funktionsbezeichner

Ein Funktionsbezeichner ist eine syntaktisch korrekte Kombination von Zeichen und Trennzeichen, die eine Funktion referenziert. Das Datenfeld, das durch eine Funktion dargestellt wird, ist durch den Funktionsnamen einschließlich der eventuell vorhandenen Argumente eindeutig bezeichnet.

Format

```
FUNCTION funktionsname-1 [({argument-1}...)]
```

Syntaxregeln

1. argument-1 muss ein Bezeichner, ein Literal oder ein arithmetischer Ausdruck sein. Zu Anzahl, Klasse und Kategorie von argument-1 sind in jeder Funktionsdefinition besondere Regeln vorhanden (siehe Kapitel „Interne Standard-Funktionen").
2. Ein Funktionsbezeichner für eine alphanumerische oder nationale Funktion darf überall dort angegeben werden, wo lt. Format ein Bezeichner zugelassen ist und die Regeln zum Format nicht ausdrücklich die Referenzierung einer Funktion verbieten; die Angabe eines Funktionsbezeichners ist nicht erlaubt
   - als Empfangsoperand einer Anweisung,
   - wenn die Regeln zu den Formaten verlangen, dass das referenzierte Datenfeld Eigenschaften besitzt (wie z.B. Klasse und Kategorie, Datenformat, Länge, Vorzeichen, zulässige Werte), die weder die entsprechende Funktionsauswertung noch die einzelnen Argumente der Funktion aufweisen.
3. Ein Funktionsbezeichner für eine ganzzahlige oder numerische Funktion darf nur in einem arithmetischen Ausdruck verwendet werden.

Allgemeine Regeln

1. Die Klasse und andere Eigenschaften der Funktion sind durch die Funktionsdefinition festgelegt.
2. Die Argumente einer Funktion werden einzeln von links nach rechts, wie in der Liste der Argumente angegeben, ausgewertet. Ein Argument kann selbst ein Funktionsbezeichner oder ein Ausdruck sein, der einen Funktionsbezeichner enthält. Es ist zulässig, dass eine Funktion sich selbst referenziert (rekursive Funktion).

2.5.5 Teilfeldselektion

Funktion

Die Teilfeldselektion definiert ein Datenfeld durch die Angaben der Position des Anfangszeichens und der Länge des Datenfeldes.

Format

```
{bezeichner-1 | FUNCTION funktionsname-1 [({argument-1}... )] } (linke-
zeichenstelle: [länge])
```

bezeichner-1 und FUNCTION funktionsname-1 sind nicht Teil des Teilfeldselektors, sondern sind hier nur zur Verdeutlichung angegeben.

Syntaxregeln

1. bezeichner-1 muss eines der folgenden Datenfelder sein:
   - ein Datenelement der Kategorie alphanumerisch oder eine alphanumerische Datengruppe.
   - ein alphabetisches Datenfeld.
   - ein numerisches bzw. alphanumerisches Datenfeld oder ein numerisches Datenfeld mit USAGE DISPLAY Angabe, wobei in allen Fällen das Datenfeld nicht einer stark typisierten Datengruppe untergeordnet sein darf.
   - ein Gruppendatenfeld, das nicht stark typisiert ist.
   - ein nationales Datenfeld.
   - ein numerisch druckaufbereitetes bzw. alphanumerisch druckaufbereitetes Datenfeld oder ein numerisches Datenfeld mit USAGE DISPLAY Angabe, wobei in allen Fällen das Datenfeld nicht einer stark typisierten Datengruppe untergeordnet sein darf.
2. linke-zeichenstelle und länge müssen arithmetische Ausdrücke sein.
3. Soweit nicht anders festgelegt, darf überall dort, wo ein Bezeichner für ein Datenfeld der Klasse alphanumerisch oder national erlaubt ist, eine Teilfeldselektion angewendet werden.
4. bezeichner-1 darf nicht teilfeldselektiert sein.
5. Die durch funktionsname-1 und ihre Argumente (falls vorhanden) angesprochene Funktion muss eine alphanumerische oder nationale Funktion sein.
6. bezeichner-1 darf sich nicht auf ein Gruppendatenfeld beziehen, das Datenfelder der Klassen objekt oder zeiger enthält.

Allgemeine Regeln

1. Jedem Zeichen von bezeichner-1 bzw. funktionsname-1 ist eine Ordinalzahl zugeordnet, die von der ganz links stehenden zu der ganz rechts stehenden Stelle schrittweise um eins zunimmt. Der ganz links stehenden Stelle ist die Zahl Eins zugewiesen. Enthält die Datenerklärung für datenname-1 eine SIGN IS SEPARATE-Klausel, wird der Vorzeichenposition ebenfalls eine Ordinalzahl in diesem Datenfeld zugewiesen.
2. Ist bezeichner-1 als numerisch, numerisch druckaufbereitet, alphanumerisch oder alphanumerisch druckaufbereitet beschrieben, wird es bei der Teilfeldselektion behandelt, als ob es als alphanumerisches Datenfeld derselben Größe redefiniert würde.
3. Ist bezeichner-1 subskribiert und für ein Subskript ALL angegeben, bezieht sich der Teilfeldselektor auf jedes der implizit angesprochenen Tabellenelemente.
4. Die Teilfeldselektion erzeugt ein eindeutiges Datenfeld, das eine Teilmenge des Datenfeldes bildet, auf das sich bezeichner-1 bzw. funktionsname-1 bezieht. Dieses eindeutige Datenfeld ist folgendermaßen definiert:

- linke-zeichenstelle gibt an, ab welcher Zeichenposition von bezeichner-1 das Teilfeld beginnen soll. linke-zeichenstelle muss einen positiven ganzzahligen Wert ungleich Null ergeben, der kleiner oder gleich der Anzahl der Zeichenstellen von bezeichner-1 bzw. funktionsname-1 und seiner Argumente (falls vorhanden) ist.
- länge bezeichnet die Länge des zu erzeugenden Teilfeldes. länge muss einen positiven ganzzahligen Wert ungleich Null ergeben.
- Die Summe von linke-zeichenstelle und länge minus 1 darf nicht größer sein als die Zeichenanzahl des mit bezeichner-1 bzw. funktionsname-1 bezeichneten Datenfeldes. Ist länge nicht angegeben, erstreckt sich das erzeugte Teilfeld von der durch linke-zeichenstelle bezeichneten Position bis zum letzten Zeichen (einschließlich) des mit bezeichner-1 bzw. funktionsname-1 bezeichneten Ausgangsfeldes.

Anmerkung: Bei nationalen Daten ist sicherzustellen, dass „surrogate pairs“ durch Teilfeldselektion nicht auseinandergerissen werden.

5. Das erzeugte Teilfeld wird als Datenelement ohne JUSTIFIED-Klausel betrachtet. Das erzeugte Teilfeld hat die gleiche Klasse und Kategorie wie bezeichner-1 bzw. wie funktionsname-1 mit der Ausnahme, dass die Kategorien numerisch, numerisch druckaufbereitet und alphanumerisch druckaufbereitet als Klasse und Kategorie alphanumerisch betrachtet werden und für eine alphanumerische Gruppe USAGE DISPLAY angenommen wird.

Beispiel 2-12

Ein Datenfeld KFZNR enthält ein 10-stelliges Kfz-Kennzeichen, von dem die letzten 6 Stellen in ein Teilfeld KURZNR übertragen werden sollen:

Programmausschnitt:

```
...
01  KFZNR    PIC X(10).
01  KURZNR   PIC X(6).
...
    MOVE KFZNR (5:6) TO KURZNR.
...
```

Die „5“ in der Klammer gibt an, dass die MOVE-Operation ab dem fünften Zeichen beginnen soll, der Doppelpunkt ist das erforderliche Trennzeichen, die „6“ gibt an, dass sechs Zeichen in das Feld KURZNR übertragen werden sollen.

2.5.6 Bezeichner

Bezeichner ist ein Begriff für einen Datennamen, der, wenn er im Programm nicht eindeutig ist, durch eine syntaktisch korrekte Folge von Kennzeichnern, Subskripten oder Teilfeldselektoren eindeutig gemacht wird.

Format 1

```
FUNCTION funktionsname-1 [({argument-1}...)] [teilfeldselektor]
```

Format 2

```
datenname-1 [{IN | OF} datenname-2] ...  [{IN | OF} {dateiname-1 | listenname-1}]
        [({subskript} ...)] [teilfeldselektor]
```

2.5.7 Objektsicht (object-view)

Eine Objektsicht veranlasst den Compiler, eine Objektreferenz (z.B. im Rahmen von Konformitätsprüfungen) so zu behandeln, als ob sie in der angegebenen Form definiert wäre. Durch Überprüfungen zur Laufzeit wird nötigenfalls sichergestellt, dass der aktuelle Inhalt der Objektreferenz die geforderten Eigenschaften hat.

Format

```
bezeichner-1 AS {  [ FACTORY OF] klassenname-1 [ONLY]
                 | interfacename-11
                 | UNIVERSAL
               }
```

Syntax-Regeln

1. bezeichner-1 muss eine Objektreferenz oder ein vordefinierter Objektbezeichner sein, wobei dieser nicht SUPER und nicht NULL oder wieder eine Objektsicht sein darf.
2. bezeichner-1 darf auch ein Klassenname sein.
3. Die Objektsicht darf nicht als empfangendes Daten-Element angegeben werden.

Allgemeine Regeln

1. Eine Objektsicht gibt eine Objektreferenz zurück, welche auf dasselbe Objekt wie bezeichner-1 verweist und für Konformitätsprüfungen die in der AS-Angabe spezifizierte Schnittstelle annimmt.
2. Ist klassenname-1 ohne einen der optionalen Ausdrücke angegeben, dann ist implizit das Ergebnis USAGE IS OBJECT REFERENCE klassenname-1. Falls das Objekt, auf das bezeichner-1 verweist, kein Objekt von klassenname-1 oder einer untergeordneten Klasse von klassenname-1 ist, dann entsteht die Ausnahmesituation EC-OO-CONFORMANCE.
3. Falls FACTORY ohne ONLY-Angabe spezifiziert ist, dann ist implizit das Ergebnis USAGE OBJECT REFERENCE FACTORY OF klassenname-1. Falls das Objekt, auf welches bezeichner-1 verweist, nicht das Fabrikobjekt von klassenname-1 oder einer untergeordneten Klasse von klassenname-1 ist, dann entsteht die Ausnahmesituation EC-OO-CONFORMANCE.
4. Ist ONLY angegeben und FACTORY nicht, dann ist implizit das Ergebnis USAGE OBJECT REFERENCE klassenname-1 ONLY. Falls das Objekt, auf welches bezeichner-1 verweist, kein Objekt von klassename-1 ist, dann entsteht die Ausnahmesituation EC-OO-CONFORMANCE.
5. Ist sowohl die Angabe FACTORY als auch ONLY spezifiziert, ist implizit das Ergebnis USAGE OBJECT REFERENCE FACTORY OF klassenname-1 ONLY. Falls das Objekt, auf das bezeichner-1 verweist, nicht das Fabrik-Objekt von klassenname-1 ist, dann entsteht die Ausnahmesituation EC-OO-CONFORMANCE.
6. Ist interfacename-1 angegeben, dann ist implizit das Ergebnis USAGE OBJECT REFERENCE interfacename-1. Ist die Schnittstelle des Objekts, auf das bezeichner-1 verweist, nicht mit interfacename-1 konform, dann entsteht die Ausnahmesituation EC-OO-CONFORMANCE.
7. Ist UNIVERSAL angegeben, so ist implizit das Ergebnis USAGE OBJECTREFERENCE, ohne zusätzliche Angabe. Es tritt keine Ausnahmebedingung auf.
8. Ist die Ausnahmesituation EC-OO-CONFORMANCE entstanden und die Überprüfung für diese Ausnahmesituation eingeschaltet, so wird der zugehörige Ausnahmezustand ausgelöst und in die entsprechende USE-Prozedur verzweigt.

2.5.8 Vordefinierte Objektreferenzen

Allgemeines Format

```
{NULL | SELF | [klassenname-1 OF] SUPER}
```

2.5.8.1 NULL

NULL ist eine vordefinierte Objektreferenz, die den Referenzwert NULL enthält, der nie ein Objekt referenziert.

Format

NULL

Syntaxregeln

1. NULL darf nicht als Empfangsfeld verwendet werden.
2. NULL ist keine universelle Objektreferenz.
3. NULL ist von der Klasse Objekt.

2.5.8.2 SELF und SUPER

SELF und SUPER sind vordefinierte Objektreferenzen, die sich auf das Objekt beziehen, dessen Methode zur Zeit ausgeführt wird.

Formate

```
SELF
```

```
[klassenname-1 OF] SUPER
```

Syntaxregeln

1. SELF und SUPER dürfen nur in einer Methodendefinition verwendet werden.
2. SELF und SUPER dürfen nicht als Empfangsfeld verwendet werden.
3. SUPER darf nur als Objekt angegeben werden, für das mit der INVOKE-Anweisung eine Methode aufgerufen wird.
4. klassenname-1 ist der Name einer Klasse, der in der INHERITS-Klausel der zugehörigen Klassendefinition angegeben ist.
5. Falls die INHERITS-Klausel der zugehörigen Klassendefinition mehr als einen Klassennamen angibt, dann muss klassenname-1 spezifiziert werden.
6. Falls in der INHERITS-Klausel der zugehörigen Klassendefinition nur ein Klassenname aufgeführt ist, dann braucht klassenname-1 nicht angegeben werden.
7. SELF und SUPER sind von der Klasse Objekt und sind keine universellen Objektreferenzen.

Allgemeine Regeln

1. SELF und SUPER beziehen sich beide auf das Objekt, für das die Methode aufgerufen wurde, in der die SELF und SUPER-Angabe verwendet wird.
2. Ist SELF in einem Methodenaufruf angegeben, dann wird die Methode in der Menge der Methoden des Objekts gesucht, das zur Laufzeit dynamisch durch SELF referenziert wird.
3. Ist SUPER in einem aktuellen Methodenaufruf angegeben, dann werden bei der Methoden-Auswahl alle Methoden der Klasse ignoriert, die den Aufruf enthält und auch alle Methoden ihrer Unterklassen. Die aufgerufene Methode kann dann nur eine von einer übergeordneten Klasse (superclass) geerbte Methode sein.
4. Ist klassenname-1 angegeben, so wird nur in den dafür definierten Methoden gesucht.

2.5.9 Vordefinierte Adresse NULL

NULL ist eine vordefinierte Adresse der Datenklasse zeiger.

Format

```
NULL
```

Syntaxregel

1. Dieses Format darf nur in folgenden Fällen benutzt werden:
   - als Parameter in einer CALL-Anweisung mit einem Programmprototypen
   - als Parameter in einem Methodenaufruf
   - als Sendefeld in einer SET-Anweisung oder einer INITIALIZE-Anweisung
   - in einer Vergleichsbedingung mit einem Datenzeiger oder Programmzeiger

Allgemeine Regeln

1. Im Zusammenhang mit einem Datenzeiger bezeichnet die vordefinierte Adresse NULL ein Datenfeld von der Klasse zeiger, das keine gültige Adresse für Daten in COBOL enthält.
2. Im Zusammenhang mit einem Programmzeiger bezeichnet die vordefinierte Adresse NULL ein Datenfeld von der Datenklasse zeiger, das keine gültige Adresse für Programme in COBOL enthält.

2.5.10 Datenadressbezeichner

Format

```
ADDRESS OF bezeichner-1
```

Syntaxregeln

1. bezeichner-1 verweist auf ein Datenelement, das in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION definiert ist.
   bezeichner-1 darf nicht in der WORKING-STORAGE SECTION oder FILE SECTION eines Objektes oder eines Fabrikobjektes (factory object) definiert sein.
2. bezeichner-1 darf weder auf eine Objektreferenz noch auf einen Zeiger oder einen Index verweisen.
   bezeichner-1 darf nicht auf ein Datenelement verweisen, das einem stark typisierten Gruppendatenfeld untergeordnet ist.
3. Der Datenadressbezeichner darf nicht als Empfangsfeld verwendet werden.
4. Wenn bezeichner-1 auf ein stark typisiertes Gruppendatenfeld verweist, muss bezeichner-1 auf Stufennummer 01 beschrieben sein.

Allgemeine Regeln

1. Ein Datenadressbezeichner erzeugt ein spezifisches Datenelement der Klasse zeiger und der Kategorie datenzeiger, das die Adresse von bezeichner-1 enthält.
2. Ist bezeichner-1 stark typisiert, so ist der Datenadressbezeichner ein Datenzeiger auf ein Datenfeld vom Typ wie bezeichner-1.

2.5.11 Programmadressbezeichner

Format

```
ADDRESS OF PROGRAM {bezeichner-1 | literal-1}
```

Syntaxregeln

1. bezeichner-1 muss der Datenkategorie alphanumerisch angehören und darf nicht in der REPORT SECTION definiert sein.
   Ist bezeichner-1 der Programmname eines einzelnen Programms bzw. des äußersten Programms eines geschachtelten Programms (enthaltene Programme sind nicht erlaubt), muss er mit einem alphabetischen Zeichen beginnen und darf nur Großbuchstaben und Ziffern enthalten. Die zulässige Länge des Programmnamens ist abhängig vom Modulformat (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).
2. literal-1 muss ein alphanumerisches Literal sein; sein Wert muss ein gültiger Programmname, wie in 1. beschrieben, sein.
3. Der Programmadressbezeichner darf nicht als Empfangsfeld verwendet werden.

Allgemeine Regel

Ein Programmadressbezeichner erzeugt ein spezifisches Datenelement der Klasse zeiger und der Kategorie programmzeiger, das die Adresse eines Einsprungpunktes des Programms enthält. Der Name des Einsprungpunktes ist in bezeichner-1 oder literal-1 enthalten.
Tritt während der Bearbeitung des Programmadressbezeichners ein Fehler auf, erfolgt Programmabbruch mit Laufzeitfehlermeldung.

2.5.12 BYTE-LENGTH OF

Format

```
BYTE-LENGTH OF bezeichner-1
```

Syntaxregeln

1. BYTE-LENGTH OF bezeichner-1 darf in der PROCEDURE DIVISION überall dort angegeben werden, wo ein numerisches Literal zulässig ist. Dies gilt auch innerhalb von Subskripten und Teilfeldselektionen.
2. bezeichner-1 darf nicht teilfeldselektiert sein.
3. Ist bezeichner-1 ein Tabellenelement, so ist die Angabe eines Subskripts nicht erforderlich. Wird ein Subskript angegeben, so wertet der Compiler lediglich das Format aus.

   > ¡ Da der Wert des Subskripts für die Auswertung der Länge nicht relevant ist, stellt der Compiler nur fest, ob die Reihenfolge innerhalb des Ausdrucks richtig ist und bezeichner-1 definiert ist. Der Datentyp wird nicht überprüft.

4. bezeichner-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.

Allgemeine Regeln

1. BYTE-LENGTH OF stellt die Länge von bezeichner-1 in **Bytes** dar.
2. BYTE-LENGTH OF bezeichner-1 ist eine Konstante, die zur Übersetzungszeit berechnet wird. Ist bezeichner-1 eine Tabelle mit variabler Anzahl von Elementen, so wird zur Berechnung der Konstanten BYTE-LENGTH OF bezeichner-1 die Obergrenze aus der OCCURS-Klausel verwendet.

   > ¡ Da die Länge von Daten, die mit der ANY LENGTH-Klausel beschrieben sind, erst zur Ablaufzeit bekannt ist, kann die Länge nur durch die Funktion FUNCTION BYTE-LENGTH ermittelt werden.

2.5.13 LENGTH OF

Format

```
LENGTH OF bezeichner-1
```

Syntaxregeln

1. LENGTH OF bezeichner-1 darf in der PROCEDURE DIVISION überall dort angegeben werden, wo ein numerisches Literal zulässig ist. Dies gilt auch innerhalb von Subskripten und Teilfeldselektionen.
2. bezeichner-1 darf nicht teilfeldselektiert sein.
3. Ist bezeichner-1 ein Tabellenelement, so ist die Angabe eines Subskripts nicht erforderlich. Wird ein Subskript angegeben, so wertet der Compiler lediglich das Format aus.

   > ¡ Da der Wert des Subskripts für die Auswertung der Länge nicht relevant ist, stellt der Compiler nur fest, ob die Reihenfolge innerhalb des Ausdrucks richtig ist und bezeichner-1 definiert ist. Der Datentyp wird nicht überprüft.

4. bezeichner-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.

Allgemeine Regeln

1. LENGTH OF stellt die Länge von bezeichner-1 in **Zeichen** dar.
2. LENGTH OF bezeichner-1 ist eine Konstante, die zur Übersetzungszeit berechnet wird. Ist bezeichner-1 eine Tabelle mit variabler Anzahl von Elementen, so wird zur Berechnung der Konstanten LENGTH OF bezeichner-1 die Obergrenze aus der OCCURSKlausel verwendet.

   > ¡ Da die Länge von Daten, die mit der ANY LENGTH-Klausel beschrieben sind, erst zur Ablaufzeit bekannt ist, kann die Länge nur durch die Funktion FUNCTION LENGTH ermittelt werden.
   >
   > Die Anwendung von LENGTH OF ist nur sinnvoll mit USAGE DISPLAY oder USAGE NATIONAL.

2.5.14 Bedingungsname

Wird explizit darauf Bezug genommen, muss ein Bedingungsname eindeutig sein oder durch Kennzeichnung und/oder Subskribierung eindeutig gemacht werden. Dies ist nicht erforderlich, wenn die Eindeutigkeit der Bezugnahme durch die Namenskonventionen für den Gültigkeitsbereich selbst gewährleistet ist.

Wird die Kennzeichnung benutzt, um einen Bedingungsnamen eindeutig zu machen, kann die dazugehörige Bedingungsvariable als erster Kennzeichner verwendet werden. Außerdem muss bei der Kennzeichnung die Hierarchie der Namen, die der Bedingungsvariablen zugeordnet sind, verwendet werden, um einen Bedingungsnamen eindeutig zu machen.

Wenn die Bezugnahme auf eine Bedingungsvariable eine Subskribierung erfordert, ist bei der Bezugnahme auf einen ihrer Bedingungsnamen dieselbe Kombination von Subskripten erforderlich.

Bei der Kennzeichnung und Subskribierung von Bedingungsnamen gelten dasselbe Format und dieselben Einschränkungen wie für die Bezeichner, mit der Ausnahme, dass datenname-1 durch bedingungsname-1 ersetzt wird.

Im allgemeinen Format der folgenden Kapitel bezieht sich „bedingungsname-n“ immer auf einen Bedingungsnamen, der, je nach den Erfordernissen, gekennzeichnet oder subskribiert ist.

2.6 Tabellenbearbeitung

Eine Tabelle ist eine Folge von gleich großen Datenfeldern. Diese Felder sind die Tabellenelemente. Sie haben alle denselben Aufbau und werden fortlaufend abgespeichert. Die gesamte Tabelle selbst bildet auch ein Datenfeld im Sinne von COBOL.

Probleme, die bei der Verarbeitung vieler gleichstrukturierter Daten auftreten, können oft besser gelöst werden, wenn diese Daten in Tabellenform aufgebaut sind. Dadurch ist es möglich, Informationen wirkungsvoll zu interpretieren und sinnvoll darzustellen.

Der gleiche Aufbau der einzelnen Tabellenelemente macht ihre Beziehungen untereinander deutlich.

Das einzelne Tabellenelement belegt einen einfach zu bestimmenden physischen Platz relativ zur Basis der Tabelle, d.h. dem Anfang der Tabelle im Arbeitsspeicher. Deshalb ist jedes Element relativ zum Beginn der Tabelle adressierbar und braucht nicht mit einem eigenen Datennamen angesprochen zu werden. Der Zugriff zu einem Tabellenelement erfolgt anhand der Tabellenelementnummer (siehe „Subskribierung" und „Indizierung").

Außerdem kann für einen gegebenen Wert eines Tabellenelementes die zugehörige Tabellenelementnummer bestimmt werden (siehe Abschnitt „SEARCH-Anweisung").

Tabellen können eine (während der Ablaufzeit) veränderliche Anzahl von Tabellenelementen haben (siehe Beispiel 7-6 auf "Datenname- oder FILLER-Klausel ").

2.6.1 Tabellendefinition

Ein Tabellenelement wird in der Datenerklärung durch Angabe der OCCURS-Klausel gekennzeichnet. Mit der OCCURS-Klausel wird festgelegt, wieviele Tabellenelemente die Tabelle umfasst. Name und Beschreibung des Tabellenelements gelten für jede Wiederholung. Bei einer mehrdimensionalen Tabelle muss für jede Dimension in der hierarchischen Ordnung eine OCCURS-Klausel angegeben sein.

Beispiel 2-13

```
01  TABELLE.
    02 TABELLEN-ELEMENT PIC XXX OCCURS 20 TIMES.
```

Das Datenfeld TABELLE umfasst 20 gleichgroße Datenfelder.
Diese Felder heißen TABELLEN-ELEMENT:

```
TABELLE:  1. TABELLEN-ELEMENT (1)   PIC XXX.
          2. TABELLEN-ELEMENT (2)   PIC XXX.
          .
          .
          .
         20. TABELLEN-ELEMENT (20)  PIC XXX.
```

Eindimensionale Tabellen

In der Datenbeschreibung des Tabellenelements wird die OCCURS-Klausel angegeben.

Beispiel 2-14

```
01  TABELLE.
    02 TABELLEN-ELEMENT OCCURS 2 TIMES.
       03 ELEMENT-FELD-1      PIC X(4).
       03 ELEMENT-FELD-2      PIC X(4).
```

TABELLE ist der Name der Tabelle.

TABELLEN-ELEMENT ist das Element der eindimensionalen TABELLE, das zweimal auftritt.

ELEMENT-FELD-1 und ELEMENT-FELD-2 sind Elemente, die TABELLEN-ELEMENT untergeordnet sind.

Mehrdimensionale Tabellen

Ist ein Datenfeld dem Tabellenelement einer zweidimensionalen Tabelle untergeordnet und enthält es eine OCCURS-Klausel, so ist dieses Datenfeld Element einer dreidimensionalen Tabelle.

Für eine Tabelle sind maximal sieben Dimensionen erlaubt.

Beispiel 2-15

```
01  TABELLE.
    02 BLK OCCURS 2 TIMES.
       03 SATZ  OCCURS 2 TIMES.
          04 FELD  OCCURS 2 TIMES PIC X(10).
```

BLK ist ein Element einer eindimensionalen Tabelle, das zweimal auftritt.

SATZ ist ein Element einer zweidimensionalen Tabelle, das zweimal innerhalb eines jeden Auftretens von BLK auftritt.

FELD ist ein Element einer dreidimensionalen Tabelle, das zweimal innerhalb eines jeden Auftretens von SATZ auftritt.

TABELLE	BLK (1)	SATZ (1, 1)	FELD (1, 1, 1)
			FELD (1, 1, 2)
		SATZ (1, 2)	FELD (1, 2, 1)
			FELD (1, 2, 2)
	BLK (2)	SATZ (2, 1)	FELD (2, 1, 1)
			FELD (2, 1, 2)
		SATZ (2, 2)	FELD (2, 2, 1)
			FELD (2, 2, 2)

Bild 3: Darstellung von TABELLE

Anfangswerte von Tabellenelementen

Eine VALUE-Klausel darf in einer Datensatzbeschreibung mit OCCURS-Klausel oder einer ihr untergeordneten Datensatzbeschreibung enthalten sein. Für die Definition von Bedingungsnamen ist die VALUE-Klausel auch hier erlaubt und notwendig.

Mit Hilfe der VALUE-Klausel können in der WORKING-STORAGE SECTION oder LOCAL-STORAGE SECTION Anfangswerte einer Tabelle angeben werden.

Beispiel 2-16

```
 WORKING-STORAGE SECTION.
****** 1. VALUE-ANGABE AUF GRUPPENEBENE ******

 01  WOCHE  VALUE
           "MONTAG    DIENSTAG  MITTWOCH  DONNERSTAG
           "FREITAG   SAMSTAG   SONNTAG   ".
     02  TAG  PIC X(10) OCCURS 7 TIMES.

****** 2. VALUE-ANGABE IN DER OCCURS-KLAUSEL ******

 01  WEEK.
     02  WDAY  PIC X(10) OCCURS 7 TIMES VALUE FROM (1)
               "MONDAY" "TUESDAY" "WEDNESDAY" "THURSDAY"
               "FRIDAY" "SATURDAY" "SUNDAY".
****** 3. VALUE-ANGABE IN EINEM DER OCCURS-KLAUSEL UNTERGEORDNETEN FELD
*****

 01  UGE.
     02  FILLER      OCCURS 7 TIMES.
         03  DAG   PIC X(10) VALUE FROM (1)
             "MANDAG" "TISDAG" "ONSDAG" "TORSDAG"
             "FREDAG" "LOERDAG" "SOENDAG".
```

Bezugnahmen auf Tabellenelemente

Die Tabellenelemente einer Tabelle haben denselben Datennamen. Man unterscheidet die Tabellenelemente dadurch, dass man die in Klammern eingeschlossenen Tabellenelementnummern (Indizes) hinter dem Datennamen einfügt.

Beispiel 2-17

```
01  TABELLE.
    02  ELEMENT  OCCURS 10 TIMES.
          .
          .
          .
    MOVE ELEMENT OF TABELLE (8) TO ...
```

Hierbei wird auf das achte Tabellenelement zugegriffen.
Für jede Dimension muss eine Tabellenelementnummer angegeben werden.

Es gibt zwei Techniken, Tabellenelemente anzusprechen:

- Subskribierung
- Indizierung

2.6.2 Subskribierung

Eine Methode, die Elementnummer anzugeben, besteht darin, dem Datennamen einen oder mehrere Subskripte beizufügen. Ein Subskript ist eine ganze Zahl, deren Wert die Elementnummer eines Tabellenelements oder eines der Felder, die diesem Tabellenelement untergeordnet sind, angibt. Das Subskript kann dargestellt werden

- durch ein ganzzahliges Literal,
- durch einen Datennamen, der als numerisches Datenelement definiert ist, das keine Zeichenstellen rechts vom angenommenen Dezimalpunkt hat,
- durch einen arithmetischen Ausdruck, der weder ein direktes noch ein relatives Subskript ist.

In jedem Fall müssen die Subskripte in Klammern eingeschlossen sein und dem Namen des Tabellenelements unmittelbar folgen. Einem Tabellenelement müssen genausoviele Subskripte beigefügt sein, wie die zugehörige Tabelle Dimensionen hat. Es muss also für jede OCCURS-Klausel, einschließlich der, die den Datennamen innerhalb der definierten Hierarchie enthält, ein Subskript angegeben werden.

Im Beispiel 2-15 im Abschnitt "Tabellendefinition" (dreidimensionale Tabelle) benötigt man:

- ein Subskript bei Bezugnahmen auf BLK
- zwei Subskripte bei Bezugnahmen auf SATZ
- drei Subskripte bei Bezugnahmen auf FELD.

Die Subskripte werden in der Reihenfolge von der äußersten zur innersten Tabelle geschrieben.

So bezeichnet z.B.

FELD (1, 2, 2)

das zweite Element FELD
innerhalb des zweiten Elements SATZ
innerhalb des ersten Elements BLK.

Eine Bezugnahme auf ein Datenfeld darf nur dann subskribiert werden, wenn das Datenfeld ein Tabellenelement oder ein Datenfeld bzw. ein Bedingungsname innerhalb eines Tabellenelements ist.

Es gibt drei Formen der Subskribierung:

- Direkte Subskribierung
- Relative Subskribierung
- Subskribierung mittels eines arithmetischen Ausdrucks

Direkte Subskribierung

Bei der direkten Subskribierung wird das Subskript entweder durch ein ganzzahliges Literal oder durch einen Datennamen angegeben. Der Datenname muss als numerisches Datenelement erklärt sein, das keine Zeichenstellen rechts vom angenommenen Dezimalpunkt hat. Im vorhergehenden Beispiel wurde direkte Subskribierung verwendet.

Relative Subskribierung

Folgt dem Namen des Tabellenelements ein Subskript in Form von

```
(datenname + ganzzahl-1),
```

errechnet sich die benötigte Tabellenelementnummer aus dem Wert, den datenname zur Programmablaufzeit hat, plus ganzzahl-1.

Nimmt man die Form

```
(datenname - ganzzahl-2)
```

erhält man die Tabellenelementnummer, indem vom Wert datenname ganzzahl-2 subtrahiert wird.

Relative Subskribierung wird genauso behandelt wie relative Indizierung. Weitere Erklärungen siehe „Indizierung".

Subskribierung mittels eines arithmetischen Ausdrucks

Ein Subskript kann aus einem arithmetischen Ausdruck bestehen, der als Ergebnis eine ganze Zahl liefert.

Besteht ein Subskript aus einem arithmetischen Ausdruck, der weder ein direktes noch ein relatives Subskript ist, so errechnet sich die benötigte Tabellenelementnummer aus dem Wert, den der arithmetische Ausdruck zur Programmablaufzeit hat.

Sowohl zur Übersetzungs- als auch zur Ablaufzeit werden arithmetische Ausdrücke als Subskripte langsamer verarbeitet als direkte oder relative Subskripte. Aus diesem Grund ist das Vertauschen von datenname und ganzzahl in relativen Subskripten ebenso zu vermeiden, wie das Einschließen eines direkten bzw. relativen Subskripts in runde Klammern, da solche Ausdrücke als arithmetische Ausdrücke gelten.

Endet ein Subskript mit einem Datennamen oder Index, so darf ein unmittelbar nachfolgendes Subskript nicht mit einer öffnenden Klammer beginnen, da diese die Subskribierung des Datennamens bzw. Indexes einleiten würde.

2.6.3 Indizierung

Die Indizierung ist eine weitere Methode, auf Tabellenelemente Bezug zu nehmen.Die Indizierung ist möglich, wenn die INDEXED BY-Angabe in der OCCURS-Klausel angegeben ist.

Der Index benötigt keine eigene Datenerklärung. Zur Programmablaufzeit ist der Wert eines Index eine Zahl in Binärform, die die Distanz zum Tabellenbeginn angibt. Der Wert dieser Binärzahl lässt sich aus der Nummer und Länge des Tabellenelements wie folgt errechnen:

Binärwert des Index = (Tabellenelementnummer - 1) * Tabellenelementlänge.

Der Wert eines Index kann nur mit der SET-, SEARCH- und PERFORM-Anweisung gesetzt werden. Der Anfangswert ist undefiniert und muss explizit gesetzt werden.

Es gibt zwei Formen der Indizierung:

- Direkte Indizierung
- Relative Indizierung

Direkte Indizierung

Direkte Indizierung liegt dann vor, wenn man einen Index in der Art eines direkten Subskripts benutzt.

Beispiel 2-18

```
01 TABELLE.
   02  TABELLE-A PIC XX OCCURS 10 TIMES INDEXED BY INDEX-A.
   02  TABELLE-B PIC X(3) OCCURS 5 TIMES INDEXED BY INDEX-B.
   ...
   PROCEDURE DIVISION.
       ...
       SET INDEX-A TO 7.
       ...
       MOVE "X7" TO TABELLE-A (INDEX-A).
       ...
```

Es werden zwei Tabellen definiert:

- TABELLE-A mit 10 Elementen der Länge 2 Byte
- TABELLE-B mit 5 Elementen der Länge 3 Byte

Durch die INDEXED BY-Angabe wird der Index INDEX-A für TABELLE-A und der Index INDEX-B für TABELLE-B vereinbart. Indizes dürfen nur mit dem entsprechenden Datenelement benutzt werden, z.B. TABELLE-A (INDEX-A) oder TABELLE-B(INDEX-B).

Die SET-Anweisung setzt den Index auf einen Wert, der auf das siebte Element von TABELLE-A zeigt. Die Distanz zum Anfang der Tabelle, also der interne binäre Inhalt von INDEX-A, ist (7-1) * 2 = 12. So überträgt die MOVE-Anweisung X7 in das siebte Tabellenelement.

Relative Indizierung

Folgt dem Namen eines Tabellenelements ein Index in Form von

(index + ganzzahl-1),

errechnet sich die benötigte Tabellenelementnummer aus dem Wert, den index zur Programmablaufzeit hat, plus ganzzahl-1.

Nimmt man die Form

(index - ganzzahl-2),

erhält man die Tabellenelementnummer, indem man ganzzahl-2 von der entsprechenden Tabellenelementnummer subtrahiert.

Durch Einsetzen der relativen Indizierung werden die Werte der Indizes im Zielprogramm nicht verändert.

Erlaubte Wertebereiche für Indizes

Der Wert eines Index sollte lt. Norm einer gültigen Tabellenelementnummer der zugehörigen Tabelle entsprechen. Dieser Compiler lässt jedoch auch 0, ZERO oder negativen Zahlen entsprechende Elementnummern und Werte jenseits der höchsten erlaubten Elementnummer zu, mit der Einschränkung, dass der Binärwert des Index in dem (in 4 Byte darstellbaren) Bereich von -2^{31} bis $+2^{31}-1$ bleibt. Vor der Verwendung muss in diesen Fällen der Index auf eine gültige Elementnummer gesetzt werden (z.B. mit SET UP bzw. SET DOWN) oder es muss durch entsprechende relative Indizierung dafür gesorgt werden, dass nur gültige Tabellenelemente angesprochen werden.

2.6.4 Vergleich von Subskribierung und Indizierung

Verfügbarkeit der Tabellenelementnummer für den Benutzer

Subskribierung:

Die Tabellenelementnummer steht unmittelbar zur Verfügung.

Indizierung:

Die Tabellenelementnummer steht nur nach einer vorherigen SET-, SEARCH- oderPERFORM-Anweisung zur Verfügung und wird wie folgt errechnet:

Wert des Index geteilt durch Tabellenelementlänge plus 1.

Bezugnahmen auf Tabellenelemente

Subskribierung:

Aus Subskripten (außer in Form von Literalen) muss während der Programmablaufzeit die Adresse des Tabellenelements immer wieder neu berechnet werden; d.h. dass solche subskribierten Datenfelder nicht so schnell angesprochen werden können wie Datenfelder außerhalb einer Tabelle.

Indizierung:

Bei der Indizierung ist die Bezugnahme auf ein Tabellenelement schneller als bei der Subskribierung mit Bezeichnern, da bereits die Distanz zum Tabellenanfang im Index abgespeichert ist.

Verändern des Index

Subskribierung:

Ein Subskript in Form eines Datennamens (mit MOVE, ADD usw.) kann schneller verändert werden als ein Index mit SET, weil bei der SET-Anweisung die Elementnummer erst noch in die Distanz zum Tabellenanfang umgerechnet werden muss. Das gilt dann, wenn der Index nicht um einen festen Wert herauf- oder herabgesetzt wird.

Indizierung:

Ein Index kann mit PERFORM- oder SEARCH-Anweisungen schneller verändert werden als ein Subskript.

Gültigkeit

Subskribierung:

Ein Subskript kann auch für andere Tabellenelemente verwendet werden.

Indizierung:

Ein Index darf nur mit seinem Tabellenelement benutzt werden (außer in den Anweisungen SET, PERFORM und SEARCH).

2.7 Anweisungen und Programmsätze

Es gibt vier Arten von Anweisungen:

- bedingte Anweisungen
- Übersetzungssteueranweisungen
- unbedingte Anweisungen
- explizit begrenzte Anweisungen

Desgleichen gibt es drei Arten von Programmsätzen:

- bedingte Programmsätze
- Übersetzungsprogrammsätze
- unbedingte Programmsätze

2.7.1 Bedingte Anweisungen und bedingte Programmsätze

- Eine **bedingte Anweisung** dient dazu, den Wahrheitswert einer Bedingung zu bestimmen und den von diesem Wahrheitswert abhängigen folgenden Programmschritt festzulegen.
  Zu den bedingten Anweisungen gehören:
  - die IF-, EVALUATE-, SEARCH- und RETURN-Anweisung,
  - eine OPEN DOCUMENT-Anweisung, die die (NOT) AT-END-Angabe enthält,
  - eine READ-Anweisung, die die (NOT) AT END- oder (NOT) INVALID KEY-Angabe enthält,
  - eine WRITE-Anweisung, die die (NOT) INVALID KEY- oder (NOT) END-OF-PAGE-Angabe enthält,
  - eine START-, REWRITE- oder DELETE-Anweisung, die die (NOT) INVALID KEY-Angabe enthält,
  - eine ADD-, COMPUTE-, DIVIDE-, MULTIPLY- oder SUBTRACT-Anweisung, die die (NOT) ON SIZE ERROR-Angabe enthält,
  - eine STRING- oder UNSTRING-Anweisung, die die (NOT) ON OVERFLOW-Angabe enthält,
  - eine CALL-Anweisung, die die ON OVERFLOW- oder (NOT) ON EXCEPTION-Angabe enthält.
  - eine ACCEPT- oder DISPLAY-Anweisung, die die (NOT) ON EXCEPTION-Angabe enthält.
  - eine INVOKE-Anweisung, die die (NOT) ON EXCEPTION-Angabe enthält.
- Ein **bedingter Programmsatz** ist eine bedingte Anweisung, der eine unbedingte Anweisung vorangehen kann und die durch einen Punkt und ein nachfolgendes Leerzeichen abgeschlossen wird.

2.7.2 Übersetzungssteueranweisungen

Eine **Übersetzungssteueranweisung** besteht aus einem der Übersetzungssteuerverben COPY, REPLACE oder USE und den zugehörigen Operanden bzw. aus einer der folgenden Compiler Direktiven:

- >>CALL-CONVENTION
- >>DEFINE
- >>EVALUATE
- >>FLAG-85
- >>IF
- >>IMP
- >>LISTING
- >>PAGE
- >>SOURCE

Eine Übersetzungssteueranweisung veranlasst den Compiler zu bestimmten Aktionen während der Übersetzung.

2.7.3 Unbedingte Anweisungen und unbedingte Programmsätze

- Eine unbedingte Anweisung veranlasst, dass im Programm eine bestimmte Aktion durchgeführt wird.
- Eine unbedingte Anweisung ist eine Anweisung, die mit einem unbedingten Verb beginnt und angibt, dass eine Aktion unbedingt ausgeführt werden muss, oder eine bedingte Anweisung, die durch einen expliziten Bereichsbegrenzer abgeschlossen wird.
- Eine unbedingte Anweisung kann aus einer Folge von unbedingten Anweisungen, die jeweils durch ein Trennungszeichen abgetrennt sind, bestehen. Zu den unbedingten Anweisungen gehören:

ACCEPT (7)	FREE	READ (4)
ADD (1)	GENERATE	RELEASE
ALLOCATE	GO TO	RESUME
ALTER	INITIALIZE	REWRITE (2)
CALL (6)	INITIATE	SET
CANCEL	INSPECT	SORT
CLOSE	INVOKE (7)	START (2)
CLOSE DOCUMENT	MERGE	STOP
COMPUTE (1)	MOVE	STRING (3)
CONTINUE	MULTIPLY (1)	SUBTRACT (1)
DELETE (2)	OPEN	TERMINATE
DISPLAY (7)	OPEN DOCUMENT (8)	UNSTRING (3)
DIVIDE (1)	PERFORM	WRITE (5)
EXIT	RAISE	XML (7)

(1) ohne die (NOT) ON SIZE ERROR-Angabe

(2) ohne die (NOT) INVALID KEY-Angabe

(3) ohne die (NOT) ON OVERFLOW-Angabe

(4) ohne die (NOT) AT END- oder (NOT) INVALID KEY-Angabe

(5) ohne die (NOT) INVALID KEY- oder (NOT) END-OF-PAGE-Angabe

(6) ohne die ON OVERFLOW- oder (NOT) ON EXCEPTION-Angabe

(7) ohne die (NOT) ON EXCEPTION-Angabe

(8) ohne die (NOT) AT END-Angabe

- Wenn unbedingte-anweisung im Anweisungsformat auftritt, so ist damit auch eine Folge von unbedingten Anweisungen gemeint, die mit einem Punkt oder mit einer Angabe abgeschlossen ist, die zu einer Anweisung gehört, die unbedingte-anweisung enthält. So ist z.B.

  ```
  DIVIDE A INTO B.
  ```

  eine unbedingte Anweisung wegen des abschließenden Punkts;bzw. in

  ```
  READ D AT END
         DIVIDE A INTO B
         NOT AT END ...
  ```

  ist `DIVIDE` eine unbedingte Anweisung wegen `'NOT ...'` .

- Ein unbedingter Programmsatz ist eine unbedingte Anweisung, die durch einen Punkt und ein nachfolgendes Leerzeichen abgeschlossen wird.

2.7.4 Explizit begrenzte Anweisungen

Eine explizit begrenzte Anweisung ist jede Anweisung, die einen expliziten Bereichsbegrenzer enthält.

2.7.5 Bereichsbegrenzer (Scope Terminators)

Bereichsbegrenzer legen den Gültigkeitsbereich bestimmter Anweisungen in der PROCEDURE DIVISION fest. Anweisungen, die einen expliziten Bereichsbegrenzer enthalten, werden explizit begrenzte Anweisungen genannt.
Der Gültigkeitsbereich von Anweisungen, die innerhalb von anderen Anweisungen stehen, kann auch implizit begrenzt werden.
Der Punkt, der einen Programmsatz abschließt, begrenzt implizit auch alle Anweisungen, die im Gültigkeitsbereich einer äußeren Anweisung stehen.

Bei geschachtelten Anweisungsfolgen begrenzt die nächste Angabe in der äußeren Anweisung den Gültigkeitsbereich jeder nicht abgeschlossenen inneren Anweisung.Steht eine explizit begrenzte Anweisung innerhalb einer anderen explizit begrenzten Anweisung mit demselben reservierten Wort, begrenzt jeder einzelne explizite Bereichsbegrenzer die unmittelbar vorhergehende Anweisung.
Sind Anweisungen in andere Anweisungen, die eine Bedingungsangabe erlauben, geschachtelt, wird jede Bedingungsangabe als nächste Angabe der unmittelbar vorhergehenden nicht abgeschlossenen Anweisung betrachtet.
Eine Anweisung, die noch nicht explizit oder implizit begrenzt wurde, gilt als nicht abgeschlossene Anweisung.

Neben dem Punkt (impliziter Bereichsbegrenzer) können zur Unterstützung der Strukturierten Programmierung folgende explizite Bereichsbegrenzer eingesetzt werden:

END-ACCEPT	END-DIVIDE	END-PERFORM	END-START
END-ADD	END-EVALUATE	END-READ	END-STRING
END-CALL	END-IF	END-RECEIVE	END-SUBTRACT
END-COMPUTE	END-INVOKE	END-RETURN	END-UNSTRING
END-DELETE	END-MULTIPLY	END-REWRITE	END-WRITE
END-DISPLAY	END-OPEN	END-SEARCH	END-XML

Die expliziten Bereichsbegrenzer sind reservierte COBOL-Wörter.

2.8 Verarbeiten eines COBOL-Programms

Der COBOL-Compiler erzeugt aus einer COBOL-Übersetzungseinheit mit Hilfe eines Binders ein ablauffähiges Programm. Dieses ablauffähige Programm wird auch als **Zielprogramm** bezeichnet.

Der **Binder** hat die Aufgabe, die vom Compiler erzeugten Moduln mit den erforderlichen Laufzeitroutinen und ggf. weiteren Moduln zu verknüpfen.

Die **Laufzeitroutinen**, die als Moduln vorgegeben sind, werden zur Ausführung spezieller COBOL-Funktionen benutzt, z.B. zur Ein-/Ausgabe.

Zum symbolischen und maschinennahen Testen von COBOL-Programmen steht die Dialogtesthilfe AID zur Verfügung.

Zu weiteren Einzelheiten siehe COBOL-2000-Benutzerhandbuch [1].

2.9 EBCDIC-Zeichensatz

Vom Compiler verwendete Version des 8-Bit-Codes

Decimal	Hexadecimal	EBCDIC	Printer graphics	
0	00	0000 0000		(LOW-VALUE)
...	...	...		
64	40	0100 0000		(Leerzeichen)
...	...	...		
74	4A	0100 1010	c	(Centzeichen)
75	4B	0100 1011	.	(Punkt)
76	4C	0100 1100	<	(Kleinerzeichen)
77	4D	0100 1101	(	(öffnende runde Klammer)
78	4E	0100 1110	+	(Pluszeichen)
79	4F	0100 1111	|	(senkrechter Strich)
80	50	0101 0000	&	(kaufm. Undzeichen)
...	...	...		
90	5A	0101 1010	!	(Ausrufungszeichen)
91	5B	0101 1011	$	(Dollarzeichen)
92	5C	0101 1100	*	(Stern)
93	5D	0101 1101	)	(schließende runde Klammer)
94	5E	0101 1110	;	(Semikolon)
95	5F	0101 1111		(Nichtzeichen, logisch)
96	60	0110 0000	-	(Minuszeichen)
97	61	0110 0001	/	(Schrägstrich)
98	62	0110 0010	§	(Paragrafenzeichen)
99	63	0110 0011	[	(öffnende eckige Klammer)
100	64	0110 0100	]	(schließende eckige Klammer)
...	...	...		
103	67	0110 0111	ß	
...	...	...		
106	6A	0110 1010	^	(Undzeichen, logisch)
107	6B	0110 1011	,	(Komma)
108	6C	0110 1100	%	(Prozentzeichen)
109	6D	0110 1101	_	(Tiefstrich)
110	6E	0110 1110	>	(Größerzeichen)
111	6F	0110 1111	?	(Fragezeichen)

....	....	....	...	
122	7A	0111 1010	:	(Doppelpunkt)
123	7B	0111 1011	#	(Nummernzeichen)
124	7C	0111 1100	@	(kommerzielles a)
125	7D	0111 1101	'	(Hochkomma)
126	7E	0111 1110	=	(Gleichheitszeichen)
127	7F	0111 1111	"	(Doppelhochkomma)
...	...	...		
129	81	1000 0001	a	
130	82	1000 0010	b	
131	83	1000 0011	c	
132	84	1000 0100	d	
133	85	1000 0101	e	
134	86	1000 0110	f	
135	87	1000 0111	g	
136	88	1000 1000	h	
137	89	1000 1001	i	
138	8A	1000 1010		
139	8B	1000 1011	Ä	
140	8C	1000 1100	Ö	
141	8D	1000 1101	Ü	
...	...	...		
145	91	1001 0001	j	
146	92	1001 0010	k	
147	93	1001 0011	l	
148	94	1001 0100	m	
149	95	1001 0101	n	
150	96	1001 0110	o	
151	97	1001 0111	p	
152	98	1001 1000	q	
153	99	1001 1001	r	
...	...	...		
159 ...	9F ...	1001 1111 ...	allg. Währungszeichen bzw. € (je nach Zeichensatz)	
162	A2	1010 0010	s	
163	A	1010 0011	t	
164	A4	1010 0100	u	

165	A5	1010 0101	v
166	A6	1010 0110	w
167	A7	1010 0111	x
168	A8	1010 1000	y
169	A9	1010 1001	z
170	AA	1010 1010	
171	AB	1010 1011	ä
172	AC	1010 1100	ö
173	AD	1010 1101	ü
...	...	...	
192	C0	1100 0000	{
193	C1	1100 0001	A
194	C2	1100 0010	B
195	C3	1100 0011	C
196	C4	1100 0100	D
197	C5	1100 0101	E
198	C6	1100 0110	F
199	C7	1100 0111	G
200	C8	1100 1000	H
201	C9	1100 1001	I
...	...	...	
208	D0	1101 0000	}
209	D1	1101 0001	J
210	D2	1101 0010	K
211	D3	1101 0011	L
212	D4	1101 0100	M
213	D5	1101 0101	N
214	D6	1101 0110	O
215	D7	1101 0111	P
216	D8	1101 1000	Q
217	D9	1101 1001	R
...	...	...	
226	E2	1110 0010	S
227	E3	1110 0011	T
228	E4	1110 0100	U
229	E5	1110 0101	V

230	E6	1110 0110	W	
231	E7	1110 0111	X	
232	E8	1110 1000	Y	
233	E9	1110 1001	Z	
...	...	...		
240	F0	1111 0000	0	
241	F1	1111 0001	1	
242	F2	1111 0010	2	
243	F3	1111 0011	3	
244	F4	1111 0100	4	
245	F5	1111 0101	5	
246	F6	1111 0110	6	
247	F7	1111 0111	7	
248	F8	1111 1000	8	
249	F9	1111 1001	9	
...	...	...		
255	FF	1111 1111	~	(Tilde) (HIGH-VALUE)

Hinweis

COBOL2000 unterstützt das Eurozeichen dahingehend, dass das Eurozeichen in Zeichenketten und Kommentaren eines COBOL-Programms zugelassen wird. Das Eurozeichen wird auch in der CURRENCY-SIGN-Klausel zugelassen und dessen Verwendung in der PICTURE-Klausel unterstützt. Näheres dazu siehe [12], BS2000/OSD Softbooks.

3 Steuerung des Compilers

Für die Steuerung des Compilers stehen zum einen Anweisungen zur Quelltextmanipulation und zum anderen Compiler-Direktiven zur Verfügung. Die folgende Übersicht zeigt, welche speziellen Anweisungen und Compiler-Direktiven es gibt und in welchem logischen Schritt der Übersetzung sie wirksam sind (zur Erläuterung der Phasen siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]):

Anweisungen zur Quelltextmanipulation	**Phase**
COPY-Anweisung	Quelltextmanipulation
SUPPRESS-Option	Listen-Erzeugungsphase
REPLACE-Anweisung	Quelltextmanipulation

Compiler-Direktiven	**Phase**
CALL-CONVENTION	Übersetzungsphase
DEFINE	Quelltextmanipulation
EVALUATE	Quelltextmanipulation
FLAG-85	Übersetzungsphase
IF	Quelltextmanipulation
IMP	Übersetzungsphase und Listen-Erzeugungsphase (abhängig vom Operanden)
LISTING	Listen-Erzeugungsphase
PAGE	Listen-Erzeugungsphase
SOURCE FORMAT	Quelltextmanipulation
TURN	Übersetzungsphase

3.1 Anweisungen zur Quelltextmanipulation

Mit den Anweisungen COPY und REPLACE kann ein vorgegebener Quelltext durch Texte aus einer Bibliothek ergänzt bzw. durch anderen Text ersetzt werden. Die beiden Anweisungen, die zum Zeitpunkt der Quelltextmanipulation wirksam werden, können sowohl unabhängig voneinander als auch miteinander gekoppelt verwendet werden.

Mit einer COPY-Anweisung wird ein vorgegebener Quelltext durch COBOL-Textzeilen aus einem Bibliothekselement ergänzt. Der Compiler behandelt den Ergänzungstext als Teil der Übersetzungseinheit. Die REPLACING-Angabe bietet die zusätzliche Möglichkeit, jedes Auftreten eines Literals, eines Bezeichners, eines Wortes oder einer Gruppe von Wörtern durch einen anderen Text zu ersetzen.

Die REPLACE-Anweisung bewirkt das Ersetzen von Teilen des Quelltextes durch neuen Text. Damit können Übersetzungseinheiten, die vom Programmierer in selbstdefinierter Notation geschrieben wurden, zur Übersetzungszeit in syntaktisch korrekte Angaben, Klauseln und Anweisungen umgewandelt werden.

Der Quelltext wird vom Compiler erst dann syntaktisch geprüft, wenn alle COPY- und REPLACE-Anweisungen vollständig ausgeführt sind.

3.1.1 COPY-Anweisung

Funktion

Die COPY-Anweisung fügt Text aus einer Bibliothek in eine Übersetzungseinheit ein, wobei Teile des Textes ersetzt werden können.

Format

```
COPY textname [{OF | IN} bibliotheksname] [SUPPRESS]
[REPLACING {     {  wort-1                BY  {  wort-2                }... ].
                  | bezeichner-1                   | bezeichner-2
                  | literal-1                     | literal-2
                  | ==pseudotext-1==              | ==pseudotext-2==
                 }                              }
```

Syntaxregeln

1. textname ist der Name des Textes, der in die Übersetzungseinheit hineinkopiert wird. Der Text ist unter diesem Namen als Bibliothekselement in einer Bibliothek gespeichert. textname ist ein 1 bis 31 Zeichen langer benutzerdefinierter Name.
2. Ist textname durch bibliotheksname qualifiziert, wird die angegebene Bibliothek nach dem Text durchsucht. Wenn textname nicht durch bibliotheksname qualifiziert ist, werden bis zu zehn Bibliotheken nach dem Text durchsucht.
   Die Zuordnung von Bibliotheken während der Übersetzungszeit ist im Handbuch „COBOL2000 Benutzerhandbuch“ [1] näher beschrieben.
3. Vor der COPY-Anweisung muss ein Leerzeichen oder ein anderes Trennsymbol stehen.
4. pseudotext-1 muss mindestens ein Textwort (siehe "Textwort" im Abschnitt „Begriffserklärungen") enthalten, während pseudotext-2 auch leer (====) sein oder nur Leerzeichen, Komma, Semikolon und Kommentarzeilen enthalten kann.
5. Textwörter in pseudotext-1 und pseudotext-2 und Textwörter, die bezeichner-1, bezeichner-2, literal-1, literal-2, wort-1, wort-2 bilden, können in der nächsten Zeile fortgesetzt werden. Das Pseudotext-Begrenzungszeichen (==) darf hingegen nicht fortgesetzt werden.
6. wort-1 und wort-2 kann jedes einzelne COBOL-Wort außer COPY sein.
7. Kleinbuchstaben, die in Textwörtern verwendet werden, sind äquivalent zu den entsprechenden Großbuchstaben.
8. Eine COPY-Anweisung wird innerhalb eines Programm überall erkannt, ausgenommen
   - in Kommentarzeilen,
   - innerhalb nichtnumerischer Literale.
9. Der durch eine COPY-Anweisung erzeugte Text darf keine COPY-Anweisung enthalten. Der hier beschriebene Compiler lässt dies jedoch im äußersten COPY zu: Enthält eine COPY-Anweisung keine REPLACING-Angabe, dann darf der zugehörige Bibliothekstext COPY-Anweisungen enthalten; diese können auch eine REPLACING-Angabe haben.

Allgemeine Regeln

1. Die Ausführung einer COPY-Anweisung bewirkt, dass der mit textname angegebene Bibliothekstext in die Übersetzungseinheit kopiert wird. Der Bibliothekstext ersetzt dabei die gesamte COPY-Anweisung (einschließlich des abschließenden Trennsymbols Punkt).
2. Wenn SUPPRESS angegeben ist, wird der Bibliothekstext in die Übersetzungseinheitliste nicht aufgeführt.
3. Ist REPLACING nicht angegeben, wird der Bibliothekstext unverändert kopiert.

COPY...REPLACING...

1. Ist REPLACING angegeben, wird der Bibliothekstext kopiert, wobei der Text vor BY (A-Operanden) durch den Text nach BY (B-Operanden) ersetzt wird.
2. Eine Textersetzung findet nur statt, wenn der Text vor BY für jedes Textwort Zeichen für Zeichen mit dem entsprechenden Bibliothekstext übereinstimmt.
3. bezeichner-1, wort-1 und literal-1 werden wie ein Pseudotext behandelt, der nur bezeichner-1, wort-1 bzw. literal-1 enthält.
4. Der Vergleich zwischen dem Bibliothekstext und dem vor BY angegebenen Text (A-Operanden), von dessen Ergebnis abhängt, ob eine Textersetzung stattfindet oder nicht, verläuft folgendermaßen:
   - Alles vor dem ersten Textwort im Bibliothekstext wird in die Übersetzungseinheit kopiert. Beginnend mit dem ersten der A-Operanden (pseudotext-1, bezeichner-1, wort-1, literal-1), werden der Reihe nach alle A-Operanden mit der gleichen Anzahl von entsprechenden Bibliothekstextwörtern, jeweils beginnend mit dem ersten Bibliothekstextwort, verglichen.
   - Jedes der Trennsymbole Komma (,'BLANK'), Semikolon (;'BLANK') oder Leerzeichen in pseudotext-1 oder im Bibliothekstext wird als einzelnes Leerzeichen betrachtet. Jede Folge von einem oder mehreren Leerzeichen als Trennsymbol wird als einzelnes Leerzeichen betrachtet.
   - Wird keine Übereinstimmung festgestellt, wird der Vergleich mit den folgenden A-Operanden wiederholt, bis entweder eine Übereinstimmung gefunden wird, oder kein A-Operand mehr folgt.
   - Wenn alle A-Operanden der REPLACING-Angabe verglichen worden sind und keine Übereinstimmung festgestellt wurde, wird das erste Bibliothekstextwort in die Übersetzungseinheit kopiert. Das nächste Bibliothekstextwort wird dann als das
     erste Wort angesehen und der Vergleichsvorgang beginnt erneut mit dem ersten A-Operanden.
   - Jedesmal, wenn eine Übereinstimmung zwischen einem A-Operanden und dem Bibliothekstext auftritt, wird der zugehörige B-Operand (pseudotext-2, bezeichner-2, wort-2 oder literal-2) in die Übersetzungseinheit übertragen und er setzt den mit dem A-Operanden übereinstimmenden Bibliothekstext. Das nächste Bibliothekstextwort nach dem letzten ersetzten Textwort wird dann als erstes Textwort betrachtet, und der Vergleich wird, beginnend mit dem ersten A-Operanden, wiederholt.
   - Der Vergleichszyklus wird so lange fortgesetzt, bis das letzte Bibliothekstextwort entweder kopiert oder ersetzt worden ist.
5. Kommentar- oder Leerzeilen im Bibliothekstext und in pseudotext-1 werden beim Vergleich wie ein Leerzeichen behandelt. Kommentar- oder Leerzeilen in pseudotext-2 werden ggf. unverändert in die Übersetzungseinheit übertragen. Kommentar- oder Leerzeilen im Bibliothekstext werden unverändert in die Übersetzungseinheit kopiert, es sei denn, sie stehen innerhalb einer Textwortfolge, die mit pseudotext-1 übereinstimmt und deshalb ersetzt wird.
6. Testhilfezeilen sind im Bibliothekstext und im Pseudotext erlaubt. Textwörter in einer Testhilfezeile unterliegen den Vergleichsregeln, als ob kein „D“ im Anzeigebereich (Spalte 7) der Testhilfezeile stünde. Eine Testhilfezeile ist innerhalb eines Pseudotexts spezifiziert, wenn der öffnende Pseudotextbegrenzer auf einer Zeile steht, die der Testhilfezeile vorangeht.
7. Jedes aus der Bibliothek kopierte, aber nicht ersetzte Textwort wird so kopiert, dass es im gleichen Bereich (Programmtext-Bereich) steht wie im Bibliothekstext.
   Bei einer Ersetzung durch Pseudotext steht jedes Textwort im Programmtext-Bereich, wie in pseudotext-2 angegeben. Text aus einer Bibliothek wird in den gleichen Bereich der Übersetzungseinheit kopiert, in dem er in der Bibliothek steht. Deshalb muss er mit den Regeln des COBOL-Referenzformats übereinstimmen.
8. Steht eine COPY-Anweisung in einer Testhilfezeile, wird der kopierte Text so behandelt, als ob er ebenfalls in Testhilfezeilen stünde. Dies gilt auch für zusätzliche, auf Grund von Ersetzungen n der Übersetzungseinheit entstehende Zeilen.

9. Wenn durch eine COPY-Anweisung mit REPLACING-Angabe eine Zeile so verlängert wird, dass zusätzliche Zeilen erforderlich sind, dann werden entsprechende zusätzliche Zeilen erzeugt, die eventuell in Spalte 7 ein Fortsetzungskennzeichen enthalten.
10. Um Teile eines Textworts zu ersetzen, muss die zu ersetzende Zeichenkette im Bibliothekstext und im A-Operanden der REPLACING-Angabe durch geeignete Trennsymbole (z.B. Doppelpunkt oder runde Klammern) eingeschlossen und damit als eigenes Textwort gekennzeichnet werden (siehe Beispiel 3-4).
11. Das Referenzformat, das für die COPY-Anweisung gilt, muss dasselbe sein, das auch für die ggf. zu ersetzenden Teile im Bibliothekstext gilt.

Beispiel 3-1

Der Bibliothekstext namens ADR steht in der Bibliothek ADRLIB und besteht aus folgenden Zeilen:

```
05  STRASSE    PIC X(20).
05  PLZ        PIC 9(5).
05  ORT        PIC X(20).
05  LAND       PIC X(20).
```

In der Übersetzungseinheit wird die COPY-Anweisung wie folgt geschrieben:

```
01  ADRESSE.
    COPY ADR OF ADRLIB.
```

Nach Ausführung der COPY-Anweisung steht in der Übersetzungseinheit:

```
01  ADRESSE.
    COPY ADR OF ADRLIB.    ————————————————————————— (1)
    05 STRASSE     PIC X(20).
    05 PLZ         PIC 9(5).
    05 ORT         PIC X(20).
    05 LAND        PIC X(20).
```

(1) Die COPY-Anweisung erscheint in der Übersetzungseinheitliste, wird aber während der Übersetzung als Kommentar behandelt.

Beispiel 3-2

Um den Bibliothekstext zu ändern, wird in der COPY-Anweisung die REPLACING-Angabe verwendet:

```
COPY ADR OF ADRLIB
 REPLACING ADRESSE BY ADDRESS
           STRASSE BY STREET
           ==PLZ PIC 9(5)== BY ==POSTCODE PIC X(8)==
           ORT BY TOWN
           LAND BY COUNTRY
```

Nach Ausführung der COPY-Anweisung steht in der Übersetzungseinheit:

```
01  ADRESSE.
    COPY ADR OF ADRLIB                                    (1)
```

```
      REPLACING ADRESSE BY ADDRESS                                  |
                STRASSE BY STREET                                   |
                ==PLZ PIC 9(5)== BY ==POSTCODE PIC X(6)==           |
                ORT BY TOWN                                         |
                LAND BY COUNTRY.                                   (1)
     05  STREET PIC X(20).
     05  POST CODE PIC X(6).
     05  TOWN PIC X(20).
     05  COUNTRY PIC X(20).
```

(1) Die COPY-Anweisung erscheint in der Übersetzungseinheitliste, wird aber während der Übersetzung als Kommentar behandelt.

Die Änderungen gelten nur für diese Übersetzungseinheit. Der Text in der Bibliothek bleibt unverändert.

Beispiel 3-3

Um im Bibliothekstext die Zeichenkette

```
... PIC X(30).
```

durch die Zeichenkette

```
... PIC X(40).
```

zu ersetzen, muss in pseudotext-1 und pseudotext-2 nach dem Punkt ein Leerzeichen eingefügt werden, da sonst nach einem Punkt gesucht wird, während im Bibliothekstext ein Trennzeichen Punkt steht:

```
...REPLACING ==PIC X(30).'BLANK'== BY ==PIC X(40).'BLANK'==
```

Beispiel 3-4

Der Bibliothekstext in ELEM

```
01 FILLER
   02 :prefix:-name PIC X(30).
   02 :prefix:-address PIC X(30).
```

wird durch die COPY-Anweisungen

```
...
COPY ELEM OF COPYLIB REPLACING ==:prefix:== BY ==in==.
COPY ELEM OF COPYLIB REPLACING ==:prefix:== BY ==out==.
...
```

expandiert zu:

```
...
01 FILLER
   02 in-name PIC X(30).
   02 in-address PIC X(30).
01 FILLER
   02 out-name PIC X(30).
   02 out-address PIC X(30).
...
```

3.1.2 REPLACE-Anweisung

Funktion

Die REPLACE-Anweisung wird verwendet, um Quelltext zu ersetzen.

Format 1

```
REPLACE {==pseudotext-1== BY ==pseudotext-2==}... .
```

Format 2

```
REPLACE OFF.
```

Syntaxregeln

1. Einer REPLACE-Anweisung muss - wenn sie nicht die erste Anweisung in einem separat übersetzten Programm ist - ein Trennzeichen Punkt oder ein anderes Trennsymbol vorangehen. Die REPLACE-Anweisung muss beendet werden, bevor eine andere REPLACE-Anweisung begonnen werden kann.
2. Eine REPLACE-Anweisung wird wird innerhalb eines Programms überall erkannt, ausgenommen
   - in Kommentarzeilen,
   - innerhalb nichtnumerischer Literale.
3. pseudotext-1 muss mindestens ein Textwort enthalten, während pseudotext-2 auch leer (====) sein oder nur Leerzeichen, Komma, Semikolon und Kommentarzeilen enthalten kann.

Allgemeine Regeln

1. Format-1 der REPLACE-Anweisung gibt an, dass jedes Auftreten von pseudotext-1 durch pseudotext-2 ersetzt werden soll.
2. Format-2 der REPLACE-Anweisung bewirkt, dass die laufende Textersetzung abgebrochen wird.
3. Eine REPLACE-Anweisung gilt von ihrer Spezifizierung bis zur nächsten REPLACE-Anweisung oder bis zum Ende eines getrennt übersetzten Programms.
4. Alle REPLACE-Anweisungen, die in einer Übersetzungseinheit oder einem Bibliothekstext enthalten sind, werden erst nach der Ausführung aller COPY-Anweisungen in der Übersetzungseinheit oder Bibliothekstext abgearbeitet. Die Operanden einer
   REPLACE-Anweisung können nicht durch eine REPLACING-Angabe geändert werden.
5. Der Text, der durch die Ausführung einer REPLACE-Anweisung entsteht, darf weder eine REPLACE- noch eine COPY-Anweisung enthalten.
6. Kleinbuchstaben, die in Textwörtern verwendet werden, sind äquivalent zu den entsprechenden Großbuchstaben.
7. Die Vergleichsoperation, die über Textersetzung entscheidet, läuft auf folgende Weise ab (Übersetzungseinheittext und Bibliothekstext sind unter dem Begriff „Quelltext“ zusammengefasst):
   - Beginnend mit dem ersten auf die REPLACE-Anweisung folgenden Textwort des Quelltexts und dem ersten Wort von pseudotext-1, wird pseudotext-1 mit der glei
     chen Anzahl von aufeinanderfolgenden Textwörtern verglichen.
   - pseudotext-1 stimmt genau dann mit dem Quelltext überein, wenn die Folge von Textwörtern in pseudotext-1 Zeichen für Zeichen mit der entsprechenden Textwort
     folge im Quelltext übereinstimmt.
   - Wenn keine Übereinstimmung festgestellt wird, wird der Vergleich mit jedem folgenden Auftreten von pseudotext-1 ausgeführt, bis entweder eine Übereinstimmung
     festgestellt wird oder kein pseudotext-1 mehr vorkommt.

- Wenn jeder angegebene pseudotext-1 verglichen wurde, ohne dass eine Übereinstimmung gefunden wurde, wird das nächste Textwort des Quelltexts als erstes Textwort betrachtet. Der Vergleich beginnt erneut mit dem ersten pseudotext-1.
- Wenn eine Übereinstimmung zwischen pseudotext-1 und dem Programmtext festgestellt wird, wird diese Stelle durch den entsprechenden pseudotext-2 ersetzt.
  Das Textwort im Quelltext, das unmittelbar auf den ersetzten Text folgt, wird dann als erstes Textwort angesehen. Der Vergleich beginnt erneut mit dem ersten pseudotext-1.
- Die Vergleichsoperation läuft solange, bis das letzte von der REPLACE-Anweisung betroffene Textwort entweder ersetzt wurde oder als erstes Textwort mit jedem pseudotext-1 verglichen wurde.

8. Kommentar- oder Leerzeilen, die im Quelltext oder im pseudotext-1 vorkommen, werden beim Vergleich wie ein Leerzeichen behandelt. Die Reihenfolge von Textwörtern im Quelltext und im pseudotext-1 ist durch das Referenzformat festgelegt (siehe Abschnitt „Anweisungen und Programmsätze"). Kommentar- oder Leerzeilen von pseudotext-2 werden ggf. unverändert in die Übersetzungseinheit übertragen. Eine Kommentar- oder Leerzeile im Quelltext, die in einer Folge von Textwörtern steht, die mit pseudotext-1 übereinstimmt, erscheint nicht mehr im endgültigen Programmtext.
9. Testhilfezeilen sind im Pseudotext erlaubt. Textwörter in pseudotext-1, die in einer Testhilfezeile stehen, werden beim Vergleich berücksichtigt, als ob kein ’D’ im Anzeigebereich (Spalte 7) stünde.
10. Außer bei den COPY- und REPLACE-Anweisungen selbst kann die syntaktische Rich-tigkeit von Quelltexten nicht festgestellt werden, bevor nicht alle COPY- und REPLACE-Anweisungen vollständig ausgeführt worden sind.
11. Textwörter, die als Ergebnis einer REPLACE-Anweisung eingesetzt werden, werden entsprechend dem Referenzformat in den Programmtext gebracht.
    Wenn Textwörter von pseudotext-2 in den Programmtext gebracht werden, werden zusätzliche Leerzeichen nur eingefügt, wo im Originaltext oder pseudotext-2 Leerzeichen stehen. Eingeschlossen ist dabei das implizite Leerzeichen zwischen den Übersetzungseinheitzeilen.
12. Wenn als Ergebnis der Ausführung einer REPLACE-Anweisung zusätzliche Zeilen in die Übersetzungseinheit eingefügt werden, enthält der Anzeigebereich der eingefügten Zeilen das Zeichen, das die erste Zeile des ersetzten Bereichs enthalten hat.
    Ausnahme: Wenn die Übersetzungseinheitzeile einen Bindestrich enthält, steht in der eingefügten Zeile ein Leerzeichen.
    Wenn durch eine REPLACE-Anweisung eine Zeile so verlängert wird, dass zusätzliche Zeilen erforderlich sind, dann werden entsprechende zusätzliche Zeilen erzeugt, die eventuell in Spalte 7 ein Fortsetzungskennzeichen enthalten.
    Wenn die Ersetzung die Fortsetzung des Literals in einer Testhilfezeile verlangt, ist die Übersetzungseinheit fehlerhaft.
13. Das Referenzformat, das für die REPLACE-Anweisung gilt, muss dasselbe sein, das für die ggf. zu ersetzenden Teile im Bibliothekstext gilt.

3.2 Compiler-Direktiven

Allgemeines

Compiler-Direktiven ermöglichen es dem COBOL-Programmierer, Optionen für die Übersetzung anzugeben und die Quelltext-Manipulation zu steuern. Insbesondere kann mit Hilfe von Compiler-Direktiven die bedingte Compilierung gesteuert werden.

Syntaxregeln für alle Compiler-Direktiven

1. Einer Compiler-Direktive können beliebig viele Leerzeichen vorangehen.
2. Eine Compiler-Direktive muss, bezogen auf das Referenzformat, im ProgrammtextBereich stehen und es dürfen ihr nur Leerzeichen folgen.
3. Eine Compiler-Direktive beginnt mit der Zeichenfolge „>>“, gefolgt von optionalen Leerzeichen und der eigentlichen Anweisung.
4. Eine Compiler-Direktive muss allein in einer einzigen Zeile stehen. Ausnahmen siehe EVALUATE-Direktive ("EVALUATE-Direktive ") und IF-Direktive ("IF-Direktive ").
5. Eine Compiler-Direktive darf auch in Bibliothekstext geschrieben werden (siehe COPY, Abschnitt „Anweisungen zur Quelltextmanipulation").
6. EVALUATE- und IF-Direktive müssen in COPY-Elementen innerhalb des einkopierten Textes mit dem zugehörigen >>END-EVALUATE bzw. >>END-IF abgeschlossen sein.
7. Die maximale Schachtelungstiefe von Direktiven beträgt 256.
8. Eine Compiler-Direktive darf überall angegeben werden, außer:
   - die Nutzung wird für bestimmte Direktiven durch die Syntaxregeln eingeschränkt.
   - innerhalb einer Anweisung zur Quelltextmanipulation (siehe Abschnitt „Anweisungen zur Quelltextmanipulation")
   - zwischen fortgesetzter und Fortsetzungszeile
   - in einer Testhilfezeile

Allgemeine Regeln für alle Compiler-Direktiven

1. Eine Compiler-Direktive ist von keiner „Replacing-Aktion“ einer COPY- oder REPLACE-Anweisung betroffen.
2. Compiler-Direktiven wirken entweder in der Phase der Quelltextmanipulation, in der Listen-Erzeugungsphase oder in der Übersetzungsphase (siehe Übersicht auf "Steuerung des Compilers"). Während der Übersetzungsphase werden die Compiler-Direktiven in der Reihenfolge abgearbeitet, wie sie in der Übersetzungsgruppe auftreten.
3. Compiler-Direktiven gelten für alle nachfolgenden Quelltexte und Bibliothekstexte und sind vom Programmfluss unabhängig.
4. Compiler-Direktiven werden als **vor** einer Übersetzungseinheit stehend betrachtet, wenn sie nicht zwischen der äußersten IDENTIFICATION DIVISION (bzw., wenn diese fehlt, dem ID-Paragrafen) und dem zugehörigen END-Eintrag angegeben sind.
5. Enthält eine Übersetzungseinheit mehr als 65535 Zeilen[1], so werden Direktiven, die normalerweise während der Übersetzungsphase wirken, ignoriert, wenn sie innerhalb dieser Übersetzungseinheit angegeben sind.

Arithmetischer Ausdruck

Ein arithmetischer Ausdruck kann bestehen aus ganzen Zahlen, Konstantennamen sowie den Zeichen „(“, „)“, „+ “, „-“, „*“, „/“. Der Wertebereich muss für alle Zwischenergebnisse als 32-bit-Zahl darstellbar sein. Nachkommastellen werden ggf. abgeschnitten.

Für den Wertebereich des durch den arithmetischen Ausdruck repräsentierten Literals sowie für den Wertebereich aller bei der Berechnung des arithmetischen Ausdrucks anfallenden Zwischenergebnisse muss gelten:

$-2^{31} < \text{Wertebereich} < 2^{31}-1$

Boolescher Ausdruck

Als Boolescher Ausdruck werden bezeichnet:

- eine Vergleichsbedingung, in der beide Operanden Literale oder arithmetische Ausdrücke, gebildet aus Literalen, sind. Stellvertretend für Literale sind dabei auch Namen von Compilervariablen aus DEFINE-Direktiven zugelassen.
- DEFINED-Bedingung
- Ausdrücke, die durch die logischen Operatoren AND, OR, NOT und eventuell notwendigen Klammerungen aus booleschen Ausdrücken zusammengesetzt sind.

  Ausdrücke in Direktiven, die logische Operanden (AND bzw. OR) enthalten, erfordern eine vollständige Klammerung, um die Reihenfolge der Ausführung dieser logischen Operatoren festzulegen. Lediglich für arithmetische Teilausdrücke kann entsprechend der normalen Vorrangsregeln auf weitergehende Klammerungen verzichtet werden.

Bedingte Übersetzung

Mit Hilfe von bestimmten Compiler-Direktiven können ausgewählte Quellcode-Zeilen bei der Übersetzung entweder weggelassen oder mit übersetzt werden. Dieser Vorgang heißt bedingte Übersetzung. Dafür sind die DEFINE-, EVALUATE- und IF-Direktive vorgesehen.

DEFINED-Bedingung

Format

```
compilervariablenname-1 IS [NOT] DEFINED
```

Syntaxregel

1. compilervariablenname-1 darf kein für die Compiler-Direktiven reserviertes Wort sein (siehe „Reservierte Wörter für Compiler-Direktiven" im Abschnitt „COBOL-Wörter").

Allgemeine Regel

1. Wenn IS DEFINED angegeben und compilervariablenname-1 definiert ist, ergibt die

   Bedingung den Wert „wahr“. Wenn IS NOT DEFINED angegeben ist und compilervariablenname-1 nicht definiert ist, ergibt die Bedingung ebenfalls „wahr“.

   compilervariablenname-1 ist definiert, wenn folgende Voraussetzungen erfüllt sind:

   - compilervariablenname-1 wird in einer DEFINE-Direktive (siehe „DEFINE-Direktive") ohne OFF-Klausel spezifiziert.
   - Falls in der DEFINE-Direktive die PARAMETER-Klausel angegeben ist und der Wert vom Betriebssystem zur Verfügung gestellt wurde (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

[1] Dabei zählen Kommentar- und Leerzeilen sowie Compiler-Direktiven mit, die vor der IDENTIFICATION DIVISION angegeben und diese Übersetzungseinheit zugeordnet werden.

3.2.1 CALL-CONVENTION-Direktive

Die CALL-CONVENTION-Direktive steuert die Behandlung von Programm- und Methodennamen und bestimmt die Generierung der Schnittstelle bei Aufrufen von Programmen und Methoden.

Format

```
>>CALL-CONVENTION { COMPATIBLE
                  | COBOL
                  | ILCS
                  | ILCS-SET-RETURN-CODE
                  }
```

Allgemeine Regeln

1. Ist die CALL-CONVENTION-Direktive innerhalb einer Anweisung angegeben, so wirkt sie erst für die nächste Anweisung.

   > **i** Für WHEN-Angaben in EVALUATE- bzw. SEARCH-Anweisungen gilt dieselbe CALL-CONVENTION-Angabe, wie bei der EVALUATE- bzw. SEARCH-Anweisung selbst.

2. Ist die CALL-CONVENTION-Direktive nicht angegeben, so wird >>CALL-CONVENTION COMPATIBLE angenommen.
3. Die nachfolgende Tabelle zeigt die Wirkungen der Operanden der CALL-CONVENTION-Direktive.

Übersicht zu CALL-CONVENTION

Wirkung bei	Konvention			
	COMPATIBLE	COBOL	ILCS	ILCS-SET-RETURN-CODE
Programmname externes Programm (CALL...AS prototype (a))	auf groß umgesetzt	auf groß umgesetzt	auf groß umgesetzt	auf groß umgesetzt
Programmname externes Programm (CALL sonst (b), CANCEL, ADDRESS OF PROGRAM)	unverändert	auf groß umgesetzt	unverändert	unverändert
Programmname eingeschachteltes Programm (CALL...AS NESTED)	auf groß umgesetzt	auf groß umgesetzt	auf groß umgesetzt	auf groß umgesetzt
Programmname eingeschachteltes Programm (CALL sonst (b), CANCEL)	auf groß umgesetzt	auf groß umgesetzt	- (c)	- (c)
Methodenname (INVOKE mit Objektrefe renz (d))	auf groß umgesetzt	auf groß umgesetzt	auf groß umgesetzt	auf groß umgesetzt
Methodenname (INVOKE sonst (e))	unverändert	auf groß umgesetzt	unverändert	unverändert
oberstes Bit setzen im letzten Parameter	entsprechend Option	nein	nein	nein
Register1 nach Rückkehr aus dem Gerufenen ins Sonderregister RETURN-CODE übertragen	entsprechend Option ACTIVATE-XPG4-RETURNCODE	nein	nein	ja

(a) prototypename aus COBOL-Programm ohne PROTOTYPE-Angabe entstanden und Repository-Information erzeugt von COBOL2000 Compiler Version > V1.2A

(b) ohne prototypename oder

prototypename aus COBOL-Programm mit PROTOTYPE-Angabe entstanden oder Repository-Information erzeugt von COBOL2000 Compiler Version <= V1.2A

(c) Aufruf von eingeschachtelten Programmen nur mit AS NESTED möglich

(d) Objektreferenz SELF, SUPER oder

definiert als USAGE OBJECT REFERENCE mit klassenname oder ACTIVE-CLASS

(e) universelle Objektreferenz oder

Objektreferenz definiert als USAGE OBJECT REFERENCE mit interfacename

3.2.2 DEFINE-Direktive

Die DEFINE-Direktive definiert einen symbolischen Namen für numerische Literale oder Zeichenketten-Literale, die in einer IF-Direktive, einer EVALUATE-Direktive oder der Konstantendefinition einer DEFINE-Direktive verwendet werden.

Format

```
>>DEFINE compilation-variable-name AS {{arithmetical-expression | literal |
PARAMETER} [OVERRIDE] | OFF}
```

Syntaxregeln

1. Wenn weder die OFF- noch die OVERRIDE-Angabe verwendet wird, muss eine der folgenden Bedingungen gelten:
   - compilervariablenname darf innerhalb der Übersetzungsgruppe noch nicht definiert sein.
   - Die letzte vorhergehende DEFINE-Direktive, die sich auf compilervariablenname bezieht, muss denselben Wert oder die OFF-Klausel spezifizieren.
2. literal darf ein ganzzahliges numerisches Literal oder ein nicht hexadezimales alphanumerisches Literal sein (siehe "Literale "). Die Länge des Literals ist durch den in einer Zeile verfügbaren Platz begrenzt.
3. arithmetischer-ausdruck ist wie auf "Compiler-Direktiven " beschrieben zu bilden.

Allgemeine Regeln

1. Die DEFINE-Direktive wird zeitgleich mit der Ausführung der COPY- und REPLACE-Anweisung bearbeitet.
2. Die definierten Compilervariablen dürfen nur in anderen Direktiven an Stelle von Literalen, aber nicht im Programmtext verwendet werden.
3. Mit der OVERRIDE-Angabe wird compilervariablenname ohne Rücksicht auf ihre bisherige Verwendung mit dem neuen Wert definiert.
4. Mit der PARAMETER-Angabe wird der Wert der Compilervariablen dem Compiler beim Start übergeben (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).
5. Bei Angabe von OFF oder wenn bei der Übergabe der betreffenden Compilervariable ein Fehler aufgetreten ist, erhält compilervariablenname den Status „nicht definiert“.

3.2.3 EVALUATE-Direktive

Die EVALUATE-Direktive unterstützt die bedingte Übersetzung von mehreren Alternativen.

Format 1

```
>>EVALUATE {arithmetischer-ausdruck-1 | boolescher-ausdruck-1}

{>>WHEN {arithmetischer-ausdruck-2 | boolescher-ausdruck-2}
    [{THROUGH | THRU} arithmetischer-ausdruck-3]
       [quelltext-1]}...

    [>>WHEN OTHER
       [quelltext-2]]

    >>END EVALUATE
```

Format 2

```
>>EVALUATE TRUE
    {>>WHEN boolescher-ausdruck-1
       [quelltext-1]}...
    [>>WHEN OTHER
       [quelltext-2]]
    >>END-EVALUATE
```

Syntaxregeln

Alle Formate

1. THRU ist die Abkürzung für THROUGH.
2. Die mit >> eingeleiteten Teile der Compiler-Direktive müssen zusammen mit den nachfolgenden Operanden jeweils in einer eigenen Zeile stehen.
   quelltext-1 und quelltext-2 müssen in einer neuen Zeile beginnen.
3. quelltext-1 und quelltext-2 dürfen beliebige Codezeilen enthalten, Compiler-Direktiven eingeschlossen.
   quelltext-1 und quelltext-2 können jeweils mehrere Zeilen umfassen.
4. In quelltext-1 bzw. quelltext-2 darf maximal 1 COPY-Anweisung in einer Zeile stehen.

Format 1

1. Alle Operanden einer EVALUATE-Direktive müssen jeweils von der selben Art sein. Bei dieser Regel ist ein arithmetischer Ausdruck von der Art numerisch und ein boolescher Ausdruck von der Art boolesch.
2. Wenn die Angabe THROUGH definiert ist, müssen alle Auswahl-Elemente von der Art numerisch sein.

Allgemeine Regeln

Alle Formate

1. Die EVALUATE-Direktive wird zeitgleich mit der Ausführung der COPY- und REPLACE-Anweisung bearbeitet.

2. Sobald die Auswertung einer WHEN Angabe 'wahr' ergibt, wird der zugehörige quelltext-1 übersetzt und alle anderen Zeilen, einschließlich >>END-EVALUATE, ignoriert. Ergibt keine der WHEN Angaben den Wert 'wahr', dann wird quelltext-2, sofern angegeben, übersetzt und alle anderen Zeilen ignoriert.

Format 1

1. arithmetischer-ausdruck-1 bzw. boolescher-ausdruck-1 werden mit den Werten in der WHEN-Angabe nach folgenden Regeln verglichen:
   - ist THROUGH nicht angegeben, dann ergibt sich der Wert 'wahr', wenn arithmetischer-ausdruck-1 bzw. boolescher-ausdruck-1 gleich ist mit arithmetischer-ausdruck-2 bzw. boolescher-ausdruck-2,
   - ist THROUGH angegeben, dann ergibt sich der Wert 'wahr', wenn arithmetischer-ausdruck-1 im Wertebereich von arithmetischer-ausdruck-2 und arithmetischer-ausdruck-3 liegt.

3.2.4 FLAG-85-Direktive

Die FLAG-85-Direktive ermöglicht es, Sprachmittel durch den Compiler anzeigen zu lassen, deren Verhalten sich im Vergleich zum Standard ANS85 geändert hat oder exakt festgelegt wurde.

Format

```
>>FLAG-85 ZERO-LENGTH {ON | OFF}
```

Allgemeine Regeln

1. Ist die FLAG-85-Direktive innerhalb einer Anweisung angegeben, so wirkt sie erst für die nächste Anweisung.

   > ¡ Für WHEN-Angaben in EVALUATE- bzw. SEARCH-Anweisungen gilt dieselbe FLAG-85-Angabe, wie bei der EVALUATE- bzw. SEARCH-Anweisung selbst.

2. Ist die FLAG-85-Direktive nicht angegeben, so wird >>FLAG-85 ZERO-LENGTH OFF angenommen.
3. Falls ON angegeben oder angenommen ist, wird die Überprüfung der Sprachmittel für den nachfolgenden Quelltext eingeschaltet. Die Überprüfung bleibt eingeschaltet, bis sie durch eine weitere FLAG-85-Direktive mit der Angabe OFF ausgeschaltet wird.
4. Falls OFF angegeben ist, wird die Überprüfung der Sprachmittel für den nachfolgenden Quelltext ausgeschaltet. Die Überprüfung bleibt ausgeschaltet, bis sie durch eine FLAG-85-Direktive mit der Angabe ON eingeschaltet wird.
5. Bei eingeschalteter FLAG-85-Direktive werden für folgende Situationen Compilermeldungen ausgegeben:
   - Eine READ-Anweisung, die einen Datensatz mit Länge 0 liefern kann.
   - Eine Anweisung, in der ein potentiell null-längiges Datenfeld angesprochen wird.

3.2.5 IF-Direktive

Die IF-Direktive unterstützt die bedingte Übersetzung von einer bzw. zwei Alternativen.

Format

```
>>IF boolescher-ausdruck-1
     [quelltext-1]

   [>>ELSE
      [quelltext-2]]

    >>END-IF
```

Syntaxregeln

1. Die mit >> eingeleiteten Teile der Compiler-Direktive müssen zusammen mit den nachfolgenden Operanden jeweils in einer eigenen Zeile stehen.
   quelltext-1 und quelltext-2 müssen in einer neuen Zeile beginnen.
2. Die Zeilen müssen so geschrieben werden, wie im Format angegeben, wobei quelltext-1 und quelltext-2 jeweils mehrere Zeilen umfassen können. In den Quelltexten dürfen auch Compiler-Direktiven enthalten sein.
3. In quelltext-1 bzw. quelltext-2 darf maximal 1 COPY-Anweisung in einer Zeile stehen.

Allgemeine Regeln

1. Die IF-Direktive wird zeitgleich mit der Ausführung der COPY- und REPLACE-Anweisung bearbeitet.
2. Wenn boolescher-ausdruck-1 wahr ist, wird quelltext-1, sofern vorhanden, übersetzt und quelltext-2 ignoriert.
3. Wenn boolescher-ausdruck-1 nicht wahr ist, wird quelltext-1 ignoriert und quelltext-2, sofern vorhanden, wird übersetzt.

3.2.6 IMP-Direktive

Die IMP-Direktive ermöglicht durch Angabe direkt im Quellprogramm eine Steuerung des Compilers.

3.2.7 IMP COMPILER-ACTION

Durch diese Direktive können Aktionen des Compilers bei der Modul-Generierung gesteuert werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

3.2.8 IMP LISTING-OPTIONS

Durch diese Direktive können die Werte von Compiler-Optionen, die die vom Compiler erzeugten Listen beeinflussen, geändert werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

3.2.9 IMP PRINT-DIRECTIVES

Durch diese Direktive können die Werte von Direktiven in die Quellprogrammliste ausgegeben werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

3.2.10 IMP RUNTIME-ERRORS

Durch diese Direktive kann die Überprüfung und Behandlung von Laufzeitfehlern gesteuert werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

3.2.11 IMP SET-DIRECTIVES

Durch diese Direktive können Direktivenwerte auf ihren Defaultwert zurückgesetzt werden.

Format

```
>>IMP SET-DIRECTIVES {ALL | IMPLEMENTOR-DEFINED} TO DEFAULT
```

Syntaxregel

1. Die Direktive darf nur vor einer Übersetzungseinheit angegeben werden.

Allgemeine Regeln

1. Die Direktive wirkt in der Übersetzungsphase.
2. Bei Angabe von ALL werden alle Direktiven, für die ein Defaultwert existiert, auf diesen zurückgesetzt.
3. Bei Angabe von IMPLEMENTOR-DEFINED werden alle IMP-Direktiven, für die ein Defaultwert existiert, auf diesen zurückgesetzt.
4. Der Defaultwert von Direktiven, die einer Compiler-Option entsprechen, ist der Wert, der beim Aufruf des Compilers für diese Option angegeben wurde (ON/OFF statt YES/NO).

3.2.12 LISTING-Direktive

Die LISTING-Direktive steuert, ob Quelltext aufgelistet werden soll oder nicht.

Format

```
>>LISTING {ON | OFF}
```

Allgemeine Regeln

1. Erzeugt der Compiler kein Quelltext-Listing (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]), wird die LISTING-Direktive ignoriert, andernfalls gelten die folgenden Regeln.
2. Die LISTING-Direktive wird nach der COPY- und REPLACE-Anweisung ausgeführt.
3. Die Angabe >>LISTING ON/OFF wird immer protokolliert, auch wenn das Listing selbst durch die LISTING-Direktive unterdrückt wird.
4. Ist OFF angegeben, so wird, ausgenommen eine weitere LISTING-Direktive mit dem Wert OFF, Quelltext solange nicht aufgelistet, bis eine LISTING-Direktive mit dem Wert ON ausgeführt wird.
5. Ist ON angegeben oder impliziert, so wird Quelltext solange aufgelistet, bis entweder eine LISTING-Direktive mit dem Wert OFF ausgeführt wird oder das Ende der Übersetzungsgruppe erreicht ist.

3.2.13 PAGE-Direktive

Die PAGE-Direktive bewirkt einen Seitenvorschub und unterstützt damit die Darstellung des Source-Listings.

Format

```
>>PAGE [kommentar]
```

Syntaxregeln

1. kommentar darf beliebige Zeichen aus dem gesamten Zeichensatz der Datenverarbeitungsanlage enthalten.
2. kommentar wird nicht syntaktisch geprüft.

Allgemeine Regeln

1. kommentar dient nur zu Dokumentationszwecken.
2. Wird ein Source-Listing erstellt, so bewirkt die PAGE-Direktive einen Seitenvorschub, wird kein Listing erzeugt, wirkt sie wie eine Leerzeile.

3.2.14 SOURCE FORMAT-Direktive

Die SOURCE FORMAT-Direktive zeigt das Referenzformat des folgenden Quell- oder Bibliothekstextes an.

Format

```
>>SOURCE [FORMAT IS] {FREE | FIXED}
```

Allgemeine Regeln

1. Ist FIXED angegeben, so wird der nachfolgende Quelltext oder Bilbliothekstext als Fixed-Format behandelt. Ist FREE angegeben so wird er als Free-Format behandelt.
2. Das voreingestellte Referenzformat einer Übersetzungsgruppe ist das Fixed-Format.
3. Für einzulesenden Bilbliothekstext gilt das für die COPY-Anweisung eingestellte Referenzformat solange, bis ggf. im Bibliothekstext eine SOURCE FORMAT-Direktive angegeben ist.
4. SOURCE FORMAT-Direktiven in einem Bibliothekstext gelten solange, bis eine andere SOURCE FORMAT-Direktive angegeben ist oder das Ende des Bibliothekstextes erreicht wurde. Ist die Verarbeitung des Bibliothekstextes beendet, gilt wieder das Referenzformat, das für die auslösende COPY-Anweisung galt.

> **i** Die SOURCE FORMAT-Direktive sollte in neu erstellten COPY-Elementen verwendet werden, damit diese COPY-Elemente auch künftig unabhängig vom Format des umgebenden Quellprogramms verwendet werden können.

3.2.15 TURN-Direktive

Die TURN-Direktive wird verwendet, um die Überprüfung von Ausnahmesituationen einoder auszuschalten.

Format

```
>>TURN {ausnahmesituationsname}... CHECKING {ON | OFF}
```

Syntaxregeln

1. ausnahmesituationsname muss einer der in der Tabelle 45 im Abschnitt "Ausnahmesituationen und Ausnahmezustände" aufgeführtenNamen sein.
2. Kein ausnahmesituationsname darf mehr als einmal in einer TURN-Direktive angegeben werden.

Allgemeine Regeln

1. Ist die TURN-Direktive innerhalb einer Anweisung angegeben, so wirkt sie erst für die nächste Anweisung.

> **i** Für WHEN-Angaben in EVALUATE- bzw. SEARCH-Anweisungen gilt dieselbe TURN-Angabe, wie bei der EVALUATE- bzw. SEARCH-Anweisung selbst.

2. Ist für einen ausnahmesituationsnamen keine TURN-Direktive angegeben, so wird für diesen Namen >>TURN ausnahmesituationsname CHECKING OFF angenommen.
3. Falls ON angegeben oder angenommen ist, so wird die Überprüfung der durch ausnahmesituationsname bezeichneten Ausnahmesituation für den nachfolgenden Quelltext eingeschaltet. Die Überprüfung bleibt eingeschaltet, bis sie durch eine weitere TURN-Direktive für ausnahemsituationsname mit der Angabe OFF wieder ausgeschaltet wird.
4. Falls OFF angegeben ist, so wird die Überprüfung der durch ausnahmesituationsname bezeichneten Ausnahmesituation für den nachfolgenden Quelltext ausgeschaltet. Die Überprüfung bleibt ausgeschaltet, bis sie durch eine weitere TURN-Direktive für ausnahmesituationsname mit der Angabe ON wieder eingeschaltet wird.

4 Struktur einer COBOL-Übersetzungsgruppe

In diesem Kapitel werden folgende Themen behandelt:

4.1 Allgemeine Beschreibung

Eine COBOL-Übersetzungsgruppe kann eine oder mehrere Übersetzungseinheiten enthalten.
Mit Ausnahme der Anweisungen für Quelltextmanipulation sowie der END-Einträge werden alle Anweisungen, Einträge, Paragrafen und Kapitel einer COBOL-Übersetzungseinheit in vier Programmteile geschrieben, die wie folgt angeordnet sind:

1. IDENTIFICATION DIVISION
2. ENVIRONMENT DIVISION
3. DATA DIVISION
4. PROCEDURE DIVISION

Das Ende einer COBOL-Übersetzungseinheit wird entweder bezeichnet durch den entsprechenden END-Eintrag oder durch das Fehlen weiterer Programmzeilen.

Der Beginn eines Programmteils wird bezeichnet durch die entsprechende ProgrammteilÜberschrift bzw. den entsprechenden ID-Namen oder FACTORY bzw. OBJECT. Das Ende eines Programmteils ist gekennzeichnet

- durch eine Programmteil-Überschrift bzw. einen ID-Namen oder FACTORY bzw. OBJECT eines im Programm nachfolgenden Programmteils oder
- durch einen END-Eintrag oder
- durch das Fehlen weiterer Programmzeilen.

Die Anzahl der Zeilen eines COBOL-Gesamtprogramms ist unbeschränkt; eine eindeutige Nummerierung der Programmzeilen durch den Compiler ist aber nur bis zum Wert 65536 gegeben.

4.2 COBOL-Übersetzungsgruppe

Die nachfolgenden Formate zeigen die allgemeine Struktur einer COBOL-Übersetzungsgruppe und die Struktur der darin enthaltenen Übersetzungseinheiten. Eine Übersetzungseinheit ist separat übersetzbar.

Format der Übersetzungsgruppe

```
{Übersetzungseinheit}...
```

Compilation unit:

```
{  Programm-Prototyp
 | Programm-Definition
 | Klassen-Definition
 | Interface-Definition
}
```

Programm-Prototyp:

```
[IDENTIFICATION DIVISION.]

PROGRAM-ID. {programmprototypname-1} IS PROTOTYPE.

[environment-division]

[data-division]

[procedure-division]

END PROGRAM programmprototypname-1.
```

Program definition; nested program definition:

```
[IDENTIFICATION DIVISION.]

PROGRAM-ID. programmname-1 [IS {| COMMON | {INITIAL |  RECURSIVE } |}  PROGRAM ].

[environment-division]

[data-division]

[procedure-division [nested program-definition]...]

[END PROGRAM programmname-1.]
```

Klassen-Definition:

```
[IDENTIFICATION DIVISION.]

CLASS-ID. klassenname-1 [IS FINAL]
   [INHERITS  FROM {klassenname-2}...]
   [USING {parametername-1}...].

[environment-division]
```

```
[ [IDENTIFICATION DIVISION.]
  FACTORY.
  [environment-division]
  [data-division]
  [object-oriented procedure-division]
  END FACTORY.
]

[ [IDENTIFICATION DIVISION.]
  OBJECT.
  [environment-division]
  [data-division]
  [object-oriented procedure-division]
  END OBJECT.
]

END CLASS klassenname-1.
```

Interface-Definition:

```
[IDENTIFICATION DIVISION.]

INTERFACE-ID. interfacename-1
   [INHERITS FROM {interfacename-2}...]
   [USING {parametername-1}...].

[environment-division]

[object-oriented procedure-division]

END INTERFACE interfacename-1.
```

Objektorientierte Procedure Division

```
PROCEDURE DIVISION.
[{methoden-definition}...]
```

Methoden-Definition:

```
[IDENTIFICATION DIVISION.]

METHOD-ID. methodenname-1 [OVERRIDE] [IS FINAL].

[environment-division]

[data-division]

[procedure-division]

END METHOD methodenname-1.
```

Syntaxregeln

1. Innerhalb einer Übersetzungsgruppe müssen die Programm-Prototyp-Definitionen vor allen anderen Übersetzungseinheiten stehen.

2. Enthält eine Übersetzungsgruppe eine Programm-Definition und eine Programm-Prototyp-Definition mit dem selben Namen, müssen die Schnittstellen dieser zwei Übersetzungseinheiten identisch sein.
3. Die Data Division einer Methode innerhalb einer Interface-Definition darf nur dieLinkage Section enthalten.
4. Die Procedure Division einer Methode innerhalb einer Interface-Definition darf nur eine PROCEDURE DIVISION-Überschrift enthalten.

Allgemeine Regeln

1. In der Übersetzungseinheit Programm-Prototyp werden in der Data Division alle Sections, bis auf die Linkage Section und in der Procedure Division alles bis auf die PROCEDURE DIVISION-Überschrift, ignoriert. Die ignorierten Teile müssen syntaktisch richtiges COBOL sein.
2. Der Anfang einer Übersetzungseinheit wird durch die entsprechende Identification Division bezeichnet. Das Ende wird durch eines der folgenden bezeichnet:
   - den END-Eintrag
   - das Ende des ganzen Quellcodes.
3. Beendet ein END-Eintrag eine einzelne Übersetzungseinheit, so folgt entweder keine neue Programmierzeile oder die Identification Division der nächsten einzelnen Übersetzungseinheit (Folge von Übersetzungseinheiten).
4. Die Zuordnung von Zeilen zwischen Übersetzungseinheiten zu vorhergehenden bzw. nachfolgenden Übersetzungseinheit ist undefiniert.

4.3 END-Einträge

Funktion

END-Einträge bezeichnen das Ende einer benannten Quelleinheit (source unit), wobei eine Quelleinheit eine Anweisungsfolge ist, die mit einer Identification Division beginnt und mit einem END-Eintrag schließt.

Format

```
END  {  PROGRAM programmprototypname-1
      | PROGRAM programmname-1
      | CLASS klassenname-1
      | FACTORY
      | OBJECT
      | METHOD methodenname-1
      | INTERFACE interfacename-1
     }.
```

Syntaxregeln

1. Ein END-Eintrag muss für jede Quelleinheit (source unit) vorhanden sein, die eine weitere Quelleinheit enthält, in einer anderen enthalten ist oder einer anderen vorausgeht.
2. Steht zwischen einem PROGRAM-ID-Paragrafen für programmname-1 und dem zugehörigen END-Eintrag ein weiterer PROGRAM-ID-Paragraf für programmname-2, dann muss der END-Eintrag für programmname-2 vor demjenigen für programmname-1 kommen.
3. progammname-1 muss mit dem Programmnamen übereinstimmen, der im PROGRAM-ID-Paragrafen des zu beendenden Programms vereinbart wurde.
4. klassenname-1 muss mit dem Klassennamen übereinstimmen, der im zugehörigen CLASS-ID-Paragrafen vereinbart wurde.
5. methodenname-1 muss mit dem Methodennamen übereinstimmen, der im zugehörigen METHOD-ID-Paragrafen vereinbart wurde.
6. interfacename-1 muss mit dem Schnittstellennamen übereinstimmen, der im zugehörigen INTERFACE-ID-Paragrafen vereinbart wurde.
7. programmprototypname-1 muss mit einem Programmprototypnamen übereinstimmen, der im zugehörigen PROGRAM-ID-Paragrafen vereinbart wurde.

5 IDENTIFICATION DIVISION

In diesem Kapitel werden folgende Themen behandelt:

5.1 Allgemeine Beschreibung

Mit der IDENTIFICATION DIVISION beginnt eine Quelleinheit. In der dazugehörigen Paragrafen-Überschrift wird diese Quelleinheit spezifiziert.
Es kann sich um ein Programm, eine Klasse, ein Fabrikobjekt (Factory-Objekt), ein Objekt, eine Methode oder eine Schnittstelle handeln.

5.2 Allgemeines Format

```
{ [IDENTIFICATION DIVISION. | ID DIVISION.] }

{  program-id paragraph
 | class-id paragraph
 | factory paragraph
 | object paragraph
 | method-id paragraph
 | interface-id paragraph
}
```

5.3 Paragrafen

In diesem Kapitel werden folgende Themen behandelt:

- PROGRAM-ID-Paragraf
- CLASS-ID-Paragraf
- FACTORY-Paragraf
- OBJECT-Paragraf
- METHOD-ID-Paragraf
- INTERFACE-ID-Paragraf

5.3.1 PROGRAM-ID-Paragraf

Function

The PROGRAM-ID paragraph contains the name by which a program or program prototype is identified. The program can be assigned the appropriate attributes with the INITIAL, COMMON or RECURSIVE clause.

Format 1

```
PROGRAM-ID. programmname [ IS {|COMMON | {INITIAL | RECURSIVE } |}  PROGRAM].
```

Format 2

```
PROGRAM-ID. programmprototypname-1 IS PROTOTYPE.
```

Syntaxregeln

1. programmname muss ein 1-30 Zeichen langer benutzerdefinierter Name sein. Er muss mit einem Buchstaben beginnen.

   Anmerkung:
   Ein Programmname sollte nicht mit dem Buchstaben „I“ beginnen, damit er nicht mit Namen von COBOL-Laufzeitmoduln bzw. Moduln anderer Laufzeitsysteme in Konflikt gerät.
2. Die Programme eines geschachtelten Programms müssen verschiedene Namen tragen.
3. Die COMMON-Klausel darf nur für die inneren Programme eines geschachtelten Programms angegeben werden. Im äußersten Programm darf sie nicht angegeben werden.
4. Die INITIAL-Klausel darf in keinem Programm angegeben werden, das direkt oder indirekt in einem rekursiven Programm enthalten ist.
5. Die RECURSIVE-Klausel darf in keinem Programm angegeben werden, das direkt oder indirekt in einem Programm enthalten ist, das die INITIAL-Klausel enthält.

Allgemeine Regeln für Format 1

1. Die INITIAL-Klausel bewirkt, dass sich das Programm bei jedem Aufruf im Initialzustand befindet (siehe Abschnitt „Initialzustand bei der Programmkommunikation").
2. Die COMMON-Klausel bestimmt, dass das Programm das Attribut COMMON besitzt. Ein derartiges Programm kann nicht nur von dem direkt übergeordneten Programm aufgerufen werden, sondern auch von seinen „Geschwisterprogrammen“ und deren „Abkömmlingen“.
3. Die RECURSIVE-Klausel gibt an, dass das Programm und alle Programme, die in diesem enthalten sind, rekursiv sind. Das Programm kann gerufen werden während es bereits aktiv ist und es kann sich selbst rufen. Ist die RECURSIVE-Klausel in einem Programm angegeben oder implizit für dieses Programm gültig, darf das Programm gerufen werden, während es schon aktiv ist.

Allgemeine Regel für Format 2

1. programmprototypname-1 spezifiziert den Programm-Prototyp.

Zusatzregeln, abhängig vom Modulformat

Wird das Format *OMF generiert (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]), dann gilt für programmname bzw. programmprototypname-1:

1. Das achte Zeichen darf kein Bindestrich sein.
2. Vom Betriebssystem werden nur die ersten 8 Zeichen des Namens zur Modulidentifizierung verwendet. Deshalb sollten diese Zeichen eindeutig sein für jeden Namen in einer bestimmten Modul-/Programmbibliothek.

3. Bei Erzeugung von shared code werden nur die ersten 7 Zeichen des Namens verwendet.

Beispiel 5-1

für die Wirkung der INITIAL-Klausel

Aufrufendes Programm:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
...
DATA DIVISION.
WORKING-STORAGE SECTION.
...
PROCEDURE DIVISION.
...
    CALL "UPROG1" USING...  (1)
...
...
    CALL "UPROG1" USING...  (2)
```

Aufgerufenes Programm:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPROG1 IS INITIAL PROGRAM.
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 USER-DATE PIC X(8) VALUE SPACE.
...
PROCEDURE DIVISION USING ...
...
    MOVE "26.10.49" TO USER-DATE.
...
    EXIT PROGRAM.
```

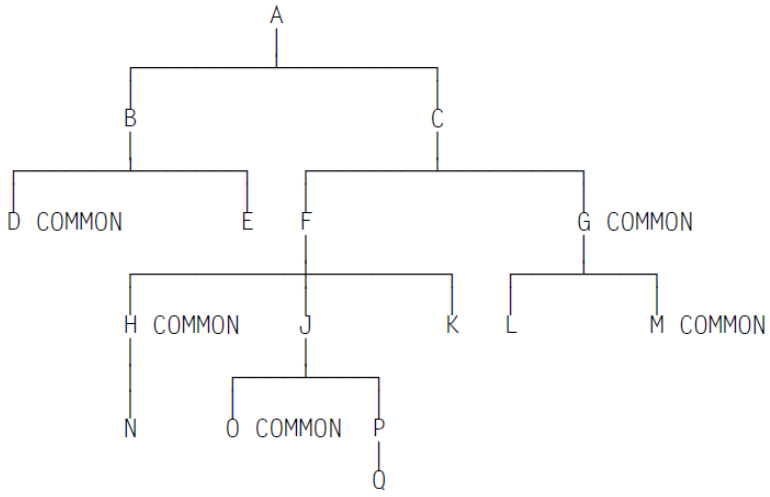
(1) ist der erste Aufruf von UPROG1 durch HAUPT.

(2) ist der zweite Aufruf von UPROG1 durch HAUPT; wie beim ersten Aufruf ist das Unterprogramm UPROG1 hier im Initialzustand. So ist z.B. der Inhalt von BENUTZER-DATUM wieder SPACE, auch wenn BENUTZER-DATUM beim ersten Aufruf verändert wurde.

Beispiel 5-2

für die Wirkung der COMMON-Klausel

Struktur eines geschachtelten Programms A:



Aufrufmöglichkeiten innerhalb des oben skizzierten geschachtelten Programms:

aufgerufenes Programm	aufrufendes Programm															
	A	B	C	D	E	F	G	H	J	K	L	M	N	O	P	Q
A																
B	x															
C	x															
D		x			x											
E		x														
F			x													
G			x			x		x	x	x			x	x	x	x
H						x			x	x				x	x	x
J						x										
K						x										
L							x									
M							x				x					
N								x								
O									x						x	x
P									x							
Q															x	

x = erlaubter Aufruf

5.3.2 CLASS-ID-Paragraf

Funktion

Der CLASS-ID-Paragraf zeigt an, dass diese Identification Division eine Klassen-Definition einleitet und spezifiziert den Namen der Klasse und ihre Attribute.

Format

```
CLASS-ID. klassenname-1 [IS FINAL]
   [INHERITS FROM {klassenname-2}...]
   [USING {parametername-1}...].
```

Syntax-Regeln

1. klassenname-1 darf nicht BASE sein.
2. klassenname-2 ist der Name einer Klasse, der im REPOSITORY-Paragrafen definiert sein muss.
3. klassenname-2 muss ungleich klassenname-1 sein.
4. klassenname-2 darf weder direkt noch indirekt von einer Klasse erben, die eine Expansion von klassenname-1 ist.
5. klassenname-2 darf nicht mit der FINAL-Klausel definiert sein.
6. Falls zwei oder mehr unterschiedliche Methoden mit demselben Namen geerbt werden, darf keine von ihnen mit der FINAL-Klausel spezifiziert sein. Falls dieselbe Methode von einer übergeordneten Klasse durch zwei oder mehr übergeordnete Zwischenklassen geerbt wird, darf sie mit der FINAL-Klausel spezifiziert sein.
7. Der gleiche klassenname-2 darf nicht mehrfach in einer INHERITS-Klausel vorkommen.
8. Gibt es eine Methode mit gleichem Namen in mehr als einer der geerbten Klassen und haben diese Methoden verschiedene Schnittstellen, dann muss in dieser Klasse eine Methode mit diesem Namen definiert werden, deren Schnittstelle Syntaxregel 5 von Method-Id erfüllen muss.
9. parametername-1 ist der Name einer Klasse oder eines Interface, der im REPOSITORY-Paragrafen definiert sein muss.
10. Der gleiche parametername-1 darf nicht mehrfach in einer USING-Klausel vorkommen.
11. parametername-1 muss ungleich klassenname-1 sein.

Allgemeine Regeln

1. klassenname-1 benennt die Klasse, die durch diese Klassendefinition vereinbart ist.
2. Die INHERITS-Klausel spezifiziert die Namen der Klassen, die von klassenname-1 geerbt werden.
3. Falls die FINAL-Klausel spezifiziert wird, kann die Klasse nicht die übergeordnete Klasse für eine andere Klasse sein.
4. Falls dieselbe Klasse mehr als einmal geerbt wird, dann wird nur eine Kopie der Daten für diese Klasse zu klassenname-1 hinzugefügt.
5. Die USING-Klausel spezifiziert diese Klasse als parametrisierte Klasse und führt die Namen der Parameter auf.

5.3.3 FACTORY-Paragraf

Funktion

Der FACTORY-Paragraf zeigt an, dass durch die Identification Division eine Fabrik-Definition (factory definition) eingeleitet wird.

Format

```
FACTORY. [IMPLEMENTS {interfacename-1}... .]
```

Syntaxregeln

1. interfacename-1 muss der Name einer Schnittstellendefinition sein, die im REPOSITORY-Paragrafen aufgeführt ist.
2. Die Schnittstelle des Fabrikobjektes muss zu allen Schnittstellen interfacename-1 konform sein.

Allgemeine Regeln

Ist die IMPLEMENTS-Klausel angegeben, überprüft der Compiler bereits bei der Übersetzung einer Klasse, ob die Schnittstelle ihres Fabrikobjektes zu interfacename-1 konform ist. Fehlt die IMPLEMENTS-Klausel, so kann diese Überprüfung erst bei der Verwendung der Objektreferenz für das Fabrikobjekt im Zusammenhang mit interfacename-1 erfolgen.

5.3.4 OBJECT-Paragraf

Funktion

Der OBJECT-Paragraf zeigt an, dass durch die Identification Division eine Objekt-Definition eingeleitet wird.

Format

```
OBJECT. [IMPLEMENTS {interfacename-1}... .]
```

Syntaxregeln

1. interfacename-1 muss der Name einer Schnittstellendefinition sein, die im REPOSITORY Paragrafen aufgeführt ist.
2. Die Schnittstelle der Objekte muss zu allen Schnittstellen interfacename-1 konform sein.

Allgemeine Regeln

Ist die IMPLEMENTS-Klausel angegeben, überprüft der Compiler bereits bei der Übersetzung einer Klasse, ob die Schnittstelle ihrer Objekte zu interfacename-1 konform ist. Fehlt die IMPLEMENTS-Klausel, so kann diese Überprüfung erst bei der Verwendung der Objektreferenz für ein Objekt im Zusammenhang mit interfacename-1 erfolgen.

5.3.5 METHOD-ID-Paragraf

Funktion

Der METHOD-ID-Paragraf zeigt an, dass durch diese Identification Division eine Methoden-Definition eingeleitet wird und spezifiziert den Namen der Methode und ihre Attribute.

Format

```
METHOD-ID. methodenname-1 [OVERRIDE] [IS FINAL].
```

Syntax-Regeln

1. Wenn OVERRIDE angegeben ist, muss eine der geerbten Klassen eine Methode mit dem Namen methodenname-1 enthalten. Diese Methode in der übergeordneten Klasse darf nicht mit der FINAL-Klausel definiert sein.
2. Ist OVERRIDE nicht angegeben, darf keine der geerbten Klassen eine Methode mit dem Namen methodenname-1 enthalten.
3. Die OVERRIDE-Klausel darf nicht in einer Interface-Methode angegeben werden.
4. Die FINAL-Klausel darf nicht in einer Interface-Definition angeben werden.
5. Ist methodenname-1 identisch mit Methoden-Namen aus geerbten Klassen, dann muss die Schnittstelle von methodenname-1 zu der Schnittstelle der geerbten Methode(n) konform sein.
6. Die Namen der Methoden in einer Klasse bzw. Interface müssen eindeutig sein.

Allgemeine Regeln

1. methodenname-1 ist der Name der Methode, die durch die Methoden-Definition spezifiziert ist.
2. Die Angabe OVERRIDE zeigt an, dass die Methode methodenname-1 eine gleichnamige geerbte Methode ersetzt.
3. Die FINAL-Klausel zeigt an, dass diese Methode in keiner erbenden Klasse mittels OVERRIDE geändert werden darf.
4. methodenname-1 kann im Aufruf einer Methode für ein Objekt der Klasse , in der diese Methode definiert ist, verwendet werden.
5. Falls ein benutzerdefiniertes Wort in der DATA DIVISION einer Methoden-Definition und in der DATA DIVISION der umgebenden Fabrik-Definition (Factory-definition) oder Objekt-Definition vereinbart ist, bezieht sich die Anwendung dieses Wortes auf die Vereinbarung in der Methode. Die Vereinbarung in der umgebenden Fabrik-Objekt-Definition oder der Objekt-Definition ist in diesem Fall für diese Methode nicht zugänglich.

5.3.6 INTERFACE-ID-Paragraf

Funktion

Der INTERFACE-ID-Paragraf zeigt an, dass diese Identification Division eine Schnittstellen-Definition einleitet und spezifiziert den Interfacenamen und seine Attribute.

Format

```
INTERFACE-ID. interfacename-1
   [INHERITS FROM {interfacename-2}...]
   [USING {parametername-1}...].
```

Syntax-Regeln

1. interfacename-1 darf weder BASEFACTORYINTERFACE noch BASEINTERFACE sein.
2. interfacename-2 ist der Name einer Schnittstelle, die im REPOSITORY-Paragrafen definiert sein muss.
3. interfacename-2 darf nicht direkt oder indirekt von interfacename-1 erben. interfacename-2 darf weder direkt noch indirekt von einem Interface erben, das eine Expansion von interfacename-1 ist.
4. Falls ein Methoden-Name von mehr als einem Interface geerbt wird, dann muss jede dieser Methoden die gleiche Schnittstelle haben.
5. parametername-1 ist der Name einer Klasse oder eines Interface, der im REPOSITORY-Paragrafen definiert sein muss.
6. Der gleiche parametername-1 darf nicht mehrfach in einer USING-Klausel vorkommen.
7. parametername-1 muss ungleich interfacename-1 sein.

Allgemeine Regeln

1. interfacename-1 benennt die Schnittstelle, die durch diese Schnittstellen-Definition vereinbart ist.
2. Die Angabe INHERITS gibt die Namen der Schnittstellen an, die von interfacename-1 gemäß den Regeln für Schnittstellen-Vererbung geerbt werden.
3. Die USING-Klausel spezifiziert dieses Interface als parametrisiertes Interface und führt die Namen der Parameter auf.

6 ENVIRONMENT DIVISION

In diesem Kapitel werden folgende Themen behandelt:

6.1 Allgemeine Beschreibung

In der ENVIRONMENT DIVISION werden diejenigen Aspekte eines Datenverarbeitungsproblems beschrieben, die von den physikalischen Gegebenheiten einer bestimmten Datenverarbeitungsanlage abhängen. In diesem Teil können Angaben über die Anlagenausstattung der Datenverarbeitungsanlage gemacht werden, auf der das Programm übersetzt wird bzw. ablaufen soll. Außerdem können Angaben bezüglich der Ein-/AusgabeSteuerung, spezieller Maschinengegebenheiten und Steuerungsverfahren gemacht werden.

Die ENVIRONMENT DIVISION muss nicht in einer COBOL-Übersetzungseinheit enthalten sein.

Sie besteht aus zwei optionalen Kapiteln:

1. CONFIGURATION SECTION
2. INPUT-OUTPUT SECTION.

Die CONFIGURATION SECTION behandelt die Eigenschaften der Übersetzungsanlage und der Programmausführungsanlage.

Dieses Kapitel ist in folgende Paragrafen unterteilt:

1. den SOURCE-COMPUTER-Paragrafen, der die Anlagenausstattung der Datenverarbeitungsanlage beschreibt, auf der die Übersetzungseinheit übersetzt werden soll;
2. den OBJECT-COMPUTER-Paragrafen, der die Anlagenausstattung der Datenverarbeitungsanlage beschreibt, auf der das Zielprogramm ablaufen soll;
3. den SPECIAL-NAMES-Paragrafen, der u.a. die vom Compiler verwendeten Herstellernamen zu den in der Übersetzungseinheit verwendeten Merknamen in Beziehung setzt.
4. den REPOSITORY-Paragrafen, der die in Programmen verwendeten Schnittstellen-, Klassen- und Programmprototypnamen spezifiziert.

Die INPUT-OUTPUT SECTION behandelt die Angaben, die notwendig sind, um die Übertragung von Daten zwischen externen Geräten und dem Zielprogramm zu steuern.

Dieses Kapitel ist in zwei Paragrafen unterteilt:

1. den FILE-CONTROL-Paragrafen, der die Dateien aufführt und externen Geräten zuordnet, und
2. den I-O-CONTROL-Paragrafen, der spezielle Steuerungsverfahren angibt, die im Zielprogramm verwendet werden.

Allgemeines Format

```
ENVIRONMENT DIVISION.


[configuration-section]
[input-output-section]
```

6.2 CONFIGURATION SECTION

Funktion

Die Configuration Section steht in der Environment Division einer Übersetzungseinheit. Sie ermöglicht folgende Angaben:

- Bezeichnung der Rechenanlage, auf der das Programm übersetzt bzw. ausgeführt werden soll
- Vereinbarung eines Währungszeichens
- Austausch von Komma und Dezimalpunkt
- Festlegung symbolischer Namen für Zeichen
- Verknüpfung von Herstellernamen mit vom Benutzer angegebenen Merknamen
- Verknüpfung von Alphabetnamen mit Zeichensätzen oder Sortierfolgen
- Verknüpfung von Klassennamen mit vom Benutzer definierten Zeichenmengen.

Format

```
CONFIGURATION SECTION.

[SOURCE-COMPUTER. [kompilierungsanlageneintrag [testhilfemodus].]]
[OBJECT-COMPUTER. [programmausführungsanlageneintrag.]]
[SPECIAL-NAMES. [sondernameneintrag.]]
[REPOSITORY. [spezifizierer]]
```

Syntaxregeln

1. Die Configuration Section und die zugehörigen Paragrafen sind optional.
2. Bei Angabe dieser Paragrafen muss die vorgegebene Reihenfolge eingehalten werden.
3. SOURCE-COMPUTER- , OBJECT-COMPUTER-, SPECIAL-NAMES und REPOSITORY-Paragrafen dürfen nicht in einer FACTORY- oder OBJECT-Definition angegeben werden.
4. CONFIGURATION SECTION darf nicht in einem geschachtelten Programm angegeben werden.
5. CONFIGURATION SECTION darf nicht in einer Methode angegeben werden.

6.2.1 SOURCE-COMPUTER-Paragraf

Funktion

Der SOURCE-COMPUTER-Paragraf bezeichnet die Datenverarbeitungsanlage, auf der die Übersetzungseinheit übersetzt werden soll; außerdem kann eine Angabe über Testhilfen gemacht werden.

Format

```
SOURCE-COMPUTER. [rechenanlagenbezeichnung [WITH DEBUGGING MODE]. ]
```

Syntaxregel

1. rechenanlagenbezeichnung muss ein vom Benutzer definiertes COBOL-Wort sein.

Allgemeine Regeln

1. Alle Klauseln dieses Paragrafen beziehen sich auf das Programm, in dem sie explizit oder implizit angegeben werden.
2. Ist der Paragraf nicht oder ohne die Rechenanlagenbezeichnung angegeben, gilt der Rechner, auf dem die Übersetzungseinheit übersetzt wird, als Übersetzungsrechner.
3. Ist die WITH-DEBUGGING-MODE-Klausel angegeben, werden alle Testhilfezeilen gemäß den in Abschnitt „Testhilfen" beschriebenen Regeln übersetzt.
4. Ist die WITH-DEBUGGING-MODE-Klausel nicht angegeben, werden alle Testhilfezeilen als Kommentar behandelt.

6.2.2 OBJECT-COMPUTER-Paragraf

Funktion

Der OBJECT-COMPUTER-Paragraf beschreibt die Datenverarbeitungsanlage, auf der das Programm ausgeführt werden soll.

Format

```
OBJECT-COMPUTER. [rechenanlagenbezeichnung
   [MEMORY SIZE ganzzahl {WORDS | CHARACTERS | MODULES}]
[PROGRAM COLLATING SEQUENCE IS alphabetname].]
```

Syntaxregeln

1. rechenanlagenbezeichnung sowie die MEMORY SIZE-Klausel dienen nur zur Dokumentation und werden als Kommentar betrachtet.
2. rechenanlagenbezeichnung muss ein vom Benutzer definiertes COBOL-Wort sein.
3. Ist die PROGRAM COLLATING SEQUENCE-Klausel angegeben, wird die durch alphabetname (siehe "SPECIAL-NAMES-Paragraf") bezeichnete Sortierfolge benutzt, um den Wahrheitswert aller alphanumerischen Vergleiche zu bestimmen:
   - Explizit angegeben in Vergleichsbedingungen
   - Explizit angegeben in Bedingungsnamen-Bedingungen
   - Implizit angegeben durch eine CONTROL-Klausel in einer Listenerklärung (siehe Abschnitt "CONTROL-Klausel").
4. Ist die PROGRAM COLLATING SEQUENCE-Klausel nicht angegeben, werden die maschineneigenen Sortierfolgen verwendet.
5. Ist bei der MERGE- bzw. SORT-Anweisung keine COLLATING SEQUENCE-Angabe vorhanden, wird für alphanumerische Sortierschlüssel die PROGRAM COLLATING SEQUENCE-Klausel als Sortierfolge verwendet (siehe Abschnitt "MERGE-Anweisung" , und Abschnitt "SORT-Anweisung").

Beispiele für PROGRAM COLLATING SEQUENCE- und ALPHABET-Klausel siehe "SPECIAL-NAMES-Paragraf" ("SPECIAL-NAMES-Paragraf ").

6.2.3 SPECIAL-NAMES-Paragraf

Function

Der SPECIAL-NAMES-Paragraf ermöglicht

1. die Verknüpfung von Systemnamen mit vom Benutzer angegebenen Merknamen,
2. die Verknüpfung von Alphabetnamen mit Zeichensätzen und/oder Sortierfolgen,
3. die Festlegung symbolischer Namen für Zeichen,
4. die Verknüpfung von Klassennamen mit Zeichenmengen,
5. die Angabe eines Zeichens, das als Währungszeichen in Maskenzeichenfolgen verwendetwerden soll,
6. den Austausch der Bedeutung von Komma und Dezimalpunkt in Maskenzeichenfolgen sowie in numerischen Literalen.

Format

```
SPECIAL-NAMES.
[ [herstellername { IS merkname-1 [ON STATUS IS bedingungsname-1
                                   [OFF STATUS IS bedingungsname-2]]
                  | IS merkname-2 [OFF STATUS IS bedingungsname-2
                                   [ON STATUS IS bedingungsname-1]]
                  | ON STATUS IS bedingungsname-1[OFF STATUS IS bedingungsname-
                  2]
                  | OFF STATUS IS bedingungsname-2[ON STATUS IS bedingungsname-
                  1]
                  }
  ]...
  [[ARGUMENT-NUMBER IS merkname-3]
  [ARGUMENT-VALUE IS merkname-4]
  [ENVIRONMENT-NAME IS merkname-5]
  [ENVIRONMENT-VALUE IS merkname-6]
  ]
  [ALPHABET
  alphabetname-1 IS          { STANDARD-1
                             | STANDARD-2
                             | NATIVE
                             | EBCDIC
                             | {literal-1 [ {THROUGH | THRU} literal-2 {ALSO
                             literal-3}... ]}...
                             }
  ]...
  [SYMBOLIC CHARACTERS { {symbolisches-zeichen-1}... {IS | ARE} {ganzzah-1}...
  }...
  [IN alphabetname-2]
  ]...
  [CLASS klassenname IS { literal-4 [{THROUGH | THRU} literal-5] }... ]...
  [CURRENCY SIGN IS literal-6]
  [DECIMAL-POINT IS COMMA].
]
```

Syntaxregeln

1. In einer Klassen-Definition darf im SPECIAL-NAMES Paragrafen nur die CURRENCY SIGN-Klausel angegeben werden.
2. In einer Schnittstellen-Definition dürfen im SPECIAL-NAMES Paragrafen nur die ALPHABET-, CURRENCY SIGN- und DECIMAL-POINT-Klausel verwendet werden.

3. Die Wörter THROUGH und THRU sind gleichbedeutend.
4. Für literal-1 bis literal-5 gilt:
   - Sind die Literale numerisch, müssen sie vorzeichenlose Ganzzahlen sein und einen Wert von 1 bis 256 haben.
   - Sind die Literale alphanumerisch und mit der THROUGH-, THRU- oder ALSO-Angabe verbunden, muss jedes Literal 1 Zeichen lang sein.
   - Die Literale dürfen keine figurative Konstante der Form symbolisches-zeichen sein.

Allgemeine Regel

1. Die einzelnen Klauseln des SPECIAL-NAMES-Paragrafen müssen, wenn sie verwendet werden, in der im Format aufgeführten Reihenfolge angegeben werden.

Im Folgenden sind die einzelnen Klauseln des SPECIAL-NAMES-Paragrafen beschrieben.

6.2.4 herstellername

Syntaxregeln

1. herstellername ist ein Systemname und muss ein Name aus der linken Spalte der folgenden Tabelle sein.

   Die Herstellernamen und ihre Bedeutung:

herstellername	Bedeutung
CONSOLE	System-, Haupt- oder Nebenbedienplatz
TERMINAL	Datensichtstation des Benutzers
SYSIPT	logische Eingabedatei des Systems
PRINTER,PRINTER01-PRINTER99	logische Druckerdatei des Systems
SYSOPT	logische Ausgabedatei des Systems
C01 bis C08	Sprung auf Kanal 1 bis 8
C10 bis C11	Sprung auf Kanal 10 bzw. 11
JV-jobvariablenname	Jobvariable, die den Linknamen einer Jobvariablen beschreibt (siehe unten)
TSW-0 bis TSW-31	Prozessschalter
USW-0 bis USW-31	Benutzerschalter
COMPILER-INFO	Informationen des Compilers
CPU-TIME,PROCESS-INFO,TERMINAL-INFODATE-ISO4	Informationen des Betriebssystems

   Tabelle 8: Herstellernamen und ihre Bedeutung

2. jobvariablenname bezeichnet eine Jobvariable des BS2000. jobvariablenname ist ein maximal 7 Zeichen langes COBOL-Wort; daraus wird der Linkname *jobvariablenname gebildet und für den Zugriff auf die Jobvariable benutzt (siehe Beispiel 6-1).

Allgemeine Regeln

1. Ist herstellername ein Benutzer- oder Prozessschalter, muss damit mindestens ein Bedingungsname verknüpft sein. Der Status der Schalter ist unter „Bedingungsnamen“ beschrieben und kann durch Testen von bedingungsname abgefragt werden (siehe Abschnitt „Schalterzustandsbedingung").

   Der Zustand eines Schalters kann mit einer SET-Anweisung, Format 3, verändert werden (siehe Abschnitt „SET-Anweisung").

2. C01 bis C08, C10 und C11 werden nur noch in dieser Version des COBOL2000-Compilers unterstützt. Falls C01 bis C08, C10 oder C11 als Herstellernamen angegeben sind, darf der zugehörige Merkname nur in einer WRITE-Anweisung mit ADVANCING-Angabe verwendet werden.

Beispiel 6-1

für den Einsatz von Jobvariablen:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. JVTEST.
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    JV-JV1 IS JOB-VAR-1.
    ...
PROCEDURE DIVISION.
    ...
    DISPLAY "xyz" UPON JOB-VAR-1.
```

Vor dem Programmaufruf:

```
/SET-JV-LINK LINK-NAME=*JV1,JV-NAME=JV1TEST
```

6.2.5 ARGUMENT-NUMBER / ARGUMENT-VALUE / ENVIRONMENT-NAME / ENVIRONMENT-VALUE

sind Erweiterungen aus dem X/OPEN Portability Guide. Sie werden nur in Verbindung mit ACCEPT- und DISPLAY-Anweisungen verwendet und sind dort beschrieben.

6.2.6 ALPHABET-Klausel

Syntaxregeln

1. Ist die Literal-Angabe in der ALPHABET-Klausel angegeben, darf ein gegebenes Zeichen für literal-1, literal-2 usw., auf das durch alphabetname in der PROGRAM COLLATING SEQUENCE-Klausel (siehe „OBJECT-COMPUTER-Paragraf“, "OBJECT-COMPUTER-Paragraf ") oder in der COLLATING SEQUENCE-Angabe der SORT- oder MERGE-Anweisung (siehe Kapitel „PROCEDURE DIVISION") Bezug genommen wird, nur einmal verwendet werden (siehe Beispiele 6-10 und 6-11).
2. Die Wörter THROUGH und THRU sind gleichbedeutend.
3. Die Angaben NATIVE und EBCDIC bedeuten im BS2000 dasselbe.

Allgemeine Regeln

1. Durch die ALPHABET-Klausel kann ein Name mit einem bestimmten Zeichensatz und/oder einer alphanumerischen Sortierfolge verknüpft werden. Ist alphabetname in der PROGRAM COLLATING SEQUENCE-Klausel (siehe „OBJECT-COMPUTER-Paragraf“, "OBJECT-COMPUTER-Paragraf ") oder in der COLLATING SEQUENCE-Angabe einer SORT- bzw. MERGE-Anweisung (siehe Kapitel „PROCEDURE DIVISION") enthalten, wird damit eine Sortierfolge festgelegt. Wird auf alphabetname-1 in der
   SYMBOLIC CHARACTERS-Klausel oder in der CODE-SET-Klausel einer Dateierklärung (für sequenziell organisierte Dateien) Bezug genommen, wird damit der Zeichensatz festgelegt.
   a. Ist STANDARD-1 angegeben, dann ist der ausgewählte Zeichensatz oder die Sortierfolge der/die im American National Standard X3.4-1968, Code for Information Interchange (ASCII), festgelegte.
   b. Ist STANDARD-2 angegeben, dann ist der bezeichnete Zeichensatz die „International Reference Version of the ISO 7-Bit Code“, definiert im International Standard 646 „7-Bit Coded Character Set for Information Processing Interchange“. Jedes Zeichen des Standard-Zeichensatzes ist verknüpft mit dem entsprechenden Zeichen des EBCDIC-Zeichensatzes.
   c. Ist NATIVE oder EBCDIC angegeben, wird der Zeichensatz oder die Sortierfolge der Datenverarbeitungsanlage benutzt (EBCDIC).
   d. Sind Literale der ALPHABET-Klausel angegeben, darf alphabetname nicht in der CODE-SET-Klausel angegeben sein (siehe Abschnitt „CODE-SET-Klausel").
      - Handelt es sich um ein numerisches Literal, bezeichnet sein Wert die Ordnungszahl eines Zeichens (beginnend bei 1) innerhalb des EBCDIC-Zeichensatzes. Der Wert darf die Anzahl der Zeichen des EBCDIC-Zeichensatzes (256) nicht überschreiten.
      - Bei einem alphanumerischen Literal bezeichnet das Literal das angegebene Zeichen innerhalb des EBCDIC-Zeichensatzes. Enthält das alphanumerische Literal mehrere Zeichen, werden alle Zeichen im Literal in die beschriebene Sortierfolge in der angegebenen Reihenfolge eingefügt (siehe Beispiel 6-2).
      - Die Reihenfolge, in der die Literale in der ALPHABET-Klausel angegeben sind, beschreibt in aufsteigender Folge die Ordnungszahl der Zeichen innerhalb der spezifizierten Sortierfolge (siehe Beispiel 6-3).
      - Alle Zeichen innerhalb der EBCDIC-Sortierfolge, die nicht explizit mit der Literal-Angabe spezifiziert sind, haben in der Sortierfolge eine höhere Position als jedes einzelne der explizit beschriebenen Zeichen. Die relative Reihenfolge dieser nicht spezifizierten Zeichen unterscheidet sich nicht von der EBCDIC-Sortierfolge.
      - Ist THROUGH/THRU angegeben, nehmen die Zeichen im EBCDIC-Zeichensatz, beginnend mit dem in literal-1 und endend mit dem in literal-2 angegebenen Zeichen, aufeinanderfolgend aufsteigende Positionen in der spezifizierten Sortierfolge ein. Der durch die THROUGH/THRU-Angabe spezifizierte Wertebereich kann Zeichen des EBCDIC-Zeichensatzes entweder in aufsteigender oder in absteigender Reihenfolge enthalten (siehe Beispiel 6-4).

- Ist ALSO angegeben, werden die Zeichen des EBCDIC-Zeichensatzes, angegeben durch literal-1 und literal-3 der selben Position in der spezifizierten Sortierfolge oder in dem Zeichensatz zugewiesen (siehe Beispiel 6-5).
  Wird in einer SYMBOLIC CHARACTERS-Klausel auf alphabetname-1 Bezug genommen, wird nur literal-1 verwendet, um das Zeichen des EBCDIC-Zeichensatzes darzustellen.

2. Das Zeichen, das die höchste Position in der Sortierfolge des Programms hat, ist der figurativen Konstante HIGH-VALUE zugeordnet. Haben mehrere Zeichen die höchste Position in der Sortierfolge des Programms, ist das letzte angegebene Zeichen der figurativen Konstante HIGH-VALUE zugeordnet (siehe Beispiele 6-6 und 6-7).
3. Das Zeichen, das die niedrigste Position in der Sortierfolge des Programms hat, ist der figurativen Konstante LOW-VALUE zugeordnet. Haben mehrere Zeichen die niedrigste Position in der Sortierfolge des Programms, ist das erste angegebene Zeichen der figurativen Konstante LOW-VALUE zugeordnet (siehe Beispiele 6-8 und 6-9).

Beispiel 6-2

```
ALPHABET ALPHATAB IS "AJKCDF".
```

1. Zeichen ist A
2. Zeichen ist J
.
.
6. Zeichen ist F

Beispiel 6-3

```
ALPHABET ALPHATAB IS "A" "C" "D" "Z".
```

1. Zeichen ist A
2. Zeichen ist C
3. Zeichen ist D
4. Zeichen ist Z

Beispiel 6-4

```
ALPHABET ALPHATAB IS "A" THRU "I".
```

1. Zeichen A
2. Zeichen B
3. Zeichen C
.
.
8. Zeichen H
9. Zeichen I

Beispiel 6-5

```
ALPHABET ALPHATAB IS "A" ALSO "B" ALSO "C" ALSO "D".
```

Der niedrigsten Position in der Sortierfolge werden die Zeichen A, B, C und D zugeordnet.

Beispiel 6-6

```
ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 194.
```

Die höchste Position in der Sortierfolge hat das Zeichen, das an der 194. Stelle des EBCDIC-Zeichensatzes steht; dies ist das Zeichen A.
Der figurativen Konstante HIGH-VALUE wird A zugeordnet.

Beispiel 6-7

```
ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 197, "A" ALSO "B" ALSO "C".
```

Den Positionen 1 bis 193 der Sortierfolge werden die Zeichen zugeordnet, die an den Positionen 193 bis 1 des EBCDIC-Zeichensatzes stehen.

Den Positionen 194 bis 253 der Sortierfolge werden die Zeichen zugeordnet, die an den Positionen 255 bis 197 des EBCDIC-Zeichensatzes stehen.

Der Position 254 werden die Zeichen A, B, C zugeordnet, womit alle Zeichen des EBCDIC-Zeichensatzes einer Position der Sortierfolge zugeordnet sind. Der höchsten Position (254) sind die Zeichen A, B, C zugeordnet, wobei C das zuletzt angegebene Zeichen ist; C wird damit der figurativen Konstante HIGH-VALUE zugeordnet.

Beispiel 6-8

```
ALPHABET ALPHATAB IS "0" "1" "2".
```

Das niedrigste Zeichen in der Sortierfolge ist 0, somit wird 0 der figurativen Konstante LOW-VALUE zugeordnet.

Beispiel 6-9

```
ALPHABET ALPHATAB IS "A" ALSO "B" ALSO "C".
```

Der niedrigsten Position in der Sortierfolge sind die Zeichen A, B, C zugeordnet; das zuerst angegebene Zeichen A wird der figurativen Konstante LOW-VALUE zugeordnet.

Beispiel 6-10

für PROGRAM COLLATING SEQUENCE- und ALPHABET-Klausel

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ABC.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER.
    PROGRAM COLLATING SEQUENCE IS ALPHATAB.
SPECIAL-NAMES.
    TERMINAL IS T
    ALPHABET ALPHATAB IS "X" "Y" "Z".
DATA DIVISION.
WORKING-STORAGE SECTION.
77  ITEM-1 PIC X(3) VALUE "ABC".
77  ITEM-2 PIC X(3) VALUE "XYZ".
PROCEDURE DIVISION.
MAIN.
    IF ITEM-1 > ITEM-2
    THEN
      DISPLAY "Richtig sortiert" UPON T
    END-IF
    STOP RUN.
```

Mit der Definition des Alphabetnamens ALPHATAB im SPECIAL-NAMES-Paragrafen wurde dem Zeichen

X die erste, Y die zweite, Z die dritte Position

der Sortierfolge zugewiesen.

Allen anderen Zeichen des EBCDIC-Zeichensatzes wird implizit eine Position der Sortierfolge zugewiesen, da ihre Positionen in der Sortierfolge höher sind als die der angegebenen Zeichen X, Y, Z und ihre Anordnung in der Sortierfolge unverändert aus dem EBCDIC-Zeichensatz übernommen wird.

1. bis 231. Position des EBCDIC-Zeichensatzes entspricht der 4. bis 234. Position der Sortierfolge.

235. bis 256. Position des EBCDIC-Zeichensatzes entspricht der 235. bis 256. Position der Sortierfolge.

A hat also Position 197, B Position 198, C Position 199.

Damit ist der Vergleich ITEM-1 > ITEM-2 wahr.

Beispiel 6-11

für PROGRAM COLLATING SEQUENCE- und ALPHABET-Klausel

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ALPH.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER.
    PROGRAM COLLATING SEQUENCE IS ALPHA.
SPECIAL-NAMES.
    TERMINAL IS T
    ALPHABET ALPHA 1 THRU 247, 251 THRU 256
                   "7" ALSO "8" ALSO "9".
DATA DIVISION.
WORKING-STORAGE SECTION.
77  ITEM-1 PIC X(3) VALUE HIGH-VALUE.
77  ITEM-2 PIC X(3) VALUE "789".
PROCEDURE DIVISION.
P1 SECTION.
VERGLEICH.
    IF ITEM-1 = ITEM-2
    THEN
      DISPLAY "1. Vergleich OK" UPON T
    ELSE
      DISPLAY "1. Vergleich nicht OK" UPON T
    END-IF
    IF ITEM-2 = HIGH-VALUE
    THEN
      DISPLAY "2. Vergleich OK" UPON T
    ELSE
      DISPLAY "2. Vergleich nicht OK" UPON T
    END-IF.
ENDE.
    STOP RUN.
```

Zeichen kleiner 7 bleiben wie in der Sortierfolge der Datenverarbeitungsanlage erhalten, Zeichen größer 9 schließen sich unmittelbar an, werden damit kleiner als 7.Die Zeichen 7, 8, 9 werden an die höchste Stelle gesetzt, wobei 9 als zuletzt angegebenes Zeichen „HIGH-VALUE“ entspricht.

Ergebnis:
1. Vergleich OK
2. Vergleich OK

6.2.7 SYMBOLIC CHARACTERS-Klausel

Syntaxregeln

1. Ein symbolischer Name für ein Zeichen darf in der SYMBOLIC CHARACTERS-Klausel nur einmal angegeben werden.
2. Die Beziehung zwischen jedem einzelnen symbolischen Namen und der jeweils korrespondierenden Ganzzahl ergibt sich aus der Reihenfolge innerhalb der SYMBOLIC CHARACTERS-Klausel: symbolisches-zeichen-1 ist verbunden mit ganzzahl-1, symbolisches-zeichen-2 mit ganzzahl-2 usf.
3. Zu jeder Angabe eines symbolischen Namens muss es eine Angabe von ganzzahl geben.
4. Die durch ganzzahl-1 angegebene Position muss im alphanumerischen maschinenspezifischen Zeichensatz vorhanden sein. Ist IN angegeben, muss die Position in dem Zeichensatz vorhanden sein, der mit alphabetname-2 benannt ist.

Allgemeine Regel

1. symbolisches-zeichen ist eine figurative Konstante.
2. Ist die IN-Angabe nicht gemacht, stellt symbolisches-zeichen-1 das Zeichen dar, dessen Position innerhalb der Sortierfolge des maschineneigenen Zeichensatzes durch ganzzahl-1 bezeichnet ist.
3. Ist die IN-Angabe gemacht, wählt die Zeichenposition ganzzahl-1 ein Zeichen aus dem durch alphabetname-2 genannten Zeichensatz aus. Die interne Darstellung von symbolisches-zeichen-1 ist dieselbe wie die des entsprechenden Zeichens im maschineneigenen Zeichensatz.

Beispiel 6-12

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SYMCHAR.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T
    SYMBOLIC CHARACTERS HEX-0A IS 11.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DRUCK-SATZ.
   02 STEUERBYTE        PIC X.
   02 DRUCK-ZEILE       PIC X(132).
PROCEDURE DIVISION.
MAIN SECTION.
P1.
MOVE HEX-0A TO STEUERBYTE.
DISPLAY STEUERBYTE UPON T.
STOP RUN.
```

Dem elften Zeichen des EBCDIC-Zeichensatz (es entspricht dem sedezimalen Wert 0A) wird der symbolische Name HEX-0A zugeordnet.
Die MOVE-Anweisung bezieht sich auf diesen symbolischen Namen, um den sedezimalen Wert 0A in das Steuerbyte zu übertragen.

6.2.8 CLASS-Klausel

Syntaxregeln

1. Die CLASS-Klausel bietet die Möglichkeit, einen Namen mit einer definierten Zeichenmenge zu verbinden. Auf diesen Klassennamen kann nur in einer Klassenbedingung Bezug genommen werden. Die Zeichen, die als Werte von literal-4, literal-5, ... angegeben werden, bilden die besondere Zeichenmenge, die klassenname benennt.
   Der Wert jedes Literals gibt an:
   a. die Ordnungszahl (Positionsnummer) eines Zeichens im alphanumerischen maschinenspezifischen Zeichensatz, wenn das Literal numerisch ist.
   b. das aktuelle Zeichen selbst, wenn das Literal alphanumerisch ist. Wenn der Wert des Literals mehrere Zeichen enthält, gehört jedes dieser Zeichen zu der mit klassenname bezeichneten Zeichenmenge.
2. Ist THROUGH angegeben, gehört die mit literal-4 beginnende und mit literal-5 endende Folge von Zeichen innerhalb des maschinenspezifischen Zeichensatzes zu der durch klassenname angegebenen Zeichenmenge. Diese Zeichenfolge darf sowohl in aufsteigender als auch in absteigender Reihenfolge angegeben werden.

Beispiel 6-13

```
SPECIAL-NAMES.
    CLASS HEXA-ZEICHEN
    194 THRU 199, 241 THRU 250.
```

194 bis 199 entsprechen den Buchstaben A bis F

241 bis 250 entsprechen den Ziffern 0 bis 9

6.2.9 CURRENCY SIGN-Klausel

Syntaxregeln

1. literal-6 muss ein alphanumerisches Literal sein. Es darf aber keine figurative Konstante sein. Es muss aus einem Zeichen bestehen, das nicht eines der folgenden ist:
   - Ziffern 0 bis 9
   - Großbuchstaben A, B, C, D, E, N, P, R, S, V, X, Z und das Leerzeichen
   - Kleinbuchstaben a - z
   - Sonderzeichen

     * (Stern)

     + (Plus)

     - (Minus)

     , (Komma)

     . (Punkt)

     : (Doppelpunkt)

     ; (Semikolon)

     ((öffnende Klammer)

) (schließende Klammer)

     " (Doppelhochkomma)

     ’ (Hochkomma)

     / (Schrägstrich)

     = (Gleichheitszeichen)
2. Wird die CURRENCY SIGN-Klausel nicht verwendet, kann nur das Währungszeichen $ in einer Maskenzeichenfolge angegeben werden.
3. In der CURRENCY SIGN-Klausel ist auch das Eurozeichen € zugelassen.

6.2.10 DECIMAL-POINT IS COMMA-Klausel

Syntaxregel

1. Die DECIMAL-POINT IS COMMA-Klausel bedeutet, dass die Funktionen von Komma und Punkt in den Zeichenfolgen der PICTURE-Klausel und in numerischen Literalen vertauscht werden.

Allgemeine Regel

1. Die DECIMAL-POINT IS COMMA-Klausel bewirkt die Vertauschung der Funktionen von Dezimalpunkt und Komma in numerischen Literalen und in Maskenzeichenfolgen. Bei Verwendung dieser Klausel muss der in einem numerischen Literal oder in einer Maskenzeichenfolge verlangte Dezimalpunkt durch ein Komma dargestellt werden. Der Dezimalpunkt muss für die Funktion verwendet werden, die normalerweise durch das Komma ausgeführt wird.

6.2.11 REPOSITORY-Paragraf

Der REPOSITORY-Paragraf definiert die Klassen-und Schnittstellennamen, die im Geltungsbereich der entsprechenden ENVIRONMENT DIVISION verwendet werden können.

Format

```
REPOSITORY.
[{  CLASS klassenname-1 [EXPANDS klassenname-2 USING {klassenname-3 |
interfacename-3}... ]

  | INTERFACE interfacename-1 [EXPANDS interfacename-2 USING {klassenname-4 |
interfacename-4}... ]

 }... .
]
```

1. Ist klassenname-1 identisch mit dem Namen der Klassendefinition in der der REPOSITORY-Paragraf spezifiziert ist, wird dieser Eintrag ignoriert.
2. Ist interfacename-1 identisch mit dem Namen der Interfacedefinition in der der REPOSITORY-Paragraf spezifiziert ist, wird dieser Eintrag ignoriert.
3. Ist programmprototypname-1 identisch mit dem Namen der Programmdefinition in der der REPOSITORY-Paragraf spezifiziert ist, wird dieser Eintrag ignoriert.
4. Im COBOL-Standard 2002 ist die EXPANDS-Angabe innerhalb einer Klassen- und Interfacedefinitionen verboten, die die USING-Klausel in ihrem ID-Paragrafen nutzen. Dieser Compiler erlaubt jedoch eine weitere Schachtelungsebene für die EXPANDS-Angabe. Um Namenskonflikte zu vermeiden und um die Übersichtlichkeit und Portabilität solcher Programme zu gewährleisten, sollte diese Schachtelung jedoch nur in begründeten Fällen verwendet werden.
5. klassenname-2, klassenname-3, und interfacename-3 müssen im gleichen REPOSITORY-Paragraf definiert sein, in dem klassenname-1 definiert ist.
6. klassenname-4, interfacename-4 und interfacename-2 müssen im gleichen REPOSITORY-Paragraf definiert sein, in dem interfacename-1 definiert ist.
7. klassenname-2 und interfacename-2 dürfen nur in einer EXPANDS-Angabe angesprochen werden.
8. Die Expansion einer parametrisierten Klasse darf nicht direkt oder indirekt als aktueller Parameter der gleichen parametrisierten Klasse verwendet werden.
9. Die Expansion eines parametrisierten Interface darf nicht direkt oder indirekt als aktueller Parameter des gleichen parametrisierten Interface verwendet werden.

Allgemeine Regeln

1. Innerhalb einer Klassen-/Schnittstellendefinition dürfen im REPOSITORY-Paragraf keine Klassen-/Interfacenamen angegeben werden, die direkt oder indirekt auf die aktuell definierte Klasse in ihrem REPOSITORY-Paragraf Bezug nehmen.
2. klassenname-1 ist der Name der Klasse, der für die gesamte Quelleinheit gilt, die die entsprechende Environment Division enthält.
3. interfacename-1 ist der Name der Schnittstelle, der für die gesamte Quelleinheit gilt, die die entsprechende Environment Division enthält.
4. programmprototypname-1 ist der Name des Programmprototyps, der für die gesamte Quelleinheit gilt, die die entsprechende Environment Division enthält.
5. Fehlt die EXPANDS-Angabe, dann müssen im externen Repository-Daten enthalten sein, die klassenname-1 bzw. interfacename-1 charakterisieren.

6. Wenn in der Übersetzungsgruppe ein Programmprototyp für programmprototypname-1 vorhanden ist, hängt die Verwendung dieser Prototypdefinition von der Steuerung des Repositoryzugriffs ab (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).
7. klassenname-3 und interfacename-3 sind die aktuellen Parameter der parametrisierten Klasse klassenname-2.
8. klassenname-4 und interfacename-4 sind die aktuellen Parameter des parametrisierten Interface interfacename-2.
9. Die Anzahl der aktuellen Parameter in der USING-Klausel muss mit der Anzahl der formalen Parameter in der USING-Klausel der CLASS-ID von klassenname-2 übereinstimmen. Ebenso muss die Spezifikation (als Klasse oder Interface) der einander entsprechenden aktuellen und formalen Parameter übereinstimmen.
   klassenname-1 wird aus der parametrisierten Klasse klassenname-2 erzeugt mittels Ersetzen aller formalen Parameter durch die entsprechenden aktuellen Parameter sowie durch Ersetzen von klassenname-2 durch klassenname-1.
10. Die Anzahl der aktuellen Parameter in der USING-Klausel muss mit der Anzahl der formalen Parameter in der USING-Klausel der INTERFACE-ID von interfacename-1 übereinstimmen. Ebenso muss die Spezifikation (als Klasse oder Interface) der einander entsprechenden aktuellen und formalen Parameter übereinstimmen.
    interfacename-1 wird aus dem parametrisierten Interface interfacename-2 erzeugt mittels Ersetzung aller formalen Parameter durch die entsprechenden aktuellen Parameter sowie Ersetzen von interfacename-2 durch interfacename-1.

6.3 INPUT-OUTPUT SECTION

Funktion

In der INPUT-OUTPUT SECTION wird Folgendes festgelegt:

- die Definition jeder im Programm verwendeten Datei,
- die Zuordnung der Dateien zu externen Geräten,
- die Art der Datenübertragung zwischen den Geräten und dem Programm.

Die INPUT-OUTPUT SECTION ist in zwei Paragrafen unterteilt:

- der FILE-CONTROL-Paragraf, der die im Programm verwendeten Dateien bezeichnet und externen Geräten zuordnet,
- der I-O-CONTROL-Paragraf, der spezielle Ein-/Ausgabe-Techniken angibt.

Format

```
INPUT-OUTPUT SECTION.

 FILE-CONTROL. {dateisteuerungseintrag}...
[I-O-CONTROL. [ein-ausgabe-steuerungseintrag]]
```

Syntaxregeln

1. Die INPUT-OUTPUT SECTION ist wahlfrei. Ist die INPUT-OUTPUT SECTION angegeben, muss auch der FILE-CONTROL-Paragraf angegeben werden.
2. Innerhalb einer Interface-Definition darf die INPUT-OUTPUT SECTION nicht angegeben werden.

6.3.1 FILE-CONTROL-Paragraf

Funktion

Im FILE-CONTROL-Paragrafen wird jeder Datei ein Name zugeordnet. Die Dateien werden einem oder mehreren externen Geräten zugeordnet und die zur Dateiverarbeitung benötigten Informationen werden bereitgestellt. Sie geben an, wie die Daten organisiert sind und wie auf sie zugegriffen werden soll.

Format 1 Dateisteuerungseinträge für sequenzielle Dateiorganisation

```
FILE-CONTROL.

     SELECT-Klausel
     ASSIGN-Klausel
    [ORGANIZATION-Klausel]
    [PADDING CHARACTER-Klausel]
    [RECORD DELIMITER-Klausel]
    [ACCESS MODE-Klausel]
    [RESERVE-Klausel]
    [FILE STATUS-Klausel].
```

Format 2 Dateisteuerungseinträge für relative Dateiorganisation

```
FILE-CONTROL.

     SELECT-Klausel
     ASSIGN-Klausel
    [ACCESS MODE-Klausel]
    [FILE STATUS-Klausel]
     ORGANIZATION-Klausel
    [RESERVE-Klausel].
```

Format 3 Dateisteuerungseinträge für indizierte Dateiorganisation

```
FILE-CONTROL.

     SELECT-Klausel
     ASSIGN-Klausel
    [ACCESS MODE-Klausel]
    [ALTERNATE RECORD KEY-Klausel]
    [FILE STATUS-Klausel]
     ORGANIZATION-Klausel
     RECORD KEY-Klausel
    [RESERVE-Klausel].
```

Syntaxregeln

1. Die SELECT-Klausel muss der erste Dateisteuerungseintrag sein. Alle anderen Klauseln können in **beliebiger Reihenfolge** angegeben werden.
2. Wenn in der Dateierklärung die GLOBAL-Klausel angegeben ist, dürfen die in den Klauseln des FILE-CONTROL-Paragrafen angesprochenen Datenfelder nicht in einer LOCAL-STORAGE SECTION definiert sein.
3. Für Sortierdateien dürfen nur die SELECT- und die ASSIGN-Klausel angegeben werden.

Nachfolgend sind zuerst die SELECT- und ASSIGN-Klausel und daran anschließend in alphabetischer Reihenfolge die übrigen Klauseln beschrieben.

6.3.2 SELECT-Klausel

Funktion

Die SELECT-Klausel wird benutzt, um jeder Datei im Programm einen Namen zu geben.

Format 1

```
SELECT [OPTIONAL] dateiname
```

Format 2

```
SELECT dateiname
```

Syntaxregel für beide Formate

1. Unter dateiname ist hier der Name zu verstehen, mit dem eine Datei in der Übersetzungseinheit angesprochen wird (interner Dateiname). Innerhalb eines Programms, einer Factory, eines Objekts oder einer Methode darf ein dateiname in nur einer SELECT-Klausel auftreten.

Syntaxregeln für Format 1

2. Jede Datei, die in einer SELECT-Klausel angegeben ist, muss genau einen Dateibeschreibungseintrag (FD) in der DATA DIVISION des gleichen Programms, der gleichen Methode, der gleichen Factory oder des gleichen Objekts haben.
3. Die OPTIONAL-Angabe wird für Dateien benötigt, die zur Ablaufzeit des Programms nicht immer vorhanden zu sein brauchen. Ist eine Datei zur Ablaufzeit nicht vorhanden, verzweigt die erste READ-Anweisung für diese Datei zu der zugehörigen Ende-Bedingung. Die OPTIONAL-Angabe ist nur bei Dateien wirksam, die im INPUT-, I-O- oder EXTEND-Modus eröffnet sind.

Syntaxregeln für Format 2

4. dateiname bezeichnet eine Sortierdatei.
5. Zu jeder Sortierdatei, die in einer SELECT-Klausel angegeben ist, muss genau ein Sortierdateibeschreibungseintrag (SD) in der DATA DIVISION des gleichen Programms oder der gleichen Methode vorhanden sein.
6. Dieses Format darf innerhalb einer Klassendefinition nur in einer Methodendefinition angegeben werden.

> ¡ Zur Laufzeit werden maximal 30 Zeichen des Dateinamens verwendet (z. B. für Meldungen und Defaulteinstellungen).

Allgemeine Regeln

Format 1

1. Wenn beim Programmlauf kein SET-FILE-LINK-Kommando für eine Datei gegeben wurde, legt das Laufzeitsystem bei OPEN OUTPUT eine Datei mit dem Namen FILE.COB85.linkname an. Der Linkname wird aus den Angaben in der ASSIGN-Klausel gebildet (siehe „ASSIGN-Klausel").
2. Bezieht sich die OPTIONAL-Angabe auf eine *externe* Datei, muss in allen Programmen, die diese externe Datei beschreiben, OPTIONAL angegeben werden.

Format 2

3. Außer der ASSIGN-Klausel sind keine weiteren Klauseln erlaubt.
4. Jede Sortierdatei, die in der DATA DIVISION beschrieben ist, muss genau einmal im FILE-CONTROL-Paragrafen als Dateiname bezeichnet werden.

5. Die folgenden Dateinamen dürfen nicht in SELECT-Klauseln innerhalb eines Programms mit SORT vorkommen.

```
MERGExx(xx = 01,...,99)
SORTIN
SORTINxx(xx = 01,...,99)
SORTOUT
SORTWK
SORTWKx(x = 1,...,9)
SORTWKxx(xx = 01,...,99)
SORTCKPT
```

6.3.3 ASSIGN-Klausel

Funktion

Die ASSIGN-Klausel weist einer Datei des COBOL-Programms ein externes Gerät zu. Für jede Datei im Programm wird eine ASSIGN-Klausel benötigt.

Format 1 für sequenzielle Dateiorganisation

```
ASSIGN { TO { PRINTER [literal-1] | herstellername-1 | literal-2 }... | USING datenname-1 }
```

Format 2 für relative und indizierte Dateiorganisation

```
ASSIGN {TO literal-1... | USING datenname-1}
```

Format 3 für Sortierdateien

```
ASSIGN TO {herstellername-1 | literal}
```

Syntaxregeln

Nur für **sequenziell organisierte Dateien** gilt:

1. Die Angabe PRINTER ohne literal-1 bezeichnet die logische Systemdatei SYSLST.
   Die Angabe PRINTER literal-1 bezeichnet eine Druckdatei. In beiden Fällen reserviert der Compiler das Vorschubsteuerzeichen, das dem Benutzer nicht zugänglich ist.
2. herstellername-1 bezeichnet Geräte, die folgendermaßen benannt und zugeordnet sind:

Gerätename	zugeordnete Systemdatei
PRINTER01 - PRINTER99	SYSLST01 - SYSLST99
SYSIPT	SYSIPT
SYSOPT	SYSOPT

3. Mit PRINTER oder herstellername-1 zugewiesene Dateien dürfen keine externen Dateien sein.

   Für **sequenzielle, relative und indizierte Dateiorganisation** gilt:
4. literal-1, literal-2 müssen alphanumerische Literale sein und dürfen keine figurative Konstante sein. datenname-1 muss alphanumerisch sein.
5. literal-1, literal-2 oder der Inhalt von datenname-1 geben den Linknamen für die Datei an.
6. Bei Angabe eines Literals wird der Linkname aus den ersten 8 Zeichen gebildet. Diese müssen innerhalb des Programms eindeutig sein. Ist das letzte Zeichen der so gebildeten Linknamen ein Bindestrich (-), so wird dieser durch ein #-Zeichen ersetzt.
7. datenname-1 darf nicht gekennzeichnet sein.
8. datenname-1 muss als alphanumerisches Datenfeld in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION definiert sein.

   Für **Sortierdateien** gilt:
9. herstellername ist DISC
10. literal ist SORTWK.

Allgemeine Regeln

1. Die Dateiorganisation muss in der ORGANIZATION-Klausel angegeben werden (siehe "ORGANIZATION-Klausel ").
2. In der ASSIGN-Klausel wird nur der erste Eintrag analysiert, alle weiteren Einträge werden ignoriert (PRINTER literal-1 ist ein Eintrag).
3. Bezieht sich die ASSIGN-Klausel auf eine *externe* Datei, muss in allen Programmen, die diese externe Datei beschreiben, die ASSIGN-Klausel in der gleichen Form verwendet werden. Der Inhalt des Literals bzw. des Datennamens kann dagegen unterschiedlich sein.

6.3.4 ACCESS MODE-Klausel

Funktion

Die ACCESS MODE-Klausel bestimmt die Art des Zugriffs auf die Sätze einer Datei.

Format 1 für sequenzielle Dateiorganisation

```
ACCESS MODE IS SEQUENTIAL
```

Format 2 für relative Dateiorganisation

```
ACCESS MODE IS {  SEQUENTIAL [RELATIVE KEY IS datenname-1]
                | {RANDOM | DYNAMIC} RELATIVE KEY IS datenname-1
               }
```

Syntaxregeln

1. Die RELATIVE KEY-Angabe bezeichnet das Schlüsseldatenfeld, dessen Inhalt zum Auffinden oder Absetzen eines logischen Datensatzes in einer Datei benutzt wird.
2. datenname-1 darf nicht subskribiert oder indiziert sein.
3. Das durch datenname-1 angesprochene Datenfeld muss als Ganzzahl ohne Vorzeichen definiert sein. Das Symbol "P" darf in seiner PICTURE-Maskenzeichenfolge nicht angegeben sein.
4. datenname-1 darf nicht innerhalb der Datensatzerklärung der zugehörigen Datei angegeben sein.

Format 3 für indizierte Dateiorganisation

```
ACCESS MODE IS {SEQUENTIAL | RANDOM | DYNAMIC}
```

Allgemeine Regeln

Für **sequenzielle, relative und indizierte Dateiorganisation:**

1. Ist die ACCESS MODE-Klausel nicht angegeben, wird ACCESS MODE IS SEQUENTIAL angenommen.
2. SEQUENTIAL bedeutet, dass Datensätze sequenziell gelesen oder geschrieben werden, d.h. der nächste logische Datensatz der Datei wird zur Verfügung gestellt, wenn eine READ-Anweisung ausgeführt wird bzw. der nächste logische Datensatz wird in die Datei gebracht, wenn eine WRITE-Anweisung ausgeführt wird.
3. Ist für eine *externe* Datei die ACCESS MODE-Klausel angegeben, so muss in allen anderen Programmen, die diese externe Datei beschreiben, eine gleichlautende ACCESS MODE-Klausel angegeben sein. Für **relativ organisierte Dateien** muss zusätzlich datenname-1 ein in allen Programmen gleiches externes Datenfeld sein.

Für **relative und indizierte Dateiorganisation** gilt zusätzlich:

4. RANDOM bedeutet, dass Datensätze wahlfrei anhand eines Schlüssels gelesen oder geschrieben werden, d.h. es wird unter Verwendung des RECORD KEY bzw.RELATIVE KEY zugegriffen.
5. DYNAMIC bedeutet, dass wahlfreier und sequenzieller Zugriff gemischt werden können.

Nur für **relative Dateiorganisation** gilt noch:

6. Das mit datenname-1 bezeichnete Datenfeld dient zur Weitergabe einer relativen Satznummer zwischen dem Benutzer und dem Ein-/Ausgabe-System.
   Alle in einer Datei vorhandenen Datensätze sind eindeutig durch ihre relativen Satznummern bezeichnet. Die relative Satznummer eines gegebenen Datensatzes gibt die logische Position des Datensatzes

innerhalb der Satzfolge der Datei an. Der erste logische Datensatz hat die relative Satznummer eins (1), und die folgenden logischen Datensätze haben die relativen Satznummern 2, 3, 4...

7. Der RELATIVE KEY darf nicht 0 sein.

6.3.5 ALTERNATE RECORD KEY-Klausel

Funktion

Die ALTERNATE RECORD KEY-Klausel definiert einen zum Primärschlüssel alternativen Satzschlüssel, über den auf die Sätze einer indizierten Datei zugegriffen werden kann.

Format

```
ALTERNATE RECORD KEY IS datenname [WITH DUPLICATES]
```

Syntaxregeln

1. datenname muss ein alphanumerisches oder nationales Datenfeld in einer Datensatzerklärung bezeichnen. Diese Datensatzerklärung muss dem Dateinamen zugeordnet sein, dem auch die ALTERNATE RECORD KEY-Klausel untergeordnet ist.
2. datenname kann jedes Feld fester Länge innerhalb des Datensatzes sein. Das Feld kann 1 bis 127 Byte lang sein. datenname darf gekennzeichnet, aber nicht subskribiert oder indiziert werden.
3. datenname darf sich nicht auf eine Datengruppe beziehen, in der ein Element mit einer OCCURS-Klausel mit DEPENDING ON-Angabe enthalten ist.
4. Enthält die Datei Sätze variabler Länge, muss der Satzschlüssel in den ersten x Zeichenpositionen des Datensatzes enthalten sein. Dabei entspricht x der minimalen Datensatzgröße für die Datei (siehe „RECORD-Klausel"), d.h., die Länge des Satzschlüssels muss vom kürzesten Datensatz voll abgedeckt werden.
5. Pro Datei können mehrere (maximal 30) alternative Satzschlüssel angegeben werden.
6. datenname darf sich nicht auf ein Datenfeld beziehen, dessen Anfangsadresse (die am weitesten links stehende Zeichenposition) identisch ist mit der Anfangsadresse des Primärschlüsselfeldes bzw. eines anderen Alternativschlüsselfeldes. Ansonsten sind Überlappungen von Primärschlüsselfeld und Alternativschlüsselfeld erlaubt.

Allgemeine Regeln

1. Die Datenbeschreibung von datenname darf nicht von der entsprechenden Datenbeschreibung bei Erstellung der Datei abweichen. Das gleiche gilt für die relative Position des als Schlüssel definierten Datenfeldes. Die Anzahl der Alternativschlüssel muss mit der Anzahl der Alternativschlüssel bei Erstellung der Datei übereinstimmen.
2. WITH DUPLICATES gibt an, dass der Wert des zugehörigen Alternativschlüssels in mehr als einem Datensatz auftreten kann. Sätze mit identischen Schlüsselwerten werden auch als „Duplikate“ bezeichnet. Ist WITH DUPLICATES nicht angegeben, muss der Wert des zugehörigen Alternativschlüssels eindeutig sein, d.h. er darf nicht mehr als einmal in der Datei auftreten.
3. Sind mehrere Datensatzerklärungen in einer Datei vorhanden, braucht datenname nur in einer dieser Datensatzerklärungen beschrieben zu werden. Die durch datenname festgelegte Position des Schlüsselfeldes im Datensatz wird implizit für alle Schlüssel in den anderen Datensatzerklärungen der Datei berücksichtigt.
4. Bezieht sich die ALTERNATE RECORD KEY-Klausel(n) auf eine *externe* Datei, so müssen in jedem Programm, das diese externe Datei verwendet, die gleiche Anzahl ALTERNATE RECORD KEY-Klauseln angegeben sein, wobei auch die Länge und Lage der Schlüsselfelder im Datensatz sowie ggf. die DUPLICATES-Angabe übereinstimmend festgelegt sein muss.

6.3.6 FILE STATUS-Klausel

Funktion

Die FILE STATUS-Klausel gibt ein Datenfeld an, das während der Verarbeitung die Zustände der Ein-/Ausgabe-Operationen anzeigt. Ferner wird durch Angabe eines weiteren Datenfeldes ein zusätzlicher Fehlerschlüssel zur Verfügung gestellt.

Format

```
FILE STATUS IS datenname-1 [, datenname-2]
```

Syntaxregeln

1. datenname-1 und datenname-2 müssen in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION der DATA DIVISION definiert sein.
2. datenname-1 muss ein zwei Zeichen langes alphanumerisches Datenfeld sein.
3. datenname-2 muss ein sechs Byte langes Gruppenfeld mit folgenden Aufbau sein:

```
01 datenname-2.
   02 datenname-2-1 PIC 9(2) COMP.
   02 datenname-2-2 PIC X(4).
```

Allgemeine Regeln

1. Ist die FILE STATUS-Klausel angegeben, so wird vom Laufzeitsystem der Ein-/Ausgabe-Zustand nach datenname-1 übertragen.
2. Falls angegeben ist datenname-2 wie folgt belegt:
   a. Hat der Inhalt von datenname-1 den Wert Null, dann ist der Inhalt von datenname-2 undefiniert.
   b. Hat der Inhalt von datenname-1 einen von Null verschiedenen Wert, dann enthält datenname-2 den zusätzlichen Fehlerschlüssel. Der Wert 64 in datenname-2-1 zeigt an, dass es sich dabei um den (BS2000-) DVS-Code handelt, der Wert 96 in datenname-2-1 zeigt an, dass es sich um den (POSIX-) SIS-Code handelt.
   Das Kommando HELP DMS <inhalt von datenname-2-2> bzw. HELP SIS <inhalt von datenname-2-2> liefert nähere Informationen zum jeweiligen Fehlerschlüssel.
3. Der Ein-/Ausgabe-Zustand wird übertragen während der Ausführung jeder OPEN-, CLOSE-, READ-, WRITE-, REWRITE- oder START-Anweisung, die sich auf die angegebene Datei bezieht, und vor Ausführung jeder entsprechenden USE-Prozedur (siehe „Ein-/Ausgabe-Zustand“, "Allgemeine Konzepte ").

6.3.7 ORGANIZATION-Klausel

Funktion

Die ORGANIZATION-Klausel definiert den logischen Aufbau einer Datei.

Format 1 für sequenzielle Dateiorganisation

```
[ORGANIZATION { SEQUENTIAL | LINE SEQUENTIAL }
```

Format 2 für relative Dateiorganisation

```
[ORGANIZATION IS] RELATIVE
```

Format 3 für indizierte Dateiorganisation

```
[ORGANIZATION IS] INDEXED
```

Allgemeine Regeln

1. Die Dateiorganisation wird zum Zeitpunkt der Erstellung einer Datei festgelegt und kann später nicht verändert werden.
2. Ist die ORGANIZATION-Klausel nicht angegeben, wird ORGANIZATIONISSEQUENTIAL angenommen.
3. Für eine externe Datei muss in allen Programmen, die diese externe Datei beschreiben, die gleiche ORGANIZATION-Klausel angegeben sein.

6.3.8 PADDING CHARACTER-Klausel

Funktion

Die PADDING CHARACTER-Klausel dient dazu, ein Block-Füllzeichen anzugeben.

Format

```
PADDING CHARACTER IS {datenname | literal}
```

Die PADDING CHARACTER-Klausel wird vom Compiler als Kommentar behandelt.

6.3.9 RECORD DELIMITER-Klausel

Funktion

Mit der RECORD DELIMITER-Klausel kann die Methode zur Bestimmung der Satzlänge auf Externspeichern angegeben werden.

Format

```
RECORD DELIMITER IS {STANDARD-1 | BS2000}
```

Die RECORD DELIMITER-Klausel wird vom Compiler als Kommentar behandelt.

6.3.10 RECORD KEY-Klausel

Funktion

Die RECORD KEY-Klausel definiert den primären Satzschlüssel, über den auf die Sätze einer indizierten Datei zugegriffen wird.

Format

```
RECORD KEY IS datenname
```

Syntaxregeln

1. datenname muss ein alphanumerisches oder nationales Datenfeld in einer Datensatzerklärung bezeichnen. Diese Datensatzerklärung muss dem Dateinamen zugeordnet sein, dem auch die RECORD KEY-Klausel untergeordnet ist.
2. datenname kann jedes Feld fester Länge innerhalb des Datensatzes sein. Das Feld kann 1 bis 255 Byte lang sein. datenname darf gekennzeichnet, aber nicht subskribiert oder indiziert werden.
3. datenname darf sich nicht auf eine Datengruppe beziehen, in der ein Element mit einer OCCURS-Klausel mit DEPENDING ON-Angabe enthalten ist.
4. Enthält die Datei Sätze variabler Länge, muss der Satzschlüssel in den ersten x Bytes des Datensatzes enthalten sein. Dabei entspricht x der minimalen Datensatzgröße für die Datei (siehe „RECORD-Klausel"), d.h. die Länge des Satzschlüssels muss vom kürzesten Datensatz voll abgedeckt werden.

Allgemeine Regeln

1. Die Werte der Satzschlüssel müssen für alle Datensätze der Datei eindeutig sein.
2. Die Datenbeschreibung von datenname darf nicht von der Datenbeschreibung bei Erstellung der Datei abweichen. Das gleiche gilt für die relative Position des als Schlüssel definierten Datenfeldes im Datensatz.
3. Sind mehrere Datensatzerklärungen in einer Datei vorhanden, braucht datenname nur in einer dieser Datensatzerklärungen beschrieben zu werden. Die durch datenname festgelegte Position des Schlüsselfeldes im Datensatz wird implizit für alle Schlüssel in den anderen Datensatzerklärungen der Datei berücksichtigt.
4. Bezieht sich die RECORD KEY-Klausel auf eine *externe* Datei, so muss in jedem Programm, das diese externe Datei verwendet, die gleichlautende RECORD KEY-Klausel angegeben sein, wobei insbesondere die Länge und Lage der Schlüsselfelder im
Datensatz übereinstimmend festgelegt sein muss.

6.3.11 RESERVE-Klausel

Funktion

Mit der RESERVE-Klausel kann der Benutzer die Anzahl der Ein-/Ausgabe-Bereiche angeben, die dem Programm vom Compiler zugeordnet werden.

Format

```
RESERVE ganzzahl [AREA | AREAS]
```

Die RESERVE-Klausel wird vom Compiler als Kommentar behandelt.

6.3.12 I-O-CONTROL-Paragraf

Funktion

Der I-O-CONTROL-Paragraf definiert die Ereignisse, bei deren Eintreten Wiederanlaufpunkte erstellt werden sollen, und den Speicherbereich, der von verschiedenen Dateien gleichzeitig benutzt werden soll. Ferner definiert er spezielle Ein-/Ausgabe-Bedingungen und für **sequenziell organisierte Dateien** gibt er auch deren Lage auf Mehrdateienbänder an.

Format 1 für sequenzielle Dateiorganisation

```
I-O-CONTROL.

    [MULTIPLE FILE TAPE-Klausel]...
    [RERUN-Klausel]...
    [SAME AREA-Klausel]...
```

Format 2 für relative und indizierte Dateiorganisation

```
I-O-CONTROL.

     [RERUN-Klausel]...
     [SAME AREA-Klausel]...
```

Syntaxregel

1. Innerhalb einer Klassendefinition darf der I-O-CONTROL-Paragraf nur in einer Methodendefinition angegeben werden.

6.3.13 MULTIPLE FILE TAPE-Klausel

Funktion

Die MULTIPLE FILE TAPE-Klausel wird benötigt, wenn sich auf einer Magnetbandspule mehr als eine Datei befindet.

Format

```
MULTIPLE FILE TAPE CONTAINS {dateiname-1 [POSITION ganzzahl-1]}...
```

Syntaxregel

1. ganzzahl hat einen Wert zwischen 1 und 3315.

Allgemeine Regeln

1. Wenn alle Dateinamen in der Reihenfolge, wie sie auf einer Spule vorkommen, aufgeführt sind, kann die POSITION-Angabe entfallen.
2. Ist eine der Dateien nicht aufgeführt, müssen die Positionen relativ zum Bandanfang angegeben werden.
3. Unabhängig von der Zahl der Dateien auf einer Spule müssen nur diejenigen aufgeführt werden, die das Programm benutzt.
4. Zu jedem Zeitpunkt darf nur auf einer Spule höchstens eine Datei eröffnet sein.
5. REWIND kann nur ausgeführt werden, wenn die letzte Datei des Bandes verarbeitet wurde.
6. Ist für eine *externe* Datei die MULTIPLE FILE TAPE-Klausel angegeben, muss in allen Programmen, die diese externe Datei beschreiben, die MULTIPLE FILE TAPE-Klausel angegeben sein. Sind Positionsnummern angegeben, müssen diese in allen Programmen die gleichen sein.

Für weitere Angaben siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1].

6.3.14 RERUN-Klausel

Funktion

Eine RERUN-Klausel zeigt an, wann und wohin Wiederanlaufpunkte auszugeben sind. Ein Wiederanlaufpunkt beschreibt den Zustand des Programms zu einem angegebenen Zeitpunkt der Programmausführung. Er wird vom Betriebssystem auf Anforderung des Programmes erzeugt und enthält alle nötigen Informationen, um das Programm an diesem Punkt wieder anlaufen zu lassen. Die RERUN-Klausel steuert solche Anforderungen des COBOL-Programms. Nähere Informationen zur RERUN-Klausel siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]

Durch die RERUN-Klausel wird ein Wiederanlauf von Programmen ermöglicht, die eine SORT- oder MERGE-Anweisung enthalten.

Format 1 für sequenzielle Dateiorganisation

```
RERUN [ON {dateiname-1 | herstellername}]
       EVERY {  { [END OF] {REEL | UNIT} | ganzzahl-1 RECORDS } OF dateiname-2
              | ganzzahl-2 CLOCK-UNITS
              | bedingungsname
             }
```

Format 2 für relative und indizierte Dateiorganisation

```
RERUN [ON {herstellername | dateiname-1}]
       EVERY {  ganzzahl-1 RECORDS OF dateiname -2
              | ganzzahl-2 CLOCK-UNITS
              | bedingungsname
             }
```

Syntaxregeln

1. dateiname-1 muss der Name einer sequenziellen Banddatei sein; sie muss in der FILE SECTION beschrieben sein.
2. herstellername hat das Format
   SYSnnn,
   wobei 000 nnn 244.

   nnn bestimmt die Art der Verarbeitung:Ist nnn 200, werden Wiederanlaufpunkte abwechselnd in zwei Wiederanlaufdateien geschrieben. Das abwechselnde Schreiben auf zwei Dateien macht einen Wiederanlauf von den letzten zwei Wiederanlaufpunkten möglich.
3. Kommt der gleiche Herstellername auch in der SELECT-Klausel vor, so muss er zu einer Banddatei gehören.
4. Die END OF REEL/UNIT-Angabe darf nur benutzt werden, wenn dateiname-2 eine sequenzielle Datei beschreibt. REEL und UNIT haben die gleiche Bedeutung.
5. herstellername muss in der RERUN-Klausel verwendet werden, wenn ganzzahl-1 RECORDS oder ganzzahl-2 CLOCK-UNITS angegeben ist.
6. Unter Beachtung folgender Einschränkungen können für dateiname-2 mehrere RERUN-Klauseln angegeben werden:
   a. Sind mehrere ganzzahl-1 RECORDS-Angaben angegeben, darf der gleiche dateiname-2 nur einmal darin vorkommen.
   b. Sind mehrere END OF REEL- bzw. END OF UNIT-Angaben angegeben, darf der gleiche dateiname-2 nur einmal darin vorkommen.

7. dateiname-1 bzw. herstellername bezeichnet die Datei, auf die Wiederanlaufpunkte ausgegeben werden sollen.
8. Wiederanlaufpunkte werden wie folgt geschrieben:
   a. Ist dateiname-1 angegeben, werden die Wiederanlaufpunkte auf jede Spule oder jeden Datenträger ausgegeben, die oder der für dateiname-1 zugewiesen ist.
   b. Ist herstellername angegeben, werden die Wiederanlaufpunkte folgendermaßen geschrieben:
   Wenn herstellername einem Dateinamen aus der SELECT-Klausel zugewiesen ist, muss es sich dabei um eine Banddatei handeln. Die Wirkung für die Ausgabe der
   Wiederanlaufpunkte ist die gleiche, als ob der Dateiname direkt in der RERUN-Klausel angegeben worden wäre (siehe a).
   Wenn herstellername keiner Datei aus einer SELECT-Klausel zugewiesen ist, muss zur Laufzeit eine Plattenspeicherdatei zugewiesen sein, sonst wird vom Laufzeitsystem eine Datei mit dem Namen progid.RERUN.herstellername zugewiesen. Nur in diesem Fall werden Wiederanlaufpunkte auf Plattenspeicher geschrieben.
   c. Sind in der ASSIGN-Klausel und in der RERUN END OF REEL-Klausel die gleichen SYS-Nummern angegeben, wird der Wiederanlaufpunkt ans Ende des Ausgabebandes geschrieben.
9. Es gibt vier Arten der RERUN-Klausel, abhängig von den Bedingungen, unter denen Wiederanlaufpunkte verlangt werden.
   a. END OF REEL oder END OF UNIT ohne ON-Klausel:
   Wiederanlaufpunkte werden nach dateiname-2 geschrieben, der eine Ausgabedatei bezeichnen muss.
   b. END OF REEL oder END OF UNIT mit dateiname-1 in der ON-Klausel:
   Wiederanlaufpunkte werden nach dateiname-1 geschrieben, der eine Ausgabedatei bezeichnen muss. Zusätzlich wird die übliche Spulenendebehandlung für dateiname-2 durchgeführt. dateiname-2 kann eine Ein- oder Ausgabe-Datei sein.
   c. END OF REEL oder END OF UNIT mit herstellername in der ON-Klausel:
   Wiederanlaufpunkte werden in eine getrennte Datei geschrieben (siehe 8b). dateiname-2 kann eine Ein- oder Ausgabe-Datei sein.
   d. ganzzahl-1 RECORDS:
   Wiederanlaufpunkte werden immer dann in die durch dateiname-1 bzw. herstellername bezeichnete Datei geschrieben, wenn ganzzahl-1 Datensätze verarbeitet wurden. dateiname-2 kann eine Ein- oder Ausgabe-Datei sein mit beliebiger Organisation oder Zugriff; sie darf nicht in der USING/GIVING-Angabe oder in einer INPUT/OUTPUT-Prozedur beim Sortieren angesprochen werden (siehe Abschnitt „Sortieren von Datensätzen").
10. Die CLOCK-UNITS-Angabe fasst der Compiler als Kommentar auf.
11. Die Bedingungsnamen-Angabe fasst der Compiler ebenfalls als Kommentar auf. 12. Wird mit dateiname-1 eine *externe* Datei angegeben, ist das Verhalten undefiniert.

Format 3 für Sortieren von Datensätzen

```
RERUN ON {dateiname-1 | herstellername} EVERY SORT [OF dateiname-2 ... ]
```

Syntaxregeln

1. dateiname-2... muss in einer FD-Erklärung definiert sein.
2. dateiname-1 darf nicht in einer FD-Erklärung definiert sein.
3. herstellername darf in keiner SELECT-Klausel vorkommen.

Allgemeine Regeln

1. Wird OF nicht angegeben, wird die Fixpunktausgabe vor jedem Sortiervorgang ausgeführt.
2. Ist dateiname-2 angegeben, werden Fixpunkte nur für diesen spezifischen Sortiervorgang ausgegeben.

6.3.15 SAME AREA-Klausel

Funktion

Die SAME AREA-Klausel gibt an, welche Dateien während der Programmausführung einen Ein-/Ausgabe-Bereich gemeinsam benutzen.

Format 1	gilt für alle Dateien außer Sortierdateien, falls nicht RECORD angegeben ist.
Format 2	gilt für Sortierdateien

Format 1 für alle Dateiorganisationsformen

```
SAME [RECORD] AREA FOR dateiname-1 {dateiname-2}...
```

Syntaxregeln

1. In einem Programm ist mehr als eine SAME AREA-Klausel zulässig. Es ist dabei Folgendes zu beachten:

   Ein Dateiname darf nicht in mehreren SAME AREA-Klauseln vorkommen. Gleiches gilt für die SAME RECORD AREA-Klausel.

   Ein Dateiname darf gleichzeitig in einer SAME AREA- und SAME RECORD AREA-Klausel vorkommen. In diesem Fall müssen alle Dateinamen, die in der SAME AREA-Klausel aufgeführt sind, auch in der SAME RECORD AREA-Klausel stehen. Die SAME RECORD AREA-Klausel darf darüber hinaus weitere Dateinamen enthalten, die nicht in der SAME AREA-Klausel stehen.
2. Die SAME AREA-Klausel zeigt an, dass die aufgeführten Dateien (keine Sortierdateien) die ihnen zugewiesenen Ein-/Ausgabe-Bereiche gemeinsam benutzen sollen.
3. Die SAME RECORD AREA-Klausel zeigt an, dass die aufgeführten Dateien denselben Speicherbereich zur Verarbeitung des aktuellen logischen Datensatzes benutzen sollen.

   Ein logischer Datensatz in der SAME RECORD AREA-Klausel gilt als logischer Datensatz aller zur Ausgabe eröffneten Dateien, deren Name in dieser SAME RECORD AREA-Klausel aufgeführt sind. Er ist ebenfalls logischer Datensatz derjenigen Datei aus dieser Klausel, aus der die letzte Eingabe erfolgte. Dies entspricht einer impliziten Überlagerung aller Datensatzbereiche, wobei die Datensätze auf die am weitesten links stehende Zeichenposition ausgerichtet sind.
4. dateiname-1 und dateiname-2 dürfen keine XML-organisierten Dateien sein.

Allgemeine Regeln

1. Ist SAME AREA angegeben, darf zu jedem Zeitpunkt nur eine der Dateien eröffnet sein.
2. Ist die RECORD-Angabe gemacht, dürfen alle aufgeführten Dateien gleichzeitig eröff net sein.
3. Für Dateien, die sowohl in der SAME AREA- als auch in der SAME RECORD AREA-Klausel angegeben sind, gilt Allgemeine Regel 1.
4. Für *externe* Dateien darf die SAME [RECORD] AREA-Klausel nicht angegeben werden.

Format 2 für Sortieren von Datensätzen

```
SAME {RECORD | SORT | SORT-MERGE} AREA FOR dateiname-1 {dateiname-2}...
```

Syntaxregeln

1. SORT und SORT-MERGE sind gleichwertig.
2. Wird SAME SORT AREA oder SAME SORT-MERGE AREA benutzt, muss wenigstens einer der angegebenen Dateinamen eine Sortier- oder Mischdatei bezeichnen.
   Dateien, die keine Sortier- oder Mischdatei bezeichnen, können ebenfalls in der Klausel angegeben werden.

3. Es dürfen mehrere SAME AREA-Klauseln in einem Programm enthalten sein. Dabei sind folgende Einschränkungen zu beachten:
   a. Ein Dateiname darf nicht in mehr als einer SAME RECORD AREA-Klausel erscheinen.
   b. Ein Dateiname, der eine Sortierdatei bezeichnet, darf nicht in mehr als einer SAME SORT AREA- oder SAME SORT-MERGE AREA-Klausel erscheinen.
   c. Steht ein Dateiname, der keine Sortierdatei bezeichnet, in einer SAME AREA-Klausel und in einer (oder mehreren) SAME SORT AREA- oder SAME SORT-MERGE AREA-Klausel(n), muss jede Datei, die in der SAME AREA-Klausel angegeben ist, auch in der SAME SORT AREA- bzw. SAME SORT-MERGE AREA-Klausel angegeben sein.
4. Die in der SAME SORT AREA-, SAME SORT-MERGE AREA- bzw. SAME RECORD AREA-Klausel bezeichneten Dateien brauchen nicht alle die gleiche Organisation oder Zugriffsart zu haben.
5. dateiname-1 und dateiname-2 dürfen keine XML-organisierten Dateien sein.

Allgemeine Regel

1. Die SAME RECORD AREA-Klausel gibt an, dass sich zwei oder mehr Dateien den Speicherplatz teilen sollen, in dem der aktuelle logische Datensatz verarbeitet wird. Alle Dateien können zur selben Zeit geöffnet sein. Ein logischer Datensatz im „gleichen Datensatzbereich“ wird als ein logischer Datensatz jeder geöffneten Ausgabedatei betrachtet, deren Dateiname in der SAME RECORD AREA-Klausel vorkommt, sowie als logischer Datensatz der zuletzt gelesenen Eingabedatei, deren Dateiname in der SAME RECORD AREA-Klausel vorkommt. Dies ist gleichbedeutend mit einer impliziten Redefinierung des Internspeicherbereichs, wobei die Datensätze linksbündig an der ersten Zeichenstelle ausgerichtet sind.

7 DATA DIVISION

In diesem Kapitel werden folgende Themen behandelt:

- RENAMES-Klausel
- SIGN-Klausel
- SYNCHRONIZED-Klausel
- TYPE-Klausel
- TYPEDEF-Klausel
- USAGE-Klausel
- DISPLAY-Angabe
- NATIONAL-Angabe
- BINARY-Angabe oder COMPUTATIONAL-Angabe oder COMPUTATIONAL-5-Angabe
- COMPUTATIONAL-1-Angabe
- COMPUTATIONAL-2-Angabe
- COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe
- INDEX-Angabe
- OBJECT REFERENCE-Angabe
- POINTER-Angabe
- PROGRAM-POINTER-Angabe
- VALUE-Klausel

7.1 Allgemeine Beschreibung

Zwei Arten von Daten werden von einer COBOL-Übersetzungseinheit verarbeitet:

1. Daten, gespeichert auf einem externen Datenträger und
2. programminterne Daten.

Der erste Datentyp ist in Form von Datensätzen in Dateien gesammelt, die zweite Art muss vom Anwender als Datensätze oder untergeordnete Datenfelder erklärt werden.

Um Daten von ihrer speziellen Darstellung auf externen Datenträgern und in Datenverarbeitungsanlagen möglichst unabhängig zu machen, werden die Eigenschaften oder Inhalte der Daten in Beziehung zu einem Standardformat statt zu einem anlagenorientierten Format beschrieben.

Die verwendete Methode zur Datenbeschreibung gestattet die Unterscheidung der physischen und systemabhängigen Merkmale einerseits und der konzeptuellen Eigenschaften andererseits.

Physische und einige systemabhängige Merkmale von Daten auf einem externen Datenträger werden in einer COBOL-Übersetzungseinheit festgelegt, um einen wirksamen Gebrauch von speziellen Techniken zu ermöglichen.

1. Unter die physischen Merkmale von Daten fallen
    a. die Anordnung der logischen Datensätze innerhalb der physischen Grenzen des externen Datenträgers;
    b. der Satzmodus, in dem die Daten auf dem externen Datenträger gespeichert sind.
2. Unter die systemabhängigen Merkmale von Daten fällt die Beschreibung, unter der eine Datei auf einem externen Datenträger identifiziert wird.

Die konzeptuellen Eigenschaften von Daten auf einem externen Datenträger beziehen sich auf die logischen Einheiten der Daten, die logischen Datensätze, und tragen keine physischen oder systemabhängigen Merkmale.

Die Dateneigenschaften für Dateien sind in den Dateierklärungen, die für Datensätze in den Datensatzerklärungen beschrieben.

Die Datenerklärung in einer COBOL-Übersetzungseinheit ist getrennt von der Erklärung der Ablaufprozeduren. Dies ermöglicht es dem Programmierer, die Datenerklärung in einer Vielfalt von Möglichkeiten zu ändern, ohne dabei die sich darauf beziehenden Prozeduren ändern zu müssen. Deshalb können die Prozeduren einer COBOL-Übersetzungseinheit bis zu einem bestimmten Grad als datenunabhängig betrachtet werden.

7.1.1 Struktur der DATA DIVISION

Die DATA DIVISION, die einer der Abschnitte in einer Übersetzungseinheit ist, wird in nachfolgende SECTIONs unterteilt. Eine SECTION muss nicht angegeben werden, wenn ihre logische Funktion nicht gebraucht wird. Wenn SECTIONs verwendet werden, ist ihre Reihenfolge, so wie im allgemeinen Format dargestellt, einzuhalten.

7.1.2 Allgemeines Format

```
   DATA DIVISION.

[ FILE SECTION.
  [ dateierklärung.{datensatzerklärung}...
  | sortierdateierklärung.{datensatzerklärung}...
  | listendateierklärung.
  ] ...
]

[ WORKING-STORAGE SECTION.
  [ 77-stufenerklärung.
  | datensatzerklärung.
  ] ...
]

[ LOCAL-STORAGE SECTION.
  [ 77-stufenerklärung.
  | datensatzerklärung.
  ] ...
]

[ LINKAGE SECTION.
  [ 77-stufenerklärung.
  | datensatzerklärung.
  ] ...
]

[ REPORT SECTION.
  [listenerklärung. {leistenerklärung.}...]...
]

[ SUB-SCHEMA SECTION.
  datenbasiserklärung.
]
```

7.1.3 FILE SECTION

Alle Daten, die extern gespeichert sind oder gespeichert werden sollen, müssen in der FILE SECTION beschrieben werden, ehe sie durch ein COBOL-Programm verarbeitet werden können. In der FILE SECTION wird der Aufbau der Dateien festgelegt. Jede Datei wird durch eine Dateierklärung und eine oder mehrere Datensatzbeschreibungen definiert. Datensatzbeschreibungen werden unmittelbar anschließend an die Dateierklärung geschrieben.

7.1.4 WORKING-STORAGE SECTION

Information, die zur internen Verwendung bestimmt ist, muss in der WORKING-STORAGE SECTION beschrieben werden. Die WORKING-STORAGE SECTION beschreibt Datensätze und strukturunabhängige Datenfelder (siehe „Allgemeines Format“), die nicht ein Teil externer Dateien sind. Daten, die in der WORKING-STORAGE SECTION beschrieben sind, sind beim Aufruf der Quelleinheit in „last-used“ oder Initialzustand (siehe Abschnitt „Sortieren von Datensätzen"). Die WORKING-STORAGE SECTION darf in einem Programm, einer Fabrik(Factory)- oder Objekt-Definition oder einer Methoden-Definition innerhalb einer Klassen-Definition angegeben werden.

7.1.5 LOCAL-STORAGE SECTION

Die in der LOCAL-STORAGE SECTION beschriebenen Daten sind automatische Daten. Daten werden mit den Werten initialisiert, die in der zugehörigen VALUE-Klausel angegeben sind. Die LOCAL-STORAGE SECTION kann sowohl in rekursiven als auch nicht rekursiven, zulässigen Quelleinheiten vorkommen. Daten in der LOCAL-STORAGE SECTION sind bei jedem Aufruf der Quelleinheit im Initialzustand. Die LOCAL-STORAGE SECTION darf keine EXTERNAL-Klausel enthalten. Die LOCAL-STORAGE SECTION darf in der DATA DIVISION eines Programms oder einer Methode angegeben werden.

Anmerkung:
Informationen zur internen Verwendung können in der WORKING-STORAGE SECTION oder in der LOCAL-STORAGE SECTION beschrieben werden.

7.1.6 LINKAGE SECTION

Information, die von einer Quelleinheit zu einer anderen übermittelt wird, muss in der LINKAGE SECTION beschrieben werden. Die LINKAGE SECTION steht in einer aufgerufenen Quelleinheit und beschreibt Datenfelder in der aufrufenden Quelleinheit, die die aufgerufene Quelleinheit ansprechen darf. Die LINKAGE SECTION hat primär in einer Methode Bedeutung und in einem Programm, das als Unterprogramm gerufen wird. Darüber hinaus werden in der LINKAGE-SECTION Daten beschrieben, denen erst zur Ablaufzeit Speicher zugewiesen wird (siehe Abschnitt „BASED-Klausel"). Wird ein Datenfeld der LINKAGE SECTION eines Programms angesprochen, das nicht von einer anderen Quelleinheit aufgerufen oder dem noch kein Speicher zugewiesen wurde, ist das Verhalten unbestimmt.
Struktur und Inhalt der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION sind grundsätzlich identisch.

7.1.7 REPORT SECTION

Der Inhalt und das Aussehen aller Listen, die durch den Listenprogramm-Teil erzeugt werden, müssen in der REPORT SECTION erklärt werden (siehe Kapitel „Listenprogramm (Report-Writer)").

7.1.8 SUB-SCHEMA SECTION

Informationen über die Beschreibung von Datenbankstrukturen müssen in der SUB-SCHEMA SECTION angegeben werden.
Die SUB-SCHEMA SECTION darf nicht in einem Programm angegeben werden, für das die RECURSIV-Klausel spezifiziert ist. Sie wird sonst ignoriert.

(Näheres dazu siehe in der UDS/SQL-Beschreibung [6]).

7.2 Dateierklärung

Funktion

Die **Dateierklärung (FD)** bestimmt den physischen Aufbau und die Datensatznamen einer Datei. Eine **Sortierdateierklärung (SD)** enthält Informationen, die die physische Struktur, die Bezeichnung und Größe der Datensätze in einer Sortierdatei betreffen.

Eine Dateierklärung (FD bzw. SD) muss für jede im Programm zu verarbeitende Datei angegeben werden. Die in dieser Erklärung enthaltene Information bezieht sich allgemein auf die physischen Verhältnisse dieser Datei, d.h. die Beschreibung der Daten, wie sie auf dem Eingabe- oder Ausgabe-Träger vorliegen.

7.2.1 Formate der Dateierklärung

Format 1 für sequenzielle Dateiorganisation

```
FD  dateiname

    [BLOCK CONTAINS–Klausel]
    [CODE-SET–Klausel]
    [DATA RECORDS–Klausel]
    [EXTERNAL–Klausel]
    [GLOBAL–Klausel]
    [LABEL RECORDS–Klausel]
    [LINAGE–Klausel]
    [RECORD–Klausel]
    [RECORDING MODE–Klausel]
    [VALUE OF–Klausel].
```

Format 2 für relative Dateiorganisation

```
FD  dateiname

    [BLOCK CONTAINS–Klausel]
    [DATA RECORDS–Klausel]
    [EXTERNAL–Klausel]
    [GLOBAL–Klausel]
    [LABEL RECORDS–Klausel]
    [RECORD–Klausel]
    [VALUE OF–Klausel].
```

Format 3 für indizierte Dateiorganisation

```
FD  dateiname

    [BLOCK CONTAINS–Klausel]
    [DATA RECORDS–Klausel]
    [EXTERNAL–Klausel]
    [GLOBAL–Klausel]
    [LABEL RECORDS–Klausel]
    [RECORD–Klausel]
    [VALUE OF–Klausel].
```

Format 4 für Sortierdateien

```
SD  sortierdateiname

    [DATA RECORDS–Klausel]
    [LABEL RECORDS–Klausel]
    [RECORD–Klausel]
    [RECORDING MODE–Klausel]
```

Syntaxregeln für Format 1, 2 und 3

1. FD bezeichnet den Anfang einer Dateierklärung.
2. dateiname muss mit dem Dateinamen einer SELECT-Klausel übereinstimmen.

3. Die Reihenfolge der auf den Dateinamen folgenden Klauseln ist beliebig.
4. Der Dateierklärung müssen eine oder mehrere Datensatzerklärungen folgen.
5. Die zugehörigen Datensatzerklärungen müssen von der Kategorie alphabetisch, alphanumerisch, national oder numerisch sein.
6. Wenn in der Dateierklärung die GLOBAL-Klausel angegeben ist, dürfen die in den Klauseln der Dateierklärung angesprochenen Datenfelder nicht in einer LOCAL-STORAGE SECTION definiert sein.

Allgemeine Regeln für Format 1, 2 und 3

1. Folgende Tabelle gibt einen Überblick über die Funktionen der Klauseln der Dateierklärung.

Klausel	Funktion
BLOCK CONTAINS-Klausel	Angabe der physischen Blocklänge
CODE-SET-Klausel	Festlegen der Zeichencodeart zur Ausgabe von Daten auf externen Geräten
DATA RECORDS-Klausel	Bezeichnet die Namen der Datensätze der Datei
EXTERNAL-Klausel	Deklariert eine Datei als extern
GLOBAL-Klausel	Deklariert eine Datei als global
LABEL RECORDS-Klausel	Angabe der Namen und Werte der in der Datei enthaltenen Kennsätze
LINAGE-Klausel	Angabe der Größe der logischen Seite („Blattgröße“). Außerdem wird die Definition eines Seitenrumpfes und eines Seitenfußes ermöglicht.
RECORD-Klausel	Angabe der Längen der logischen Datensätze
RECORDING MODE-Klausel	Angabe des Formates der logischen Datensätze
VALUE OF-Klausel	Gibt die Werte einiger Datenfelder eines Kennsatzes an

Tabelle 9: Funktionen der Klauseln der Dateierklärung

2. Die Klausel CODE-SET gilt nur für **sequenzielle Dateiorganisation**.

Syntaxregeln für Format 4

1. Die Stufenbezeichnung SD kennzeichnet den Anfang der Sortierdateierklärung und muss dem Dateinamen vorangehen.
2. Die Klauseln, die dem Namen der Datei folgen, sind wahlweise und die Reihenfolge ihres Auftretens ist beliebig.

3. Eine oder mehrere Datensatzerklärungen für datenname-1,... müssen der Sortierdateierklärung folgen.
4. Die zugehörigen Datensatzerklärungen müssen von der Kategorie alphabetisch, alphanumerisch, national oder numerisch sein.
5. Wenn in der Dateierklärung die GLOBAL-Klausel angegeben ist, dürfen die in den Klauseln der Sortierdateierklärung angesprochenen Datenfelder nicht in einer LOCAL-STORAGE SECTION definiert sein.

Allgemeine Regeln für Format 4

1. sortierdateiname bezeichnet die Sortierdatei.
2. datenname-1... in der DATA RECORDS-Klausel bezieht sich auf die Datensätze derjenigen Datensatzbeschreibungen, die zu der jeweiligen Sortierdateierklärung (SD) gehören.
3. Die FILE SECTION muss für jede Sortierdatei eine Sortierdateierklärung enthalten, d.h. für jede Datei, die als erster Operand innerhalb einer SORT- oder MERGE-Anweisung genannt wird.
4. Die RECORDING MODE-Klausel spezifiziert das Organisationsformat der Daten auf externen Geräten.
5. Die LABEL RECORDS-Klausel ist wahlweise; fehlt die Klausel, wird LABEL RECORDS ARE OMITTED angenommen. Dies bedeutet, dass vorhandende Kennsätze überschrieben werden.
6. Wenn die LABEL RECORDS ARE STANDARD-Klausel angegeben wird, müssen die Arbeitsbänder zum Sortieren Standardkennsätze haben; es wird aber keine Kennsatzbehandlung durchgeführt. In diesem Fall bleiben die Kennsätze vollständig erhalten.
7. Die DATA RECORDS-Klausel gibt die Namen der Datensätze an, die sortiert werden sollen.
8. Ist mehr als ein Datenname vorhanden, enthält die Datei mehr als eine Art von Datensätzen. Diese Datensätze können unterschiedliche Länge, unterschiedliches Format usw. haben. Die Reihenfolge in der sie aufgeführt werden, ist unwichtig.
9. Wenn für den logischen Datensatz einer Datei mehr als eine Datensatzbeschreibung angegeben ist, belegen diese Datensätze automatisch denselben Internspeicherbereich; dies entspricht einer impliziten Redefinition des Bereichs.
10. Die RECORDING MODE-, DATA RECORDS- und die RECORD-Klausel sind wahlweise, da der Compiler die Modi, Namen und Größe der Datensätze anhand der zugehörigen Datensatzbeschreibung bestimmen kann.
11. Falls ein zu einer Sortierdateierklärung gehöriger Datensatz eine OCCURS-Klausel mit DEPENDING ON-Angabe enthält, werden Datensätze von variabler Länge angenommen (weitere Information über die vom Compiler gemachten Annahmen, wenn die RECORDING MODE-Klausel weggelassen wird, siehe „RECORDING MODE-Klausel").
12. Sortierdateinamen können nur in der SORT-, MERGE- und RETURN-Anweisung verwendet werden.

7.2.2 Klauseln für die Dateierklärung

7.2.3 BLOCK CONTAINS-Klausel

Funktion

Die BLOCK CONTAINS-Klausel gibt die maximale Größe eines physischen Blockes an.

Format

```
BLOCK CONTAINS [ganzzahl-1 TO ] ganzzahl-2 {CHARACTERS | RECORDS }
```

Syntaxregeln

1. Ein Block muss mindestens 20 und darf höchstens 32763 Bytes enthalten.
2. Die Angabe von CHARACTERS oder RECORDS zeigt an, ob die Blocklänge als Vielfaches von Bytes oder logischen Datensätzen angegeben ist.
3. Wenn weder CHARACTERS noch RECORDS angegeben ist, wird CHARACTERS angenommen.
4. ganzzahl-1 TO ganzzahl-2 gibt die Anzahl der Bytes oder Datensätze in einem Block an, abhängig von der verwendeten Angabe CHARACTERS oder RECORDS.
5. Falls nur ganzzahl-2 angegeben ist, so bezieht sich diese Angabe auf die maximale Blocklänge. Sind ganzzahl-1 und ganzzahl-2 angegeben, so beziehen sie sich auf die minimale bzw. maximale Blocklänge. Die Angabe von ganzzahl-1 dient jedoch nur zu Dokumentationszwecken und wird vom Compiler als Kommentar behandelt.

   Die **maximale** Blocklänge, die durch diese Klausel angegeben wird, hat folgende Bedeutung:

   Die Blöcke dürfen nicht größer, können aber kleiner als die angegebene Länge sein; dies kommt besonders bei ungeblockten oder geblockten Datensätzen mit variabler Datensatzlänge vor.
6. Wird die Angabe CHARACTERS verwendet, so ist die Blocklänge als Anzahl der im Block enthaltenen Bytes anzugeben. In diesem Fall müssen ganzzahl-1 und ganzzahl2 Füllzeichen und 4 Bytes für das Satzlängenfeld eines jeden Datensatzes des Blockes einschließen.
7. Wird die Angabe CHARACTERS verwendet und ist nur ganzzahl-2 angegeben, stellt ganzzahl-2 die Länge des physischen Blockes dar. Sind ganzzahl-1 und ganzzahl-2 angegeben, so beziehen sie sich auf die minimale bzw. maximale physische Blocklänge.
8. Wird die Angabe RECORDS verwendet, so wird die Blocklänge als Anzahl logischer Datensätze angegeben. In diesem Fall berechnet der Compiler die Blocklänge, indem er die Anzahl der Bytes der maximalen Satzlänge mit dem in ganzzahl-2 angegebenen Wert multipliziert. Bei Datensätzen variabler Länge kommen noch 4 Bytes für das Satzlängenfeld hinzu.
9. Wenn die BLOCK CONTAINS-Klausel nicht angegeben ist, dann nimmt der Compiler an, dass die Datensätze nicht geblockt sind, d.h. BLOCK CONTAINS 1 RECORDS wird angenommen.

   Anmerkung:
   Die BLOCK CONTAINS-Klausel kann daher weggelassen werden, wenn **alle** Blöcke nur einen einzigen Datensatz enthalten.

Allgemeine Regeln

1. Die folgende Tabelle zeigt, in welcher Weise der Compiler die Blocklängen im Sinne von Bytes berechnet und welche Angaben die BLOCK CONTAINS-Klausel dazu enthält.

   Die in Tabelle 10 verwendeten Bezeichnungen bedeuten:

   F = Datensätze fester Länge

   V = Datensätze variabler Länge

   BL = Blocklänge

   SL = Datensatzlänge

SL_{max} = maximale Datensatzlänge

BLF = Blocklängenfeld (hat den Wert 4)

SLF = Datensatzlängenfeld (hat den Wert 4)

n = ganze Zahl

Satzformat	**BLOCK CONTAINS [ganzzahl-1 TO] ganzzahl-2**	
	CHARACTERS	**RECORDS**
feste Länge	BL = ganzzahl-2 (ganzzahl-2 = n * SL)	BL = ganzzahl-2 * SL
variable Länge	BL = ganzzahl-2 + BLF	BL = ganzzahl-2 *(SL_{max} + SLF) + BLF

Tabelle 10: Berechnung der maximalen Blocklänge

2. Wenn für eine *externe* Datei die BLOCK CONTAINS-Klausel angegeben ist, muss in allen Programmen, die diese externe Datei beschreiben, die BLOCK CONTAINS-Klausel angegeben sein, wobei die aus den Angaben in der BLOCK CONTAINS-Klausel errechnete Blockgröße gleich sein muss, unabhängig davon, ob sie sich aus der Anzahl „RECORDS“ oder der Anzahl „CHARACTERS“ ergibt.

Anmerkung:
Die Angabe CHARACTERS sollte verwendet werden, wenn die RECORDS-Angabe eine zu ungenaue Blocklänge ergeben würde.
Enthält z.B. ein Block vier Sätze, einer 50 Bytes lang und drei 100 Bytes lang, so berechnet der Compiler - variable Satzlänge mit maximal 100 Bytes vorausgesetzt - bei `BLOCK CONTAINS 4 RECORDS` eine Blocklänge (einschließlich Blocklängenfeld) von 4*(100+4)+4 = 420 Bytes. Da der Block tatsächlich (ohne Blocklängenfeld) nur (50+4)+3*(100+4) = 366 Bytes benötigt, kann mit der Angabe von `BLOCKCONTAINS 366 CHARACTERS` die erforderliche Blockgröße exakt bestimmt werden.

7.2.4 CODE-SET-Klausel

Funktion

Die CODE-SET-Klausel definiert die Zeichencodevereinbarungen, in denen Daten auf externen Geräten dargestellt werden.

Format

```
CODE-SET IS alphabetname
```

Syntaxregeln

1. Ist die CODE-SET-Klausel für eine Datei angegeben, müssen in dieser Datei

   alle Daten mit USAGE IS DISPLAY beschrieben werden. Alle numerischen Daten mit Vorzeichen müssen mit SIGN IS SEPARATE beschrieben sein.
2. Die alphabetname zugeordnete ALPHABET-Klausel darf nicht die Literal-Angabe enthalten (siehe „SPECIAL-NAMES-Paragraf").

Allgemeine Regeln

1. Ist die CODE-SET-Klausel angegeben, spezifiziert alphabetname die Zeichencode-Vereinbarungen für die Darstellung der Daten auf externen Geräten. Außerdem ist damit der Algorithmus für die Umwandlung des Zeichencodes des externen Gerätes in den Zeichencode der Datenverarbeitungsanlage und umgekehrt festgelegt. Die Code-Umwandlung geschieht während der Ausführung einer Lese- bzw. Schreiboperation (siehe „SPECIAL-NAMES-Paragraf").
2. Ist die CODE-SET-Klausel nicht angegeben, wird die Zeichencodeart der Datenverarbeitungsanlage verwendet (EBCDIC).
3. Bezieht sich die CODE-SET-Klausel auf eine *externe* Datei, muss in allen Programmen, die diese externe Datei beschreiben, eine gleichlautende CODE-SET-Klausel angegeben sein.

7.2.5 DATA RECORDS-Klausel

Funktion

Die DATA RECORDS-Klausel dient nur zur Dokumentation, sie bezeichnet die Datensätze der Datei durch Namen.

Format

```
DATA {RECORD | RECORDS}  {IS | ARE} {datenname-1}...
```

Syntaxregeln

1. datenname-1 ist der Name eines Datensatzes. datenname-1 muss in der Dateierklärung die Stufennummer 01 vorangehen.
2. Das Vorhandensein von mehr als einem Datennamen zeigt an, dass die Datei mehr als eine Art von Datensätzen enthält. Diese Datensätze können verschiedene Länge, verschiedene Formate etc. haben. Ihre Reihenfolge hat keine Bedeutung.

7.2.6 EXTERNAL-Klausel

Function

Mit der EXTERNAL-Klausel kann eine Datei als extern definiert werden. Auf externe Dateien kann von jedem Programm, in dem die Datei beschrieben ist, zugegriffen werden.

Format

```
IS EXTERNAL
```

Syntaxregel

1. Namen von externen Dateien dürfen maximal 30 Zeichen lang sein.

Allgemeine Regeln

1. Ist eine Datei als extern definiert, sind die Datensätze dieser Datei implizit ebenfalls extern.
2. Wenn die Dateierklärung für eine sequenzielle Datei die LINAGE-Klausel und die EXTERNAL-Klausel enthält, ist das Sonderregister LINAGE-COUNTER implizit ein externes Datenfeld.
3. Als Namen für externe Dateien dürfen nicht verwendet werden:
   - externe Datensatznamen aus der WORKING-STORAGE SECTION anderer Übersetzungseinheiten in der Ablaufeinheit,
   - PROGRAM-ID-Namen der Ablaufeinheit, ausgenommen Programmnamen von inneren Programmen eines geschachtelten Programms,
   - Namen, die in der ENTRY-Anweisung als Einsprungpunkte verwendet werden,
   - Namen, die eine Schnittstelle benennen (LZS-Namen u.a.),
4. Die FILE STATUS-Klausel wirkt für externe Dateien stets programmlokal, d.h. der Dateizustand wird nur von Ein-/Ausgabe-Operationen in dem Programm versorgt, das eine entsprechende Angabe in der Dateibeschreibung enthält.
5. Die EXTERNAL-Klausel darf nicht in Datei- oder Datensatzerklärungen von Dateien angegeben werden, die einen gemeinsamen Ein-/Ausgabebereich (SAME RECORD AREA-Klausel) belegen.
6. Die EXTERNAL-Klausel darf nicht für Dateien angegeben werden, die den Systemgeräten SYSIPT, SYSOPT, PRINTER oder PRINTERnn zugewiesen sind.
7. Die EXTERNAL-Klausel darf nicht für Dateien angegeben werden, für die Benutzerkennsätze und entsprechende USE-Prozeduren vereinbart sind.
8. Eine externe Datei muss in allen Programmen, die auf sie zugreifen wollen, durch explizite Klauseln oder durch implizite Standardwerte weitgehend gleich beschrieben sein. Die folgende Tabelle zeigt Art und Umfang der notwendigen Übereinstimmung:

Klauseln / Angaben	in allen Programmen
Name der externen Datei	gleich in der vollen Länge (30 Zeichen)
OPTIONAL-Angabe (SELECT-Klausel)	gleiche Angabe*)
ASSIGN TO datenname	gleiche Zuweisungsform
ASSIGN TO PRINTER literal	gleiche Zuweisungsform
ORGANIZATION-Klausel	gleiche Organisationsform
ACCESS MODE-Klausel	gleiche Zugriffsmethode
RELATIVE KEY-Angabe	gleiche Anzahl Ziffern
RECORD KEY-Klausel	gleiche Länge und Position
ALTERNATE RECORD KEY-Klausel	gleiche Anzahl, Position, Länge und DUPLICATES-Angabe
BLOCK CONTAINS-Klausel	gleiche Blockgröße in Bytes
MULTIPLE FILE TAPE-Klausel	gleiche Positionsnummer

RECORD-Klausel	gleiche minimale und maximale Satzlänge
LABEL RECORDS-Klausel	gleiche Angabe*)
REPORT-Klausel (Report Writer)	gleiche Angabe*)
LINAGE-Klausel	gleiche Angabe*)
CODE SET-Klausel	gleiche Angabe*)
RECORDING MODE-Klausel	gleiche Angabe*)

*) Gleiche Angabe heißt: Die betreffende Klausel darf entweder in keinem der Programme angegeben sein oder muss in allen Programmen gleich angegeben sein.

Alle Programme, die auf die gleiche externe Datei zugreifen, müssen mit dem gleichen Wert der Compiler-Option ENABLE-UFS-ACCESS bzw. mit dem gleichen Modulformat übersetzt worden sein (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

ℹ Ist eine Datei als extern definiert, so ist der zugehörige Dateiname nicht implizit ein globaler Name.

Zusatzregeln, abhängig vom Modulformat

Wird das Format *OMF generiert (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]), so gilt für Namen von externen Dateien:

1. Das achte Zeichen darf kein Bindestrich sein.
2. Es werden nur die ersten 7 Zeichen des Namens zur Identifizierung verwendet. Deshalb sollten diese Zeichen eindeutig sein für jeden externen Namen in Ablaufeinheit.

7.2.7 GLOBAL-Klausel

Funktion

Die GLOBAL-Klausel kann nur innerhalb eines geschachtelten Programms verwendet werden. Sie legt fest, dass ein Dateiname (Datenname oder Listenname, siehe dazu Kapitel Datenerklärung bzw. Listenprogramm), global ist. Auf einen globalen Namen kann das Programm, das ihn vereinbart, und jedes in diesem Programm direkt oder indirekt enthaltene Programm zugreifen.

Format

```
IS GLOBAL
```

Syntaxregeln

1. Die GLOBAL-Klausel darf nur in einer Dateierklärung angegeben werden.
2. Die GLOBAL-Klausel darf nicht in Datei- oder Datensatzerklärungen von Dateien ange-geben werden, die einen gemeinsamen Ein-/Ausgabebereich (SAME RECORD AREA-Klausel) belegen.
3. Die GLOBAL-Klausel darf nicht in einer Methoden-Definition angegeben werden.

Allgemeine Regeln

1. Ein Dateiname, dessen Beschreibung eine GLOBAL-Klausel enthält, ist ein globaler Name. Alle einem globalen Namen untergeordneten Datennamen sowie alle Bedingungsnamen, die mit einem globalen Namen in Verbindung stehen, sind globale Namen.
2. Auf einen globalen Namen darf jedes Programm zugreifen, das in dem Programm enthalten ist, das den globalen Namen beschreibt. Der globale Name braucht in dem Programm, das ihn referenziert, nicht nochmals beschrieben zu werden. Bei Referenzen auf gleiche Namen haben die lokalen Namen Vorrang (siehe „Bestimmung des gültigen Namens" im Abschnitt "Lokale und globale Namen").
3. Ist in einer Beschreibung einer sequenziellen Datei sowohl die LINAGE als auch die GLOBAL-Klausel angegeben, so ist das Sonderregister LINAGE-COUNTER ebenfalls global.

7.2.8 LABEL RECORDS-Klausel

Function

The LABEL RECORDS clause specifies whether labels are present, and identifies them if they are.

Format 1 für sequenzielle Dateiorganisation

```
LABEL {RECORD | RECORDS} {IS | ARE} {OMITTED | STANDARD | {datenname-1}...}
```

Format 2 für relative und indizierte Dateiorganisation

```
LABEL {RECORD | RECORDS} {IS | ARE} STANDARD
```

Syntaxregeln

1. Für datenname-1 müssen Datensatzerklärungen für die betreffende Datei vorhanden sein. datenname-1... dürfen nicht als Operanden in der DATA RECORDS-Klausel dieser Datei auftreten.
2. Die OMITTED-Angabe gibt an, dass entweder keine eindeutigen Kennsätze für die Datei vorhanden sind oder dass die vorhandenen Kennsätze nicht standardisiert sind und der Benutzer keine Bearbeitung durch eine USE-Prozedur zur Kennsatzverarbeitung wünscht (z.B. wenn der Benutzer die Kennsätze als Datensätze verarbeiten will).
3. Die STANDARD-Angabe gibt an, dass für die Datei Kennsätze vorhanden sind und dass diese Kennsätze mit Systemkonventionen übereinstimmen (siehe hierzu DVS-Handbuch [9]).
4. Die Angabe von datenname-1 zeigt entweder das Vorhandensein von Benutzerkennsätzen als Zusatz zu Standardkennsätzen oder das Vorhandensein von nicht standardisierten Kennsätzen an. datenname-1 erklärt den Namen eines BenutzerkennsatzDatensatzes.
   Sind Benutzerkennsätze zu verarbeiten, so kann datenname-1 für Dateien, mit Ausnahme von Einheitsdatensatzdateien, erklärt werden.

Allgemeine Regel

1. Die Angabe OMITTED ist nicht für Dateien erlaubt, die mit „ASSIGN TO literal“ zugewiesen sind.
2. Für das Format der Systemkennsätze siehe DVS-Handbuch [9].
3. Benutzerkennsätze sind wie folgt formatiert:
   a. Jeder Benutzerkennsatz ist 80 Bytes lang,
   b. die Stellen 1 bis 3 eines Benutzer-Anfangskennsatzes müssen die Zeichen UHL enthalten,
   c. die Stellen 1 bis 3 eines Benutzer-Endekennsatzes müssen die Zeichen UTL enthalten,
   d. die Stelle 4 zeigt die relative Stellung des Kennsatzes in einer Folge von Anfangs- oder Endekennsätzen an, d.h. diese Stelle muss eine Ziffer von 1 bis 9 enthalten.
      Bei nur jeweils einem Kennsatz (UHL und/oder UTL) muss die Stelle 4 das Zeichen 1 enthalten,
   e. die Stellen 5 bis 80 sind formatiert nach den Angaben des Benutzers.

   Weitere Einzelheiten siehe DVS-Handbuch [9].
4. Benutzer-Anfangskennsätze folgen den Datei-Anfangskennsätzen des Systems, gehen jedoch dem ersten Datensatz voraus.
5. Benutzer-Endekennsätze folgen den Datei-Endekennsätzen des Systems.
6. Nicht standardisierte Kennsätze können 1 bis 4095 Bytes lang sein. Ihr Format und Inhalt werden vom Benutzer festgelegt.
7. Ist datenname-1 angegeben, so müssen alle Bezugnahmen in der PROCEDURE DIVISION auf die angegebenen Datennamen oder auf Datenfelder, die diesen Datennamen untergeordnet sind, in Benutzervereinbarungen (USE-Prozeduren) enthalten sein.

8. Bezieht sich die LABEL RECORDS-Klausel auf eine *externe* Datei, muss in allen Pro-grammen, die diese externe Datei beschreiben, eine gleichwertige LABEL RECORDS-Klausel angegeben werden.

7.2.9 LINAGE-Klausel

Funktion

Die LINAGE-Klausel dient dazu, für eine Ausgabedatei die Länge einer logischen Seite in Form der Zeilenanzahl zu bestimmen. Außerdem kann der Abstand vom oberen bzw. unteren Rand der logischen Seite festgelegt sowie die Zeile innerhalb des Seitenrumpfes angegeben werden, in der der Fußteil beginnen soll.

Format

```
LINAGE IS {datenname-1 | ganzzahl-1} LINES [WITH FOOTING AT {datenname-2 |
ganzzahl-2}]
    [LINES AT TOP {datenname-3 | ganzzahl-3}] [LINES AT BOTTOM {datenname-4 |
ganzzahl-4}]
```

Syntaxregeln

1. datenname-1, datenname-2, datenname-3 und datenname-4 müssen ganzzahlige, numerische Datenelemente ohne Vorzeichen sein.
2. datenname-1, datenname-2, datenname-3 und datenname-4 dürfen Kennzeichner haben.
3. Der Wert von ganzzahl-1 bzw. des Datenfeldes, auf das sich datenname-1 bezieht, muss größer 0 sein.
4. Der Wert von ganzzahl-2 bzw. des Datenfeldes, auf das sich datenname-2 bezieht, muss größer 0 sein, darf aber den Wert von ganzzahl-1 bzw. des Datenfeldes, auf das sich datenname-1 bezieht, nicht überschreiten.
5. Der Wert von ganzzahl-3 und ganzzahl-4 bzw. der Datenfelder, auf die sich datenname-3 und datenname-4 beziehen, darf 0 sein.
6. Die LINAGE-Klausel ist nicht für Dateien erlaubt, die mit OPEN EXTEND eröffnet werden.
7. Die LINAGE-Klausel ist nur für Dateien erlaubt, die PRINTER literal-1 oder literal-2 zugewiesen sind.
8. Mit Hilfe der LINAGE-Klausel kann man die Größe einer logischen Seite in Form der Zeilenzahl bestimmen. Die logische Seitengröße ergibt sich aus der Summe aller Angaben aus der LINAGE-Klausel mit Ausnahme der FOOTING-Angabe. Ist LINES AT TOP oder LINES AT BOTTOM nicht angegeben, so ist der Wert dieser Funktion 0. Fehlt die Angabe für FOOTING, so ist kein Seitenfuß definiert (siehe Tabelle 11).

oberer Rand LINES AT TOP	datenname-3/ ganzzahl-3 > = 0 (Standardwert = 0)
Seitenrumpf LINAGE IS Seitenfuß WITH FOOTING AT	1 2 3 datenname-1/ . } Anzahl der Druckzeilen ganzzahl-1 . . n datenname-2/ (Standardwert = 0 Fußzeilen) ganzzahl-2 . . . datenname-1/ ganzzahl-1
unterer Rand LINES AT BOTTOM	datenname-4 / ganzzahl-4 > = 0 (Standardwert = 0)

Tabelle 11: Aufbau einer logischen Seite

9. Die Größe einer logischen Seite muss nicht unbedingt der Größe einer physischen Seite entsprechen.

10. Der Wert von ganzzahl-1 bzw. des Datenfeldes, auf das sich datenname-1 bezieht, gibt die Anzahl der Zeilen an, die auf einer logischen Seite geschrieben bzw. freigehalten werden können. Dieser Teil der logischen Seite, in dem Zeilen geschrieben bzw. freigehalten werden, wird als Seitenrumpf bezeichnet.
11. Der Wert von ganzzahl-3 bzw. des Datenfeldes, auf das sich datenname-3 bezieht, bestimmt die Anzahl von Zeilen, die den oberen Rand einer logischen Seite darstellen. Dieser Bereich wird nicht beschrieben.
12. Der Wert von ganzzahl-4 bzw. des Datenfeldes, auf das sich datenname-4 bezieht, bestimmt die Anzahl der Zeilen, die den unteren Rand einer logischen Seite darstellen. Dieser Bereich wird nicht beschrieben.
13. Der Wert von ganzzahl-2 bzw. des Datenfeldes, auf das sich datenname-2 bezieht, bestimmt die Zeilennummer innerhalb des Seitenrumpfes, bei der der Seitenfuß beginnt.
14. Als Seitenfuß wird derjenige Teil der logischen Seite bezeichnet, der zwischen der Zeilennummer (ganzzahl-2 / datenname-2) und der Zeilenzahl (ganzzahl-1 / datenname-1) liegt.
15. Die Werte von ganzzahl-1, ganzzahl-3 und ganzzahl-4 bzw. die Werte in den entsprechenden Datenfeldern werden zur Ausführungszeit einer OPEN-Anweisung mit OUTPUT-Angabe dazu verwendet, die jeweilige Zeilenanzahl für jeden der genannten Bereiche innerhalb der ersten logischen Seite festzulegen. Gleichzeitig wird anhand des Wertes von ganzzahl-2 bzw. des Wertes des entsprechenden Datenfeldes der Seitenfuß bestimmt.

    Tritt zur Ausführungszeit einer WRITE-Anweisung mit dem Zusatz ADVANCING eine Seitenüberlaufbedingung auf, so werden die Werte von ganzzahl-1, ganzzahl-3 und ganzzahl-4 dazu verwendet, die jeweilige Zeilenanzahl für jeden der genannten Bereiche in der nächsten logischen Seite festzulegen.

    Der Wert von ganzzahl-2 bzw. des Datenfeldes, auf das sich datenname-2 bezieht, dient dann zur Feststellung des Seitenfußes der nächsten logischen Seite.

Allgemeine Regeln

1. Das COBOL-Register LINAGE-COUNTER wird beim Auftreten einer LINAGE-Klausel generiert. Der LINAGE-COUNTER enthält stets die Zeilennummer, auf die der Drucker innerhalb des Seitenrumpfes positioniert ist. Die erste mögliche Zeile einer logischen Seite, die gedruckt werden kann, hat die Nummer 1.

    Für jede Datei, deren Beschreibung im Kapitel Dateien eine LINAGE-Klausel enthält, ist ein eigener LINAGE-COUNTER vorhanden.
2. Der LINAGE-COUNTER kann von Anweisungen der PROCEDURE DIVISION angesprochen, aber nicht verändert werden. Da innerhalb eines Programmes mehrere LINAGE-COUNTER auftreten können, muss der Benutzer den LINAGE-COUNTER nötigenfalls durch den Dateinamen kennzeichnen.
3. Bei Ausführung einer WRITE-Anweisung für eine Datei wird der zugehörige LINAGE-COUNTER automatisch geändert:
    a. Bei Angabe des Zusatzes ADVANCING PAGE in einer WRITE-Anweisung wird der LINAGE-COUNTER automatisch auf den Wert 1 gesetzt.
    b. Bei Angabe des Zusatzes ADVANCING ganzzahl oder ADVANCING bezeichner-2 in einer WRITE-Anweisung wird der LINAGE-COUNTER jeweils um den Wert von ganzzahl bzw. des Datenfeldes, auf das sich bezeichner-2 bezieht, erhöht.
    c. Fehlt der Zusatz ADVANCING in einer WRITE-Anweisung, so wird der LINAGE-COUNTER automatisch um den Wert 1 erhöht.
    d. Der Wert des LINAGE-COUNTERs wird automatisch auf 1 gesetzt, wenn der Drucker auf die erste Zeile einer folgenden logischen Seite, auf die geschrieben werden kann, positioniert wird (siehe „WRITE-Anweisung").
    e. Bei Ausführung einer OPEN-Anweisung für eine Datei wird der zugehörige LINAGE-COUNTER automatisch auf den Wert 1 gesetzt.
4. Bezieht sich die LINAGE-Klausel auf eine *externe* Datei, muss in allen Programmen, die diese externe Datei beschreiben, eine gleichwertige LINAGE-Klausel angegeben werden. Im Gegensatz zum Standard

verlangt der hier beschriebene Compiler nur gleichartige Angaben (d.h. entweder nur datenname-Angaben oder nur ganzzahl-Angaben). Die Inhalte der Datenfelder bzw. die Zahlwerte dürfen unterschiedlich sein.

7.2.10 RECORD-Klausel

Funktion

Die RECORD-Klausel legt die Länge der Datensätze einer Datei fest und hat Einfluss auf das externe Datensatzformat (für **sequenziell organisierte Dateien** siehe auch „RECORDING MODE-Klausel").

Format 1 Gibt Datensätze fester Länge an, und zwar durch die Anzahl der Bytes eines Datensatzes.

Format 2 Gibt Datensätze variabler Länge an. Dabei muss die Größe des Datensatzes innerhalb eines angegebenen Längenbereichs liegen.

Format 3 Gibt Datensätze variabler Länge an. Dabei wird die minimale und maximale Anzahl der Bytes eines Datensatzes angegeben.

Format 1

```
RECORD CONTAINS ganzzahl-1 CHARACTERS
```

Syntaxregeln

1. Die Länge eines jeden Datensatzes ist durch seine Datensatzerklärung genau festgelegt. Die Längenangabe in der RECORD-Klausel bleibt insofern außer Betracht.
2. Die Anzahl der Bytes jeder Datensatzerklärung für die Datei muss gleich ganzzahl-1 sein.
3. ganzzahl-1 muss mindestens 1 und darf höchstens 32767 sein. Bei Verwendung der RECORD-Klausel in einer Sortierdateierklärung beträgt das Maximum von ganzzahl-1 32755 minus Sortierschlüssellänge.

Format 2

```
RECORD IS VARYING IN SIZE [[FROM ganzzahl-2] [TO g anzzahl-3] CHARACTERS]
    [DEPENDING ON datenname-1]
```

Syntaxregeln

1. In keiner Datensatzerklärung für die Datei darf die in ganzzahl-2 angegebene Länge unterschritten und die in ganzzahl-3 angegebene Länge überschritten werden.
2. ganzzahl-3 muss größer sein als ganzzahl-2.
3. ganzzahl-2 muss größer oder gleich 0 sein, ganzzahl-3 darf höchstens 32763 sein. Bei Verwendung der RECORD-Klausel in einer Sortierdateierklärung beträgt das Maximum von ganzzahl-3 32751 minus Sortierschlüssellänge.
4. datenname-1 muss als vorzeichenloses ganzzahliges Datenelement der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION beschrieben sein.
5. Dieses Format darf nicht angegeben werden für **relativ organisierte Dateien**, die mit der DVS-Zugriffsmethode UPAM verarbeitet werden sollen (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

Allgemeine Regeln

1. Ist ganzzahl-2 nicht angegeben, dann wird angenommen, dass die Länge des kürzesten Datensatzes 1 ist. Für jede SORT- bzw. MERGE-Anweisung muss der Sortier schlüssel vollständig innerhalb dieser Minimallänge liegen.
2. Ist ganzzahl-3 nicht angegeben, dann wird die Länge des längsten Datensatzes der für diese Datei beschriebenen Datensatzerklärungen angenommen. Mit dieser Zahl wird auch die Blockgröße ermittelt.
3. Ist datenname-1 angegeben, dann muss die Anzahl der Bytes des Datensatzes nach datenname-1 übertragen werden, bevor eine RELEASE-, REWRITE- oder WRITE-Anweisung für diese Datei ausgeführt wird.

4. Ist datenname-1 angegeben, dann ändert sich sein Inhalt nicht, wenn eine RELEASE-, REWRITE- oder WRITE-Anweisung ausgeführt wird oder eine READ- oder RETURN-Anweisung nicht erfolgreich war.
5. Während der Ausführung einer RELEASE-, REWRITE- oder WRITE-Anweisung wird die Datensatzlänge wie folgt festgelegt:
   a. Ist datenname-1 angegeben: durch den Inhalt von datenname-1
   b. Ist datenname-1 nicht angegeben und enthält die Datensatzerklärung keine OCCURS-Klausel mit DEPENDING ON-Angabe: durch die Anzahl der Bytes des Datensatzes.
   c. Ist datenname-1 nicht angegeben und enthält die Datensatzerklärung eine OCCURS-Klausel mit DEPENDING ON-Angabe: durch den festen Teil der Daten satzerklärung, d.h. durch alle Datenerklärungen ohne DEPENDING ON-Angabe, und durch die Anzahl der Wiederholungen der Tabellenelemente während der Ausführung.
6. Ist datenname-1 angegeben und eine READ- oder RETURN-Anweisung erfolgreich ausgeführt, enthält datenname-1 die Anzahl der Bytes des gerade gelesenen Datensatzes. Diese Zahl ist jedoch nicht größer als ganzzahl-3. Sie kann aber größer sein als die größte Datensatzerklärung.
7. Ist in einer READ- oder RETURN-Anweisung INTO angegeben, wird die Anzahl der Bytes im aktuellen Datensatz, der im impliziten MOVE das Sendefeld ist, wie folgt festgelegt:
   a. Ist datenname-1 angegeben: durch den Inhalt von datenname-1
   b. Ist datenname-1 nicht angegeben: durch den Wert, der übertragen würde, wenn datenname-1 angegeben wäre.
8. Ist die ermittelte Anzahl von Bytes 0, so ist der Datensatz ein null-längiges Datenfeld.

Format 3

```
RECORD CONTAINS ganzzahl-4 TO ganzzahl-5 CHARACTERS
```

Syntaxregeln

1. ganzzahl-4 beschreibt die Anzahl der Bytes des kürzesten Datensatzes, ganzzahl-5 die des längsten.
2. In keiner Datensatzerklärung für die Datei darf die in ganzzahl-4 angegebene Länge unterschritten und die in ganzzahl-5 angegebene Länge überschritten werden.
3. ganzzahl-5 muss größer sein als ganzzahl-4.
4. ganzzahl-4 muss größer oder gleich 0 sein, ganzzahl-5 darf höchstens 32763 sein. Bei Verwendung der RECORD-Klausel in einer Sortierdateierklärung beträgt das Maximum von ganzzahl-5 32751 minus Sortierschlüssellänge.
5. Dieses Format darf nicht angegeben werden für **relativ organisierte Dateien**, die mit der DVS-Zugriffsmethode UPAM verarbeitet werden sollen (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

Allgemeine Regeln für alle Formate

1. Die Länge jedes Datensatzes ist durch seine Datensatzerklärung genau festgelegt. Ist die RECORD-Klausel angegeben, wird die Satzlänge mit den Angaben in der Klausel verglichen (für sequenziell organisierte Dateien siehe auch „RECORDING MODE Klausel“).
2. Die Länge eines Datensatzes wird bestimmt durch die Summe der Bytes aller Datenelemente und der vom Compiler erzeugten Füllfeld-Bytes. Enthält der Datensatz eine Tabelle, ist die minimale bzw. maximale Anzahl der Tabellenelemente in der Längenberechnung berücksichtigt. Weitere Angaben siehe Abschnitt „SYNCHRONIZED-Klausel", Abschnitt „USAGE-Klausel" und „Ausrichtung von Daten“ ("Herstellerabhängige Darstellung und Ausrichtung von Daten").
3. Ist die RECORD-Klausel nicht angegeben, ergibt sich die minimale bzw. maximale Satzlänge aus den Angaben der entsprechenden Datensatzerklärung. Enthält eine Datensatzerklärung eine OCCURS-Klausel mit DEPENDING-Angabe, wird für diese als minimale Satzlänge der Wert 1 angenommen.

4. Ist für eine *externe* Datei die RECORD-Klausel angegeben, muss in allen Programmen, die diese externe Datei beschreiben, eine RECORD-Klausel vom gleichen Format angegeben sein, wobei die aus den Angaben in der RECORD-Klausel bzw. den entsprechenden Datensatzbeschreibungen errechnete minimale und maximale Satzlänge übereinstimmen muss.
5. Folgende Tabelle zeigt, welche Angaben in den Formaten der RECORD-Klausel zur Berechnung der minimalen und maximalen Datensatzlänge maßgeblich sind.

	Format 1	**Format 2**	**Format 3**	**keine RECORD-Klausel**
minimale Datensatzlänge	längster Datensatz der Datensatzerklärung	ganzzahl-2; falls nicht angegeben: 1	ganzzahl-4	kürzester Datensatz der Datensatzerklärung; bei OCCURS DEPENDING: 1
maximale Datensatzlänge	wie oben	ganzzahl-3; falls nicht angegeben: längster Datensatz der Datensatz erklärung	ganzzahl-5	längster Datensatz der Datensatzerklärung

Tabelle 12: Angaben in der RECORD-Klausel

6. Die angegebenen Höchstgrenzen vermindern sich bei **relativ organisierten Dateien** um 8 Bytes.
7. Ist die ermittelte Anzahl von Bytes 0, so ist der Datensatz ein null-längiges Datenfeld.

7.2.11 RECORDING MODE-Klausel

Funktion

Die RECORDING MODE-Klausel gibt an, dass das Format der logischen Datensätze der Datei „undefiniert“ ist.

Format

```
RECORDING MODE IS U
```

Syntaxregel

1. Die Angabe U bedeutet, dass die Datei eine beliebige Kombination von festen oder variablen Datensätzen enthalten kann. Datensätze im U-Modus können nicht geblockt werden, und es geht ihnen kein Steuerfeld (Satzlängenfeld, SLF) voraus.
   Ist RECORDING MODE IS U angegeben, erübrigt sich die Angabe der BLOCK CONTAINS-Klausel.

Allgemeine Regeln

1. Die Datensatzformate „F“ (Datensätze fester Länge) und „V“ (Datensätze variabler Länge) werden durch die RECORD-Klausel bestimmt:

   Feste Satzlänge durch eine RECORD-Klausel vom Format 1,

   Variable Satzlänge durch eine RECORD-Klausel vom Format 2.
2. Bezieht sich die RECORDING MODE-Klausel auf eine *externe* Datei, muss in allen Programmen, die diese externe Datei beschreiben, eine gleichlautende

7.2.12 VALUE OF-Klausel

Funktion

Die VALUE OF-Klausel vereinbart die Beschreibung von Datenfeldern eines Dateikennsatzes.

Format

```
VALUE OF {{IDENTIFICATION | ID} IS {datenname-1 | literal-1}} ...
```

Die VALUE OF-Klausel wird vom Compiler als Kommentar behandelt.

7.3 Datenerklärung

In diesem Kapitel werden folgende Themen behandelt:

7.3.1 Allgemeine Beschreibung

Organisation der Einträge der DATA DIVISION

Datenerklärung ist der allgemeine Ausdruck für die Beschreibung jedes einzelnen Datenfeldes in der DATA DIVISION; eine solche Erklärung setzt sich zusammen aus der Stufennummer und falls notwendig gefolgt von einem Datennamen, und mehreren Datenklauseln.

Die **Datensatzerklärung** dient zur Beschreibung aller Datenerklärungen, die mit einem bestimmten Datensatz verbunden sind, d.h. die Datensatzerklärung beschreibt alle Eigenschaften dieses Satzes. Die Typerklärung ist eine spezielle Form der Datensatzerklärung. Sie dient als Vorlage, die in anderen Datensatzerklärungen wieder verwendet werden kann.

Stufennummern werden zur Strukturierung eines logischen Datensatzes verwendet, um Datenbezugnahmen auf Datensatzunterteilungen zu ermöglichen. Ist einmal eine Unterteilung angegeben, so kann sie weiter unterteilt werden, um noch detailliertere Datenbezugnahmen zu erlauben.

Tabelle 13 zeigt die erlaubten Stufennummern und die damit zusammenhängenden Einträge in der DATA DIVISION.

Stufennummer	Verwendung
01	Datensatzerklärungen
02 - 49	Datenerklärungen, eine Unterteilung eines Datensatzes beschreibend
77	Beschreibung von unabhängigen oder nicht strukturabhängigen Datenfeldern, die nicht Teile anderer Datenfelder und selbst nicht unterteilt sein dürfen
66	Datenelemente oder Datengruppen, beschrieben durch die RENAMES-Klausel, zum Zwecke der Umgruppierung von Datenfeldern (siehe „RENAMES-Klausel")
88	Einträge von Bedingungsnamen, um anzugeben, dass Bedingungsnamen mit bestimmten Werten einer Bedingungsvariablen verknüpft sind (siehe „VALUE-Klausel“, Format 2, "VALUE-Klausel")

Tabelle 13: Bedeutung der Stufennummern

Aufeinanderfolgende Datenerklärungen können dasselbe Format wie der erste solche Eintrag haben oder können, der Stufennummer entsprechend, eingerückt sein. Die Einrückung ist vorteilhaft für Dokumentationszwecke, hat aber keine Auswirkung auf die Übersetzung.

Mehrfach definierte Datensatzerklärungen der Stufe 01 und Datenfeldbeschreibungen der Stufe 77 werden nicht als Fehler gemeldet, wenn diese nicht in der PROCEDURE DIVISION benutzt werden. Mehrfach definierte Typerklärungen werden nicht als Fehler gemeldet, wenn diese nicht in einer TYPE-Klausel oder in der USAGE-Klausel für typbezogene Datenzeiger verwendet werden.

Die Strukturierung eines logischen Datensatzes erfolgt nach dem Stufenkonzept. Dieses Konzept entsteht aus der Notwendigkeit, Teile eines Datensatzes zu benennen, um darauf zugreifen zu können. Ist einmal eine Unterteilung spezifiziert, so kann sie weiter unterteilt werden, um detailliertere Datenbezüge zu erlauben.

Innerhalb der REPORT SECTION ist eine Leiste gleichbedeutend mit einem Datensatz in anderen Kapiteln der DATA DIVISION. Die **Leistenerklärung** dient zur Beschreibung aller Datenerklärungen, die mit einer bestimmten Leiste verbunden sind. Innerhalb einer Leistenerklärung unterscheidet man zwischen der ersten Datenerklärung und den nachfolgenden Datenerklärungen (siehe unter Kapitel „Listenprogramm (Report-Writer)").

Nicht weiter unterteilte Bestandteile eines Datensatzes werden **Datenelemente** genannt; ein Datensatz besteht also aus einer Folge von Datenelementen oder ist selbst ein Datenelement.

Die Länge eines Datenelementes darf 131071 Zeichen nicht überschreiten.

Um eine Anzahl von Datenelementen auf einmal ansprechen zu können, werden die Datenelemente in **Gruppen** oder **Datengruppen** zusammengefasst. Jede Gruppe besteht aus einer benannten Folge von einem oder mehreren Datenelementen. Gruppen wiederum können zusammengefasst werden zu Gruppen von zwei oder mehr Gruppen usw. Infolgedessen kann ein Datenelement zu mehr als einer Gruppe gehören.

Das Wort **Datenfeld** wird in den Fällen verwendet, in denen die Unterscheidung von Datenelementen oder Datengruppen belanglos ist.

Datenfelder, die in keiner hierarchischen Beziehung zueinander stehen, werden als unabhängige Datenelemente in Verbindung mit der Stufennummer 77 beschrieben.

Neben den Pflichtklauseln können weitere Datenklauseln zur Datenerklärung verwendet werden. Ihre Anwendung ist in dem Abschnitt „Klauseln für die Datenerklärung" beschrieben.

7.3.2 Formate der Datenerklärung

Funktion

Eine Datenerklärung beschreibt die Eigenschaften eines einzelnen Datenfeldes.

Format 1

```
stufennummer    [datenname | FILLER]

                [REDEFINES–Klausel]
                [ANY LENGTH–Klausel]
                [BASED–Klausel]
                [BLANK WHEN ZERO–Klausel]
                [DYNAMIC–Klausel]
                [EXTERNAL–Klausel]
                [GLOBAL–Klausel]
                [GROUP-USAGE–Klausel]
                [JUSTIFIED–Klausel]
                [OCCURS–Klausel]
                [PICTURE–Klausel]
                [SIGN–Klausel]
                [SYNCHRONIZED–Klausel]
                [TYPE–Klausel]
                [TYPEDEF–Klausel]
                [USAGE–Klausel]
                [VALUE–Klausel]
```

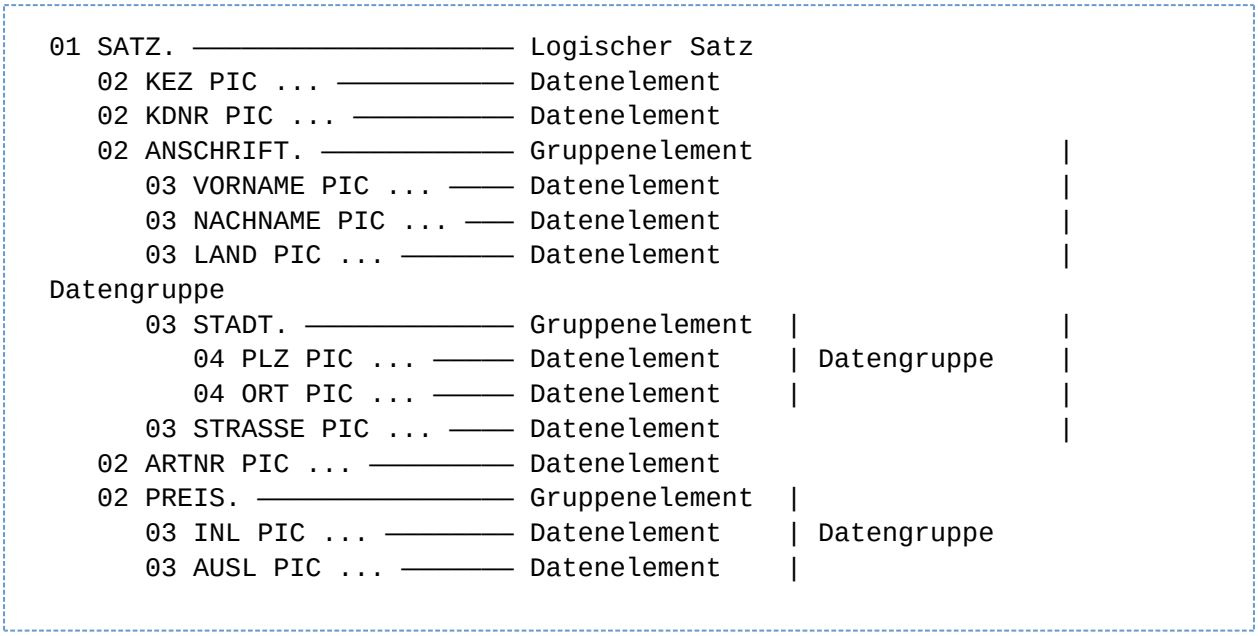
Syntaxregeln

1. stufennummer muss eine Nummer von 01 bis 49 oder die Nummer 77 sein.
   Erklärungen der Stufe 77 beschreiben Datenfelder, die in keiner hierarchischen Beziehung zueinander stehen und die nicht weiter unterteilt sind.
2. Die Klauseln dürfen in beliebiger Reihenfolge geschrieben werden. Eine Ausnahme bilden die FILLER- und datenname-Angabe sowie die REDEFINES-Klausel und die TYPEDEF-Klausel. Die FILLER- oder datenname-Angabe muss unmittelbar auf die Stufennummer folgen. Wenn die REDEFINES-Klausel angegeben ist, muss sie unmittelbar als erste Klausel vor allen anderen Klauseln stehen. Die TYPEDEF-Klausel darf nicht zusammen mit der REDEFINES-Klausel angegeben werden. Wenn die TYPEDEF-Klausel angegeben ist, muss sie unmittelbar auf die Stufennummer und datenname-Angabe folgen.
3. Die PICTURE-Klausel darf für Datenelemente, die mit USAGE COMP-1, COMP-2, INDEX, OBJECTREFERENCE, POINTER oder PROGRAM-POINTER definiert sind, nicht angegeben werden. Für alle anderen Datenelemente muss die PICTURE-Klausel angegeben werden, mit folgender Ausnahme:
   Die PICTURE-Klausel darf für ein Datenelement fehlen, sofern eine VALUE-Klausel vom Format 1 mit einem alphanumerischen oder nationalen Literal angegeben ist und es nicht von einem Eintrag mit Stufennummer 88 gefolgt wird.
   Der Compiler nimmt dann eine PICTURE-Klausel PICTURE X(länge) oder PICTURE N(länge) an, wobei länge die Anzahl der durch das Literal dargestellten Zeichen ist.
4. Die OCCURS-Klausel darf nicht in Verbindung mit den Stufennummern 01, 66, 77 und 88 verwendet werden.
5. Ist die ANY LENGTH-Klausel angegeben, so darf außer Stufennummer und datenname nur noch die PICTURE-Klausel und die USAGE-Klausel angegeben werden.
6. Folgende Angaben müssen in jeder 77-Stufenerklärung vorhanden sein:
   - Datenname

- PICTURE-Klausel
  oder
- USAGE-Klausel mit Angabe INDEX, COMPUTATIONAL-1, COMPUTATIONAL-2, OBJECT REFERENCE, POINTER oder PROGRAM-POINTER
  oder
- eine VALUE-Klausel mit einem alphanumerischen oder nationalen Literal oder eine TYPE-Klausel

Beispiel 7-1

für die Struktur eines Datensatzes mit der Beschreibung von Datengruppen



Das Gruppenelement enthält keine Angaben über Datenklasse und Feldgröße. Es können aber auf den Gruppenelement-Namen noch Definitionen (z.B. REDEFINES, OCCURS) folgen. Der Eintrag wird mit einem Punkt abgeschlossen.

Format 2

```
66 datenname-1 RENAMES datenname-2 [{THROUGH | THRU} datenname-3].
```

Syntax- und Allgemeine Regeln siehe „RENAMES-Klausel".

Format 3

```
88 bedingungsname {VALUE | VALUES} {IS | ARE} {literal-1 [{THROUGH | THRU} literal-2]}...
```

Syntax- und Allgemeine Regeln siehe Format 2 im Abschnitt „VALUE-Klausel".

7.3.3 Stufennummer

Funktion

Die Stufennummer zeigt die Hierarchie der Daten innerhalb eines logischen Satzes. Zusätzlich dient sie zur Kennzeichnung von Erklärungen der Datenfelder in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION, der Bedingungsnamen und der Datenfelder in der RENAMES-Klausel.

Format

```
stufennummer
```

Syntaxregeln

1. Die Stufennummer ist ein spezielles numerisches Literal und besteht aus 1 bis 2 Ziffern. Eine Stufennummer kleiner 10 wird entweder als einstellige Zahl oder mit führender Null geschrieben.
2. Datenerklärungen eines FD- oder SD-Eintrags müssen Stufennummern mit den Werten von 01 bis 49, 66 oder 88 haben.
3. Datenerklärungen eines RD-Eintrags dürfen nur die Stufennummer 01 und 02 haben.
4. In jeder Datenerklärung muss eine Stufennummer als erstes Element angegeben werden.

Allgemeine Regeln

1. Die Stufennummer 01 kennzeichnet die erste Erklärung einer jeden Satzbeschreibung oder einer Leiste.
2. Bestimmten Erklärungen, für die es kein wirkliches Stufenkonzept gibt, werden spezielle Stufennummern zugewiesen. Sie sind im Folgenden beschrieben:

   Die Stufennummer 66 dient zur Kennzeichnung von Neubenennungs-Einträgen und kann nur im Zusammenhang mit der RENAMES-Klausel verwendet werden.

   Die Stufennummer 77 dient zur Kennzeichnung von strukturunabhängigen Datenfeldern in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION und kann nur so verwendet werden, wie es unter „77-Stufenerklärung“ beschrieben ist.

   Die Stufennummer 88 bezieht sich auf Erklärungen von Bedingungsnamen, die einer Bedingungsvariablen zugeordnet sind, und kann nur so verwendet werden, wie es im Format 2 der VALUE-Klausel beschrieben ist.
3. Mehrere Erklärungen der Stufennummer 01, die einer gegebenen Stufenbezeichnung außer RD untergeordnet sind, stellen implizit eine Neubelegung des gleichen Bereichs dar.

Beispiel 7-2

```
01  ADRESSE.
    02  NAME.
        03  VORNAME            PIC X(18).
        03  NACHNAME           PIC X(20).
    02  WOHNUNG.
        03  POSTLEITZAHL.
            04  ZIFFER-1       PIC 9.
            04  ZIFFER-2       PIC 9.
            04  ZIFFER-3       PIC 9.
            04  ZIFFER-4       PIC 9.
            04  ZIFFER-5       PIC 9.
        03  ORT                PIC X(19).
        03  STRASSE            PIC X(16).
        03  HAUSNUMMER         PIC XXX.
```

Mit der Anweisung

```
MOVE ADRESSE TO...
```

wird die gesamte Gruppe übertragen.

Mit der Anweisung

```
MOVE NAME TO...
```

wird Vor- und Nachname übertragen etc.

7.3.4 Klauseln für die Datenerklärung

7.3.5 ANY LENGTH-Klausel

Funktion

Die ANY LENGTH-Klausel bewirkt, dass die Länge eines formalen Parameters einer Methode durch die Länge des Argumentes bestimmt wird.

Format

```
ANY LENGTH
```

Syntaxregeln

1. Die ANY LENGTH-Klausel darf nur in Datenerklärungen mit der Stufennummer 01 oder 77 in der LINKAGE SECTION einer Methode angegeben werden.
2. Die Datenerklärung muss eine PICTURE-Klausel enthalten, deren Maskenzeichenfolge aus einem einzelnen "X" oder "N" besteht.

Allgemeine Regeln

1. Durch die ANY LENGTH-Klausel wird das Datenelement als variabel lang definiert. Die aktuelle Länge des Datenelementes entspricht der Länge des Argumentes bzw. Rückgabeelementes des Aufrufers.
2. Ist das dem Datenelement entsprechende Argument bzw. Rückgabeelement des Aufrufers ein null-längiges Datenfeld, so ist auch das Datenelement ein null-längiges Datenfeld.

Beispiel 7-3

```
METHOD-ID. SEARCH-CHAR.
DATA DIVISION.
LINKAGE SECTION.
01    PAR1    PIC X ANY LENGTH.
01    PAR2    PIC X.
01    IDX     PIC 9(9) USAGE COMP-5.
PROCEDURE DIVISION USING PAR1 PAR2 RETURNING IDX.
  PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > FUNCTION LENGTH(PAR1)
    IF PAR1(IDX:1) = PAR2
        EXIT METHOD           *> Zeichen gefunden ---->>>>
     END-IF
  END-PERFORM
  MOVE 0 TO IDX.            *> Zeichen nicht gefunden
  EXIT METHOD.
END METHOD SEARCH-CHAR.
```

Ausschnitt eines Programms, aus dem die Methode SEARCH-CHAR aufgerufen wird:

```
...
01 OBJ      USAGE IS OBJECT REFERENCE.
01 I        PIC 9(9) USAGE IS COMP-5.
01 F1       PIC x(50).
01 F2
   02 ELEM  OCCURS 100 DEPENDING ON I.
...
INVOKE OBJ "SEARCH-CHAR" USING F1,"X" RETURNING I. ————————— (1)
INVOKE OBJ "SEARCH-CHAR" USING F2,"Y" RETURNING I. ————————— (2)
...
```

(1) Die aktuelle Länge des Parameters PAR1 beträgt 50. Die Länge muss vom Anwender nicht übergeben werden.

(2) Die aktuelle Länge des Parameters PAR1 ist erst dynamisch zur Ablaufzeit bekannt.

Änderungen des RETURNING-Parameters IDX in der Methode SEARCH-VAR wirken sich auf die Länge des aktuellen Parameters F2 erst nach Rückkehr aus der Methode SEARCH-CHAR aus.

Die unterschiedliche Länge der aktuellen Parameter F1 bzw. F2 stellt wegen der ANY LENGTH-Klausel **keinen** Verstoß gegen Conformance-Regeln dar.

7.3.6 BASED-Klausel

Die BASED-Klausel kennzeichnet einen Datensatz als reine Schablone, dem kein Speicherplatz zugeordnet ist.

Format

BASED

Syntaxregeln

1. Die BASED-Klausel darf nur für Datendefinitionen in der LINKAGE SECTION mit der Stufennummer 01 oder 77 angegeben werden.
2. Die Angabe von USAGE OBJECT REFERENCE, POINTER und PROGRAM-POINTER ist nicht erlaubt. Auch Datenbeschreibungen, die der Datenbeschreibung mit der BASED-Angabe untergeordnet sind, dürfen USAGE OBJECT REFERENCE, POINTER und PROGRAM-POINTER nicht enthalten.
3. Die Angaben BASED und REDEFINES dürfen nicht zusammen verwendet werden.

Allgemeine Regel

1. Die Anfangsadresse dieses Datensatzes wird auf die vordefinierte Adresse NULL gesetzt. Eine Adresse wird erst dann zugeordnet, wenn sie explizit gesetzt wird.

Beispiel 7-4

```
WORKING-STORAGE SECTION.
    01 POINTER-A USAGE POINTER.
LINKAGE SECTION.
    01 DSECT-A BASED.
    02 ...
PROCEDURE DIVISION.
...
    SET ADDRESS OF DSECT-A TO POINTER-A.
```

7.3.7 BLANK WHEN ZERO-Klausel

Funktion

Die BLANK WHEN ZERO-Klausel bewirkt, dass ein Datenfeld immer mit Leerzeichen aufzufüllen ist, wenn sein Inhalt den Wert Null enthält.

Format

```
BLANK WHEN ZERO
```

Syntaxregeln

1. Die BLANK WHEN ZERO-Klausel darf nur bei Datenelementen, die numerisch oder numerisch-druckaufbereitet sind, angegeben werden.
2. Die numerischen oder numerisch druckaufbereiteten Datenelemente, für die BLANK WHEN ZERO angegeben ist, müssen explizit oder implizit als USAGE DISPLAY definiert sein.

Allgemeine Regeln

1. Wird die BLANK WHEN ZERO-Klausel benutzt, so enthält das Datenfeld nur Leerzeichen, wenn sein Wert Null ist.
2. Wird die BLANK WHEN ZERO-Klausel für numerische Datenfelder benutzt, wird die Kategorie des Datenfeldes als numerisch-druckaufbereitet betrachtet.
3. Wird die BLANK WHEN ZERO-Klausel und die PICTURE-Klausel mit der Angabe Stern (*) als Nullunterdrückungszeichen zusammen in einem Datenerklärungseintrag verwendet, so hebt die Druckaufbereitung durch Nullunterdrückung die Funktion der BLANK WHEN ZERO-Klausel auf (siehe „PICTURE-Klausel").

Beispiel 7-5

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BLWHENZ.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 BETRAGSBEISPIEL.
   02 BETRAG PICTURE $Z.99 BLANK WHEN ZERO.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    MOVE ZERO TO BETRAG.
    DISPLAY BETRAG UPON T.
    STOP RUN.
```

Wert von BETRAG nach der MOVE-Anweisung:

'BLANK' 'BLANK' 'BLANK' 'BLANK' 'BLANK' (5 Leerzeichen)

7.3.8 DYNAMIC-Klausel

Funktion

Die DYNAMIC-Klausel ermöglicht die dynamische Speicherbereitstellung in einem vom Benutzer definierten Umfang.

Format

```
01 datenname IS DYNAMIC.
```

(stufennummer und datenname sind nicht Teil der DYNAMIC-Klausel; sie werden hier nur zur Verdeutlichung angegeben.)

Syntaxregel

1. Die DYNAMIC-Klausel darf nur in Datensatzerklärungen der Stufe 01 der WORKING-STORAGE SECTION angegeben werden.
2. Ist für ein Datenfeld die DYNAMIC-Klausel angegeben, darf keine andere Klausel für dieses Datenfeld angegeben werden.

Allgemeine Regel

1. Das Datenfeld, das mit der DYNAMIC-Klausel versehen ist, wird zur Ablaufzeit im Arbeitsspeicher angelegt und beginnt auf 4-Kbyte-Grenze.

7.3.9 Datenname- oder FILLER-Klausel

Funktion

Ein Datenname benennt die zu beschreibenden Daten. Das reservierte Wort FILLER bezeichnet Datenelemente oder Datengruppen eines logischen Satzes, die nie angesprochen werden und deshalb nicht mit einem Namen versehen werden.

Format

```
stufennummer [datenname | FILLER]
```

(Die Stufennummer ist nicht ein Teil der Datenname- oder FILLER-Klausel; sie wird hier nur zur Verdeutlichung angegeben.)

Syntaxregeln

1. datenname muss nach den Regeln für Programmiererwörter gebildet werden.
2. In der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION muss FILLER oder datenname das erste Wort nach der Stufennummer sein.
3. Das reservierte Wort FILLER wird verwendet, um Datenelemente oder Datengruppen, die im Programm nie angesprochen werden und deshalb nicht mit einem Datennamen versehen werden müssen, zu bezeichnen. Ein FILLER-Datenfeld kann nicht direkt angesprochen werden.
4. Fehlt die Angabe datenname bzw. FILLER, wird FILLER angenommen.

Allgemeine Regel

1. Alle angesprochenen Datenerklärungseinträge mit den Stufennummern 77 und 01 in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION müssen - wenn sie angesprochen werden sollen - mit eindeutigen Datennamen versehen werden, da sie nicht gekennzeichnet werden können. Ein untergeordneter Datenname muss nicht eindeutig sein, wenn er durch Kennzeichnung eindeutig gemacht werden kann.

Beispiel 7-6

```
01 DATENSATZ.
   02 NUMMER-EINS      PICTURE 9(8).
   02 NUMMER-ZWEI      PICTURE 9(12).
   02 FILLER           PICTURE X(60).
```

Hier wird ein Datensatz mit dem Datennamen DATENSATZ und die ersten zwei Felder mit den Datennamen NUMMER-EINS und NUMMER-ZWEI benannt. Da auf das dritte Feld im Programm nicht Bezug genommen wird, folgt der Stufennummer das reservierte Wort FILLER.

7.3.10 EXTERNAL-Klausel

Funktion

Mit der EXTERNAL-Klausel kann ein Datensatz als extern definiert werden. Auf externe Datensätze kann von jedem Programm, in dem der Datensatz beschrieben ist, zugegriffen werden.

Format für Datenerklärungen

```
IS EXTERNAL
```

Syntaxregeln

1. Die EXTERNAL-Klausel darf nur in der Datensatzbeschreibung der WORKING-STORAGE SECTION stehen.
2. Im selben Programm darf ein als extern definierter Datenname nicht noch einmal definiert werden.
3. Die EXTERNAL-Klausel darf sich nur auf eine Datenbeschreibung der Stufennummer 01 beziehen.
4. Namen von externen Datensätzen dürfen maximal 30 Zeichen lang sein.
5. Als Namen für externe Datensätze dürfen nicht verwendet werden:
   - Namen von externen Dateien aus anderen Übersetzungseinheiten in der Ablaufeinheit
   - PROGRAM-ID-Namen der Ablaufeinheit, ausgenommen Namen von inneren Programmen eines geschachtelten Programms,
   - Namen, die in der ENTRY-Anweisung als Einsprungpunkte verwendet werden,
   - Namen, die eine Schnittstelle benennen (LZS-Namen u.a.)
6. Die EXTERNAL-Klausel und die REDEFINES-Klausel dürfen nicht in ein und derselben Datenbeschreibung verwendet werden.
7. Die EXTERNAL-Klausel und die TYPEDEF-Klausel dürfen nicht in ein und derselben Datenbeschreibung verwendet werden.
8. Die EXTERNAL-Klausel darf nicht für ein Datenfeld der Klassen objekt, zeiger oder stark typisiert angegeben werden.

Allgemeine Regeln

1. Ein externer Datensatz, der in mehreren Programmen einer Ablaufeinheit beschrieben ist, muss in diesen Programmen den gleichen Namen haben und auch in der Länge gleich sein. Der Compiler lässt auch unterschiedliche Längendefinitionen zu; davon sollte bei Verwendung von CALL bezeichner allerdings nicht Gebrauch gemacht werden.
2. Ist ein Datensatz als extern definiert, so ist der zugehörige Datenname nicht implizit ein globaler Name.
3. VALUE-Klauseln, die in einer Datenbeschreibung angegeben werden, die einer Datenbeschreibung mit EXTERNAL-Klausel zu- oder untergeordnet ist, haben nur bei der Ausführung einer INITIALIZE-Anweisung eine Bedeutung.
   Sind einem externen Datensatz, der in mehreren Programmen einer Ablaufeinheit beschrieben ist, in mehr als einem Programm VALUE-Klauseln zu- oder untergeordnet, dann müssen diese VALUE-Klauseln identisch sein. Der Compiler lässt auch unterschiedliche VALUE-Klauseln zu.

Zusatzregeln, abhängig vom Modulformat

Wird das Format *OMF generiert (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]), dann gilt für Namen eines externen Datensatzes:

1. Das achte Zeichen darf kein Bindestrich sein.
2. Es werden nur die ersten 8 Zeichen des Namens zur Identifizierung verwendet. Deshalb sollten diese Zeichen eindeutig sein für jeden externen Namen in der Ablaufeinheit.

7.3.11 GLOBAL-Klausel

Funktion

Die GLOBAL-Klausel kann nur innerhalb eines geschachtelten Programms verwendet werden. Sie legt fest, dass ein Datenname global ist. Auf einen globalen Namen kann das Programm, das ihn vereinbart, und jedes in diesem Programm direkt oder indirekt enthaltene Programm zugreifen.

Format

```
IS GLOBAL
```

Syntaxregeln

1. Die GLOBAL-Klausel darf nur in Einträgen mit Stufennummer 01 angegeben werden:
   - für Datenerklärungen in der WORKING-STORAGE oder der FILE SECTION.
   - für Typdeklarationen in der WORKING-STORAGE, LOCAL-STORAGE, LINKAGE oder FILE SECTION.
2. Sind zwei Datenfelder in derselben DATA DIVISION mit demselben Namen definiert, darf in keiner der entsprechenden Datenerklärungen die GLOBAL-Klausel angegeben werden.
3. Die GLOBAL-Klausel darf nicht in einer Methoden-, Factory- oder Objekt-Definition angegeben werden.

Allgemeine Regeln

1. Ein Datenname, dessen Beschreibung eine GLOBAL-Klausel enthält, ist ein globaler Name. Alle einem globalen Namen untergeordneten Datennamen sowie alle Bedingungsnamen, die mit einem globale Namen in Verbindung stehen, sind globale Namen.
2. Auf einen globalen Namen darf jedes Programm zugreifen, das in dem Programm enthalten ist, das den globalen Namen beschreibt. Der globale Name braucht in dem Programm, das ihn referenziert, nicht nochmals beschrieben zu werden. Bei Referenzen auf gleiche Namen haben die lokalen Namen Vorrang (siehe „Bestimmung des gültigen Namens").
3. Enthält eine Datenerklärung neben der GLOBAL-Klausel auch die REDEFINES-Klausel, so wird dadurch das redefinierte Datenfeld nicht implizit global.
4. Enthält ein globales Datenfeld eine variabel lange Tabelle, so muss das entsprechende DEPENDING ON-Element in derselben DATA DIVISION und ebenfalls als global beschrieben sein.
5. Die Datennamen in der CONFIGURATION SECTION sind stets implizit global.
6. Der Index einer indizierten Tabelle, die einem globalen Datenfeld zugeordnet ist, ist ebenfalls global.

7.3.12 GROUP-USAGE-Klausel

Funktion

Die GROUP-USAGE-Klausel erlaubt es, eine Datengruppe wie ein nationales Datenelement zu behandeln.

Format

```
GROUP-USAGE IS NATIONAL
```

Syntaxregeln

1. Die GROUP-USAGE-Klausel darf nur für Datengruppen angegeben werden. Sie dürfen nicht stark typisiert sein.
2. Die Angabe impliziert USAGE NATIONAL für den Dateneintrag. Eine USAGE-Klausel darf nicht (zusätzlich) angegeben werden.
3. Alle untergeordneten Datenelemente müssen explizit oder implizit mit USAGE NATIONAL beschrieben sein. Alle untergeordneten Datengruppen müssen explizit oder implizit mit GROUP-USAGE NATIONAL beschrieben sein.
4. Die GROUP-USAGE-Klausel darf nicht in Datenerklärungen mit der Stufennummer 01 in der FILE SECTION angegeben werden.

Allgemeine Regeln

1. Die Klausel macht die Gruppe zu einer nationalen Datengruppe. Ihre Klasse und Kategorie sind national.
2. Wenn im Einzelfall nicht anders angegeben, wird eine nationale Gruppe immer wie ein Datenelement mit USAGE NATIONAL und PICTURE N(m) behandelt, wobei m die Länge der Gruppe ist.
3. Für alle Untergruppen wird GROUP-USAGE NATIONAL implizit angenommen.
4. Ohne explizite oder implizite Angabe einer GROUP-USAGE-Klausel ist eine nicht stark typisierte Gruppe immer eine alphanumerische Gruppe.

> **i** Die GROUP-USAGE-Klausel ist erforderlich, wenn eine Gruppe, die nur nationale Datenfelder enthält, auch als Gruppe wie ein nationales Datenfeld behandelt werden soll, z.B. beim Auffüllen mit Leerzeichen im Rahmen von Übertragungen.

7.3.13 JUSTIFIED-Klausel

Funktion

Die JUSTIFIED-Klausel wird verwendet, um nichtnumerische Daten innerhalb eines nichtnumerischen Empfangsfeldes in einer anderen Form als der standardmäßigen auszurichten.

Format

```
{JUSTIFIED | JUST}  RIGHT
```

Syntaxregeln

1. JUST ist die Abkürzung für JUSTIFIED.
2. Die JUSTIFIED-Klausel darf nur für Datenelemente angegeben werden.
3. Die JUSTIFIED-Klausel darf nur für alphabetische, alphanumerische oder nationale Datenfelder angegeben werden.
4. Die JUSTIFIED-Klausel darf nicht für Datenfelder mit der Stufennummer 66 oder 88 angegeben werden.
5. Die JUSTIFIED-Klausel darf nicht für ein Empfangsfeld einer STRING-Anweisung angegeben werden (siehe „STRING-Anweisung").

Allgemeine Regeln

1. Wird für das Empfangsfeld die JUSTIFIED-Klausel angegeben und ist das Sendefeld größer als das Empfangsfeld, erfolgt die Ausrichtung nach der am weitesten rechts stehenden Zeichenposition, und die am weitesten links stehenden Zeichen werden abgeschnitten.

   Wird für das Empfangsfeld die JUSTIFIED-Klausel angegeben und ist das Empfangsfeld größer als das Sendefeld, werden die Daten nach der am weitesten rechts stehenden Zeichenstelle ausgerichtet, die restlichen unbenutzten Zeichenstellen werden auf der linken Seite mit Leerzeichen entsprechend der Klasse des Empfangsfeldes aufgefüllt.
2. Wird die JUSTIFIED-Klausel nicht angegeben, gelten die Regeln zur Ausrichtung von Daten innerhalb eines Datenelements (siehe Abschnitt „Konzept der maschinenunabhängigen Datenbeschreibung").

Beispiel 7-7

Standardausrichtung (ohne JUSTIFIED):

Sendefeld kleiner Empfangsfeld

Sendefeld größer oder gleich Empfangsfeld

Ausrichtung, wenn die JUSTIFIED-Klausel angegeben ist:

Sendefeld kleiner Empfangsfeld

Sendefeld größer Empfangsfeld

7.3.14 OCCURS-Klausel

Funktion

Die OCCURS-Klausel wird benutzt, um Tabellen zu definieren. In der Klausel wird angegeben, wieviele Elemente die Tabelle haben soll, d.h. wie oft ein Feld wiederholt werden soll. Alle Elemente haben dasselbe Format. Die Größe der Tabelle kann variabel sein. Außerdem
können Indizes vereinbart werden.

Format 1	gibt die genaue Anzahl der Wiederholungen eines Datenfeldes an.
Format 2	gibt eine variable Wiederholungszahl eines Datenfeldes an, die zwischen einer **maximalen** und einer **minimalen** Tabellenelementnummer liegt. Die Angabe des Minimalwertes kann entfallen.

Format 1

```
OCCURS ganzzahl-2 TIMES
    [{ASCENDING | DESCENDING} KEY IS {datenname-2}... ] ...
    [INDEXED BY {index-1}...]
```

Format 2

```
OCCURS [ganzzahl-1 TO] ganzzahl-2 TIMES DEPENDING ON datenname-1
    [{ASCENDING | DESCENDING} KEY IS {datenname-2}... ] ...
    [INDEXED BY {index-1}...]
```

Syntaxregeln für beide Formate

1. Die OCCURS-Klausel darf in einer Datenerklärung nicht angegeben werden, falls diese
   a. eine Stufennummer 01, 66, 77 oder 88 hat, oder
   b. ein Feld mit variabler Länge beschreibt (die Länge eines Feldes ist variabel, wenn die Datenerklärung irgendeines ihm untergeordneten Feldes eine OCCURS-Klausel mit der DEPENDING-Angabe enthält).
2. datenname-1, datenname-2,... können mit OF oder IN gekennzeichnet sein (siehe „Kennzeichnung").
3. datenname-2 kann entweder der Datenname sein, auf den sich die OCCURS-Klausel bezieht (in diesem Fall muss er als erster angegeben sein), oder er muss der Datengruppe, auf die sich die OCCURS-Klausel bezieht, untergeordnet sein.
4. Stimmt datenname-2 nicht mit dem Datennamen überein, auf den sich die OCCURS-Klausel bezieht, dann darf datenname-2 nicht mit einer OCCURS-Klausel erklärt sein. Er darf auch nicht einem Eintrag untergeordnet sein, der eine andere OCCURS-Klausel enthält.
5. datenname-2 ist folgenden zusätzlichen Regeln unterworfen:
   a. Bis zu 12 Schlüsselfelder können für ein gegebenes Tabellenelement angegeben werden.
   b. Die Summe der Länge aller Schlüsselfelder, die mit einem Tabellenelement verknüpft sind, darf 256 Bytes nicht überschreiten.
   c. Die Schlüsselfelder dürfen nicht von der Klasse objekt oder zeiger sein.
6. Die OCCURS-Klausel darf nicht zusammen mit einer IDENTIFIED-Klausel oder in einem Eintrag, der direkt einem Eintrag mit einer IDENTIFIED-Klausel untergeordnet ist, angegeben werden.

Syntaxregeln für Format 2

7. ganzzahl-1 muss eine positive ganze Zahl oder 0 sein.
8. Werden ganzzahl-1 und ganzzahl-2 gemeinsam benutzt, muss ganzzahl-1 kleiner sein als ganzzahl-2.

9. datenname-1 muss als ganzzahliges numerisches Datenfeld beschrieben sein.
10. Wenn datenname-1 in demselben Datensatz auftritt wie die Tabelle, deren Wiederholungen er steuert, muss er vor dem variablen Teil dieses Datensatzes stehen. Das Datenfeld, das durch datenname-1 beschrieben ist, muss also vor dem Teil des Datensatzes liegen, der mit der OCCURS-Klausel mit DEPENDING ON-Angabe beschrieben ist.
11. Wird die OCCURS-Klausel in einer Datenbeschreibung einer Datensatzbeschreibung angegeben, die die EXTERNAL- oder GLOBAL-Klausel enthält, so muss sich datenname-1, falls angegeben, auf ein Datenfeld beziehen, das in derselben DATA DIVISION als EXTERNAL bzw. GLOBAL beschrieben ist.
12. Einer Datenerklärung, die eine OCCURS-Klausel mit DEPENDING ON-Angabe enthält, dürfen innerhalb dieser Datensatzbeschreibung nur Datenerklärungen folgen, die ihr untergeordnet sind. Der COBOL2000-Compiler lässt auch hierarchisch unabhängige Datenerklärungen zu (siehe Regel 15).
13. Die OCCURS-Klausel darf nicht in der Satzbeschreibung einer XML-organisierten Datei angegeben werden.
14. Ein Eintrag, der eine OCCURS-Klausel mit DEPENDING ON-Angabe enthält oder dem ein Eintrag mit einer OCCURS-Klausel mit DEPENDING ON-Angabe untergeordnet ist, darf keine REDEFINES-Klausel enthalten.

Allgemeine Regeln für beide Formate

1. ganzzahl-2 stellt die Anzahl der Wiederholungen dar.
2. Außer der OCCURS-Klausel selber gelten alle Datenbeschreibungsklauseln des Datenfeldes, das diese OCCURS-Klausel enthält, für jede Wiederholung des beschriebenen Datenfeldes.
3. Der Datenname, auf den sich die OCCURS-Klausel bezieht, muss immer dann indiziert oder subskribiert werden, wenn darauf in einer Anweisung Bezug genommen wird.

   Ausnahme: SEARCH und SORT (Format 2)-Anweisung.

   Ist der Datenname Name einer Datengruppe, müssen ebenso alle Datennamen, die zu der Gruppe gehören, indiziert oder subskribiert werden, falls sie als Operanden benutzt werden.

   Ein indizierter oder subskribierter Datenname nimmt Bezug auf ein bestimmtes Tabellenelement. In der SEARCH und SORT (Format 2)-Anweisung wird mit dem Datennamen auf die gesamte Tabelle Bezug genommen.
4. Die ASCENDING-/DESCENDING-Angabe in Verbindung mit der INDEXED BY-Angabe wird bei der Ausführung einer SEARCH ALL und SORT (Format 2)-Anweisung verwendet.
5. Ein Index muss initialisiert werden (mit einer SET-, SEARCH ALL- oder PERFORM-Anweisung mit VARYING-Angabe), bevor er als Index benutzt wird.
6. Mit der ASCENDING/DESCENDING KEY-Angabe wird festgelegt, ob die Elemente der Tabelle in aufsteigender (ASCENDING) oder absteigender (DESCENDING) Reihenfolge entsprechend dem Inhalt von datenname-2 sortiert sind. Dabei sind diese Datennamen in hierarchisch absteigender Reihenfolge aufzuführen. Diese Ordnung wird in der SEARCH ALL-Anweisung vorausgesetzt (der Benutzer muss in diesem Fall für die richtige Sortierung der Tabellenelemente sorgen).
7. Mit der INDEXED BY-Angabe wird festgelegt, dass auf den Datennamen, auf den sich die OCCURS-Klausel bezieht, und auf jeden ihm untergeordneten Eintrag durch Indizierung Bezug genommen werden kann. Die Speicherzuordnung und das Format der hiermit definierten Indizes legt der Compiler automatisch fest.
8. Jeder Index enthält einen binären Wert, der die Distanz zum Tabellenbeginn angibt und einer Tabellenelementnummer entspricht. Der Wert errechnet sich aus der Tabellenelementnummer minus eins, multipliziert mit der Länge des Eintrages, der mit dem Index indiziert ist (siehe „Indizierung").
9. Verlangt ein Datenfeld innerhalb eines Tabellenelements eine besondere Ausrichtung, so sorgt der Compiler dafür, dass diese Ausrichtung auch für jede Wiederholung des Tabellenelements zutrifft (siehe Abschnitt „Herstellerabhängige Darstellung und Ausrichtung von Daten").

Allgemeine Regeln für Format 2

10. ganzzahl-1 legt die Minimal-, ganzzahl-2 die Maximalzahl der Wiederholungen fest. Der Inhalt von datenname-1 muss zwischen ganzzahl-1 und ganzzahl-2 liegen.
11. Der aktuelle Wert von datenname-1 darf bei einem Zugriff auf die Tabelle den Wert von ganzzahl-2 nicht überschreiten.
12. datenname-1 in der DEPENDING ON-Angabe der OCCURS-Klausel eines Datenbereichs variabler Länge muss mit einem Wert versorgt sein, bevor dieser Datenbereich in einer Übertragung als Sende- oder Empfangsfeld benutzt wird. Derselbe datenname-1 kann von Sende- und Empfangsfeld verwendet werden (siehe Beispiel 7-8).
13. DEPENDING ON gibt an, dass das Datenfeld, das durch die OCCURS-Klausel beschrieben wird, eine variable Anzahl von Wiederholungen hat. Zur Ausführungszeit wird die Anzahl der Wiederholungen durch den Wert von datenname-1 bestimmt.
14. Wird auf eine Datengruppe, der ein Eintrag mit einer OCCURS-Klausel mit DEPENDING ON-Angabe untergeordnet ist, Bezug genommen, wird der Teil des Tabellenfeldes in der Operation wie folgt benutzt:
    a. Ist das Datenfeld datenname-1 nicht Bestandteil der Datengruppe, wird nur der Teil des Tabellenfeldes benutzt, der bei Beginn der Operation durch den Inhalt von datenname-1 festgelegt ist.
    Sind der Datengruppe keine weiteren Einträge untergeordnet und der Inhalt von datenname-1 gleich 0, so ist die Datengruppe ein null-längiges Datenfeld.
    Sind der Datengruppe nur Einträge untergeordnet, die mit einer OCCURS-Klausel mit DEPENDING ON-Angabe definiert sind und die Inhalte von allen durch datenname-1 bezeichneten Datenfelder ist 0, so ist die Datengruppe ein null-längiges Datenfeld.
    b. Ist das Datenfeld datenname-1 Bestandteil der Datengruppe und ist das Gruppendatenfeld als Sendefeld vereinbart, wird nur der Teil des Tabellenfeldes benutzt, der bei Beginn der Operation durch den Inhalt von datenname-1 festgelegt ist. Ist die Datengruppe ein Empfangsfeld, wird die maximale Länge der Datengruppe benutzt.
15. Die Lage jedes Datenbereichs, der innerhalb einer Datensatzerklärung Datenfeldern mit DEPENDING ON-Angabe folgt und keinem dieser Datenfelder untergeordnet ist, hängt von den aktuellen Werten von datenname-1 in den vorhergehenden DEPENDING ON-Angaben ab.

    Bei Änderung des Wertes von datenname-1 (Änderung der Tabellenlänge) verschiebt sich entsprechend die Lage dieser Datenbereiche. Ihr ursprünglicher Inhalt wird aber **nicht** mit verschoben (siehe Beispiel 7-8).

> ¡ Vermindert man zur Programmausführungszeit den Wert von datenname-1, ist der Inhalt der Datenfelder, deren Tabellenelementnummer den neuen Wert von datenname-1 übersteigt, undefiniert.

Beispiel 7-8

Versorgung von datenname-1 in OCCURS DEPENDING ON am Beispiel einer Übertragung mit MOVE-Anweisung. Folgende Datendefinition sei gegeben:

```
WORKING-STORAGE SECTION.
01  A-FELD.
    02  A-ZAEHL PIC 9.
    02  A-DATEN "ABCDEFGHI".
        03  A-ZEICHEN PIC X OCCURS 1 TO 9
            DEPENDING ON A-ZAEHL.
01  B-FELD.
    02  B-ZAEHL PIC 9.
    02  B-DATEN.
        03  B-ZEICHEN PIC X OCCURS 1 TO 9
            DEPENDING ON B-ZAEHL.
```

Folgende Übertragungen sollen durchgeführt werden:

a) Sendefeldlänge größer als Empfangsfeldlänge

```
MOVE 6 TO A-ZAEHL,
MOVE 3 TO B-ZAEHL,
MOVE A-FELD TO B-FELD.
```

Inhalt nach erfolgter Übertragung:

```
ITEM-A: 6ABCDEF
ITEM-B: 6ABCDEF
```

Bei MOVE A-DATEN TO B-DATEN lautet der Inhalt:

```
ITEM-A: 6ABCDEF
ITEM-B: 3ABC
```

b) Sendefeldlänge kleiner als Empfangsfeldlänge (Inhalt der Felder wieder wie vor Fall a))

```
MOVE 3 TO A-ZAEHL,
MOVE 6 TO B-ZAEHL,
MOVE A-FELD TO B-FELD.
```

Inhalt nach erfolgter Übertragung:

```
ITEM-A: 3ABC
ITEM-B: 3ABC
```

Bei MOVE A-DATEN TO B-DATEN lautet der Inhalt:

```
ITEM-A: 3ABC
ITEM-B: 6ABC'BLANK''BLANK''BLANK
```

Die Übertragungen erfolgen nach den Regeln für alphanumerische Übertragung (siehe „MOVE-Anweisung").

Beispiel 7-9

`OCCURS DEPENDING ON datenname-1`

Folgende Datendefinition sei gegeben:

```
WORKING-STORAGE SECTION.
01  DATENSATZ.
```

```
02  TABELLE.
    03  LAENGE   PIC 9.
    03  ELEM     PIC X OCCURS 1 TO 9
                 DEPENDING ON LEN.
02  FELD         PIC X.
```

Ist 9 der aktuelle Wert von LAENGE, so ergibt sich folgende Lage der Datenfelder:

DATENSATZ	TABELLE	LAENGE	9
		ELEM (1)	A
			B . . . H
		ELEM (9)	I
	FELD		J

Nach MOVE 1 TO LAENGE

ändert sich die Tabellenlänge und damit die Lage von FELD

DATENSATZ	TABELLE	LAENGE	1
		ELEM (1)	A
	FELD		B
momentan unbenutzter Teil von DATENSATZ			. . . H I J

Das Datenfeld LAENGE hat jetzt den Wert 1, das Datenfeld FELD den Wert B.

7.3.15 PICTURE-Klausel

Funktion

Die PICTURE-Klausel legt die allgemeinen Eigenschaften eines Datenelementes fest und liefert außerdem die Angaben über die Druckaufbereitung.

Format

```
{PICTURE | PIC}  IS maskenzeichenfolge
```

Syntaxregeln

1. PIC ist die Abkürzung für PICTURE.
2. maskenzeichenfolge besteht aus bestimmten erlaubten Zeichenkombinationen aus dem COBOL-Zeichenvorrat. Diese Zeichenkombinationen bestimmen die Kategorie des Datenelementes.
3. Es gibt 18 Zeichen, die in einer Maskenzeichenfolge verwendet werden können: A, N, Komma (,), X, 9, P, Z, *, B, 0, +, Minus (-), Währungszeichen ($), Schrägstrich (/), S, V, Punkt (.), Kreditzeichen (CR), und Debetzeichen (DB). Die Funktionen jedes dieser Symbole und Zeichen sind unter „Zusammenfassung der Zeichen und Symbole in der Maskenzeichenfolge“ (siehe unten) beschrieben.
4. Die Zeichen S, V, ., CR und DB dürfen in der PICTURE-Klausel nur einmal geschrieben werden.
5. Eine Ganzzahl, die in Klammern eingeschlossen ist, kann den Symbolen A, N, Komma(,), X, 9, P, Z, *, $, B, Schrägstrich (/), 0, Minus (-) und Plus (+) folgen und bestimmt dann die Anzahl der fortlaufenden Wiederholungen des Symbols. Die Zahl in Klammern muss mindestens 1 sein und darf nicht größer als 131071 sein.
6. Wenigstens eines der Symbole A, N, X, Z, 9, * oder wenigstens zwei der Symbole +, - oder WZ müssen in einer Maskenzeichenfolge vorkommen. -
7. Die maximale Anzahl der Zeichen, die in einer Maskenzeichenfolge geschrieben werden darf, ist 50. Dies begrenzt jedoch nicht die Anzahl der Zeichen des damit beschriebenen Datenelements, die viel größer als 50 sein kann.
8. Die erlaubten Reihenfolgen der Symbole in einer Maskenzeichenfolge sind in der Tabelle 14 zusammengefasst.

   Ein X im Kreuzungspunkt bedeutet: In einer Maskenzeichenfolge darf das in der Spalte angegebene „Zweite Symbol“ an beliebiger Stelle rechts neben dem in der Zeile stehenden „Ersten Symbol“ folgen. Die linke Spalte und die obere Zeile gelten für die Symbole links vom Dezimalpunkt (l). Die rechte Spalte und die untere Zeile gelten für die Symbole rechts vom Dezimalpunkt (r).

ERSTES SYMBOL			ZWEITES SYMBOL																			
			nichtgleitende Einfügesymbole							gleitende Einfügesymbole						andere Symbole						
			B 0 /	,	.	+ -	+ -	CR DB	WZ[1)]	Z *	Z *	+ -	+ -	WZ[1)]	WZ[1)]	9	A X	S	V	P	P	N
						l	r			l	r	l	r	l	r					l	r	
nicht-gleitende Einfüge-symbole	B 0 /		X	X	X		X	X		X	X	X	X	X	X	X	X		X	X		
	,		X	X	X		X	X		X	X	X	X	X	X	X			X	X		
	.		X	X			X	X			X		X		X	X						
	+ -	l	X	X	X				X	X	X			X	X	X			X	X	X	
	+ -	r																				
	CR DB																					

	WZ[1)]		X	X	X		X	X		X	X	X	X			X			X	X	X	
gleitende Einfüge-symbole	Z *	l	X	X	X		X	X		X	X					X			X	X		
	Z *	r	X	X			X	X			X											
	+ -	l	X	X	X							X	X			X			X	X		
	+ -	r	X	X									X									
	WZ[1)]	l	X	X	X	X		X						X	X	X				X	X	
	WZ[1)]	r	X	X	X			X							X							
andere Symbole	9		X	X	X		X	X								X	X		X	X		
	A X		X													X	X					
	S															X			X	X	X	
	V		X	X			X	X			X		X		X	X					X	
	P	l					X	X											X	X		
	P	r	X	X			X	X			X		X		X	X					X	
	N																					X

Tabelle 14: Präzedenzreihenfolge der in der Picture-Klausel verwendeten Symbole

[1)] WZ ist die Abkürzung für das Währungszeichen

9. Die Anzahl der Zeichen in der Maskenzeichenfolge bestimmt die Größe des Datenelements. Die aktuelle Größe des Datenelements im Internspeicher wird jedoch durch die Kombination der PICTURE- und USAGE-Klausel bestimmt (siehe „USAGE-Klausel").
10. Daten der folgenden Kategorien können mit einer PICTURE-Klausel beschrieben werden:
    - alphabetisch
    - alphanumerisch
    - alphanumerisch druckaufbereitet
    - national
    - numerisch
    - numerisch druckaufbereitet
11. Es gibt zwei allgemeine Methoden der Druckaufbereitung in der PICTURE-Klausel: durch Einfügen oder durch Unterdrücken und Ersetzen.

    Es gibt verschiedene Arten für die Druckaufbereitung durch Einfügen:
    - einfaches Einfügen
    - besonderes Einfügen
    - festes Einfügen
    - gleitendes Einfügen

    Es gibt 2 Arten für die Druckaufbereitung durch Unterdrücken und Ersetzen:
    - Nullenunterdrückung und Ersetzen durch Leerzeichen
    - Nullenunterdrückung und Ersetzen durch Stern (*).

Allgemeine Regel

1. Die PICTURE-Klausel ist nur für Datenelemente erlaubt.

Zusammenfassung der Zeichen und Symbole in der Maskenzeichenfolge

Die Zeichen und Symbole, die in einer Maskenzeichenfolge erlaubt sind, haben folgende

Bedeutung:

A Jedes A in der Maskenzeichenfolge stellt eine Zeichenstelle dar, die nur einen Buchstaben oder ein Leerzeichen enthalten kann.

B Jedes B in der Maskenzeichenfolge stellt eine Zeichenstelle dar, in die ein Leerzeichen eingefügt wird. Jedes Leerzeichen wird zur Größe des Datenelements gerechnet.

N Jedes N in der Maskenzeichenfolge stellt eine Zeichenstelle dar, die jedes beliebige Zeichen aus dem UTF-16-Zeichensatz enthalten kann.

P Das Zeichen P stellt eine numerische Ziffernstelle dar, die zwar in der Anzahl der Ziffernstellen mitzählt, für die jedoch kein Speicherplatz reserviert wird. Gleichzeitig zeigt es die Stellung des gedachten Rechendezimalpunktes an:

- links von den P’s, wenn die P’s die am weitesten links stehenden Zeichen der Maskenzeichenfolge sind
- rechts von den P’s, wenn die P’s die am weitesten rechts stehenden Zeichen der Maskenzeichenfolge sind

In einigen Operationen, die ein Datenelement ansprechen, dessen Maskenzeichenfolge das Zeichen P enthält, wird an Stelle des aktuellen Inhalts des Datenelements sein algebraischer Wert verwendet. Für den algebraischen Wert wird jedes P so behandelt, als würde es eine Null enthalten und als würde der Dezimalpunkt in der angegebenen Stelle stehen.
Operationen, die den algebraischen Wert verwenden, sind:

- Jede Operation, die ein numerisches Sendefeld erfordert.
- Eine elementare MOVE-Anweisung (Elementarübertragung) mit numerischem Sendefeld, dessen Maskenzeichenfolge ein oder mehrere P's enthält.
- Eine MOVE-Anweisung mit numerisch druckaufbereitetem Sendefeld, dessen Maskenzeichenfolge ein oder mehrere P's enthält und einem numerischen oder numerisch druckaufbereiteten Empfangsfeld.
- Eine Vergleichsoperation, in der beide Operanden numerisch sind.

In allen anderen Operationen werden die mit dem Zeichen P beschriebenen Ziffernstellen ignoriert und nicht bei der Größe des Datenelements mitgerechnet.

S Das Zeichen S zeigt das Vorhandensein, nicht jedoch die Darstellung oder Position eines Vorzeichens an. Es muss als erstes Zeichen in der Maskenzeichenfolge angegeben sein. Das Zeichen S wird nicht zur Größe des Datenelements gerechnet, es sei denn, dass eine SIGN-Klausel mit der SEPARATE CHARACTER-Angabe gemacht wurde.

V Das Zeichen V zeigt das Vorhandensein eines Rechendezimalpunktes an. Da ein numerisches Datenelement keinen Druckdezimalpunkt enthalten kann, dient der Rechendezimalpunkt dem Compiler nur zur Information hinsichtlich der Skalenausrichtung des Datenelements bei Rechenoperationen. Für das Zeichen V wird niemals Speicherplatz reserviert und es wird deshalb nicht zur Größe des Datenelements gerechnet. Ist der Rechendezimalpunkt das am weitesten rechts stehende Zeichen in der Maskenzeichenfolge, ist das Zeichen V überflüssig.

X Jedes X in der Maskenzeichenfolge stellt eine Zeichenstelle dar, die jedes beliebige Zeichen aus dem EBCDIC-Zeichensatz enthalten kann.

Z Jedes Z in der Maskenzeichenfolge stellt eine führende numerische Zeichenstelle dar. Enthält eine solche Zeichenstelle eine Null, so wird diese Null durch ein Leerzeichen ersetzt. Jedes Z wird zur Größe des Datenelements gerechnet.

9 Jede 9 in der Maskenzeichenfolge stellt eine Zeichenstelle dar, die eine Ziffer enthält. Jede 9 wird zur Größe des Datenelements gerechnet.

0 Jede 0 in der Maskenzeichenfolge stellt eine Zeichenstelle dar, in die die Ziffer Null eingefügt wird. Jede 0 wird zur Größe des Datenelements gerechnet.

/ Jeder Schrägstrich (/) in der Maskenzeichenfolge stellt eine Zeichenstelle dar, in die ein Schrägstrich eingefügt wird. Jeder Schrägstrich wird zur Größe des Datenelements gerechnet.

, Jedes Komma (,) in der Maskenzeichenfolge stellt eine Zeichenstelle dar, in die ein Komma eingefügt wird. Jedes Komma wird zur Größe des Datenelements gerechnet.

. Der Punkt (.) in der Maskenzeichenfolge ist ein Druckaufbereitungssymbol und stellt den Dezimalpunkt dar, an dem das Datenelement ausgerichtet wird. Zusätzlich stellt er eine Zeichenstelle dar, in die ein Punkt eingefügt wird. Der Punkt wird zur Größe des Datenelements gerechnet.

Bemerkung

In einem Programm können die Funktionen von Punkt und Komma vertauscht werden, wenn die DECIMAL-POINT IS COMMA-Klausel im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION angegeben wurde. Die Regeln für den Punkt gelten dann für das Komma und umgekehrt, wann immer sie in einer Maskenzeichenfolge auftreten.

+
-
CR
DB Diese Zeichen sind Vorzeichen-Aufbereitungssymbole. Jedes dieser Symbole - stellt eine Zeichenstelle dar, in die ein Vorzeichen-Aufbereitungssymbol eingefügt wird. Diese Symbole schließen sich innerhalb einer Maskenzeichenfolge gegenseitig aus.
Jedes Zeichen, das in einem Symbol angegeben wird, wird zur Größe des Daten elements gerechnet. Die Vorzeichen-Aufbereitungssymbole liefern für positive und negative Datenele mente, abhängig vom Wert, verschiedene Ergebnisse (siehe „Druckaufbereitung durch festes Einfügen").

* Dieses Symbol ist ein Schecksicherungssymbol. Jeder Stern (*) in der Maskenzei- chenfolge stellt eine führende numerische Zeichenstelle dar, in die ein Stern eingefügt wird, wenn eine solche Zeichenstelle eine Null enthält. Jeder Stern wird zur Größe des Datenelements gerechnet.
Wird der Stern zusammen mit der BLANK WHEN ZERO-Klausel in einer Datenbeschreibung verwendet, so hebt die Druckaufbereitung durch Nullunterdrückung die Funktion der BLANK WHEN ZERO-Klausel auf.

$ Das Währungszeichen ($) in der Maskenzeichenfolge stellt eine Zeichenstelle dar, in die das Währungszeichen eingefügt wird. Das Währungssymbol in der Maskenzeichenfolge wird entweder durch das Zeichen $ oder durch ein einzelnes Zeichen, das in der CURRENCY SIGN-Klausel im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION angegeben wird, dargestellt. Das Währungssymbol wird zur Größe des Datenelements gerechnet.

Alphabetische Datenelemente

Syntaxregeln

1. Die Maskenzeichenfolge für ein alphabetisches Datenelement darf nur das Symbol A enthalten.
2. Im Inhalt eines Datenelements darf jede Kombination der 52 Buchstaben des Alphabets und des Leerzeichens angegeben werden.

Beispiel 7-10

Maske	Wert
`PICTURE AAA`	`NEU`

Alphanumerische Datenelemente Syntaxregeln

1. Die Maskenzeichenfolge für ein alphanumerisches Datenelement kann bestimmte Kombinationen der folgenden Symbole enthalten: A, X, 9.

Ein alphanumerisches Datenelement wird so behandelt, als würde seine Maskenzei-chenfolge nur X enthalten, wobei jedes X eine Zeichenstelle darstellt.

2. eine Maskenzeichenfolge, die entweder nur aus den Symbolen A oder nur aus 9 besteht, beschreibt kein alphanumerisches Datenelement.

Beispiel 7-11

Der alphanumerische Wert AB1234 könnte durch jede der folgenden Maskenzeichenfolgen dargestellt werden:

```
PICTURE XXXXXX
PICTURE AAXXXX
PICTURE AA9999
PICTURE A(2)X(4)
```

Nationale Datenelemente

Syntaxregel

1. Die Maskenzeichenfolge für ein nationales Datenelement darf nur das Symbol N enthalten.

Numerische Datenelemente

Es gibt zwei Arten von numerischen Datenelementen: Festpunktdatenelemente und Gleitpunktdatenelemente.

Festpunktdatenelemente

Es gibt drei Arten von Festpunktdatenelemente: externe dezimale, binäre und interne dezimale (siehe „USAGE-Klausel").

Syntaxregeln

1. Die Maskenzeichenfolge für ein Festpunktdatenelement kann jede gültige Kombination der folgenden Symbole enthalten: 9, V, P, S.
2. Sie darf 1 bis einschließlich 31 Ziffernpositionen enthalten.
3. Ist das Symbol „S“ nicht angegeben, darf der Inhalt eines Datenelements eine Kombination der Ziffern 0 bis 9 enthalten.
4. Ist das Symbol „S“ angegeben, dann darf der Inhalt zusätzlich zu den obigen Ziffern noch +, - oder andere Darstellungen des Vorzeichens enthalten (siehe „SYNCHRONIZED-Klausel").

Beispiel 7-12

Gültige Kombinationen für Festpunktdatenelemente:

```
PICTURE 9999
PICTURE S99
PICTURE S99V9
PICTURE PPP999
PICTURE S999PPP
```

Beispiel 7-13

Ein Datenelement enthalte die Ziffernfolge 8735:

Für PICTURE P(4)9(4) ist der arithmetische Wert dieses Feldes .00008735.
Für PICTURE 9(4)P(2) ist der arithmetische Wert dieses Feldes 873500.

Gleitpunktdatenelemente

Es gibt zwei Arten von Gleitpunktdatenelemente: interne und externe Gleitpunktdatenelemente (siehe „USAGE-Klausel").

Syntaxregeln

1. Die Maskenzeichenfolge für ein externes Gleitpunktdatenelement hat folgendes Format:

   { + | - } mantisseE { + | - } exponent

   Für die Elemente der Maskenzeichenfolge sind folgende Regeln zu beachten:

   Ein positives oder negatives Vorzeichen muss unmittelbar vor der Mantisse und vor dem Exponenten in der Maskenzeichenfolge geschrieben werden.

   + zeigt an, dass ein positives Vorzeichen positive Werte und ein negatives Vorzeichen negative Werte darstellt.

   - zeigt an, dass ein Leerzeichen positive Werte und ein negatives Vorzeichen negative Werte darstellt.

   mantisse

   Die Mantisse ist der dezimale Teil der Zahl; er besteht aus 1-18 Symbolen 9 (jede 9 stellt ein numerisches Zeichen dar) und einem führenden, eingebetteten oder angehängten Dezimalpunkt (.) oder V. Der Dezimalpunkt zeigt den Druckdezimalpunkt an und das V zeigt den Rechendezimalpunkt an; diese beiden Zeichen schließen sich gegenseitig aus.

   E

   folgt unmittelbar der Mantisse und zeigt an, dass ein Exponent folgt.

   exponent

   Der Exponent folgt unmittelbar dem zweiten Vorzeichen und besteht aus zwei aufeinanderfolgenden Symbolen 9.

2. Für ein internes Gleitpunktdatenelement darf **keine** PICTURE-Klausel angegeben werden.

   Beispiel 7-14 für ein externes Gleitpunktdatenelement

```
PICTURE   -9V99E-99
PICTURE   +9999.99E+99
PICTURE   -9(16)VE+99
PICTURE   +9(16).E-99
```

Alphanumerisch druckaufbereitete Datenelemente

Syntaxregeln

1. Die Maskenzeichenfolge für ein alphanumerisch druckaufbereitetes Datenelement kann bestimmte Kombinationen der folgenden Symbole enthalten:A, X, 9, 0, B und / (Schrägstrich).
2. Eine alphanumerisch druckaufbereitete Maskenzeichenfolge muss mindestens ein A oder X und mindestens ein B oder 0 oder / enthalten.
3. Nur eine Art der Druckaufbereitung wird für alphanumerisch druckaufbereitete Datenelemente durchgeführt: Druckaufbereitung durch einfaches Einfügen der Zeichen Null (0), Schrägstrich und Leerzeichen (B) (siehe „Druckaufbereitung durch einfaches Einfügen").

Beispiel 7-15

Maske	Wert

```
PICTURE BAAAB 'BLANK'NEU'BLANK'
```

Numerisch druckaufbereitete Datenelemente

Syntaxregeln

1. Die Maskenzeichenfolge für ein numerisch druckaufbereitetes Datenelement kann bestimmte Kombinationen der folgenden Symbole enthalten:
   B, / (Schrägstrich), P, V, Z, 0, 9, , (Komma), . (Punkt), *, +, -, CR, DB, $.
2. Eine numerisch druckaufbereitete Maskenzeichenfolge enthält mindestens eines der Symbole 0, B, / (Schrägstrich), Z, *, +, , (Komma), . (Punkt), -, CR, DB oder $.
3. Die maximale Anzahl der Ziffernstellen in der Maskenzeichenfolge ist 31.
4. Die maximale Länge eines numerisch druckaufbereiteten Datenelements ist 127 Zeichen.
5. Die erlaubten Kombinationen dieser Zeichen werden durch die Druckaufbereitungsregeln und durch die Regeln, welches Zeichen einem anderen vorangehen kann, bestimmt (siehe folgende Syntaxregeln und Tabelle 14).

Druckaufbereitung durch einfaches Einfügen

Syntaxregeln

1. Die Druckaufbereitung durch einfaches Einfügen wird mit folgenden Druckaufbereitungszeichen durchgeführt:
   , (Komma), B (Leerzeichen), 0 (Null), / (Schrägstrich).
2. Die Druckaufbereitungszeichen werden zur Größe des Datenelements gerechnet. Sie stellen die Zeichenstelle innerhalb eines Datenelements dar, in die sie eingefügt werden.

Beispiel 7-16

Datenkategorie	Maskenzeichenfolge des Empfangsfeldes	Übertragene Daten	Druckaufbereitetes Ergebnis
numerisch-druckaufbereitet	999,999 99B99B99 99B99B00 99/99/99	54321 654321 654321 654321	054,321 65'BLANK'43'BLANK'21 43'BLANK'21'BLANK'00 65/43/21
alphanumerisch-druckaufbereitet	XXBXXX 000X(5) XX/XX	123AA A5CD3 CD05	12'BLANK'3AA000A5CD3 CD/05

Druckaufbereitung durch besonderes Einfügen

Syntaxregeln

1. Die Druckaufbereitung durch besonderes Einfügen benutzt den Punkt (.) als Einfügungszeichen. Zusätzlich zu seiner Funktion als Einfügungszeichen dient der Punkt auch als Dezimal-punkt zur Ausrichtung.
2. Der Rechendezimalpunkt (dargestellt durch das Zeichen V) und der Druckdezimalpunkt dürfen nicht gemeinsam in einer Maskenzeichenfolge verwendet werden.
3. Druckaufbereitung durch besonderes Einfügen gilt nur für numerisch druckaufbereitete Datenelemente.
4. Als Ergebnis der Druckaufbereitung durch besonderes Einfügen tritt das Einfügungszeichen (Dezimalpunkt) an derselben Stelle auf, an der es auch in der Maskenzeichenfolge steht; auf diese Weise ist das

Einfügungszeichen der Druckdezimalpunkt. Der Druckdezimalpunkt wird zur Größe des Datenelements gerechnet.

Beispiel 7-17

Maskenzeichenfolge des Empfangsfeldes	Übertragene Daten*)	Druckaufbereitetes Ergebnis
999.99	123&4	123.40
999.99999.99999.99	12&341&234&1234	012.34001.23000.12

*)& bezeichnet die Stelle des Rechendezimalpunkts, der bei Übertragung nicht erscheint.

Druckaufbereitung durch festes Einfügen

Syntaxregeln

1. Die Druckaufbereitung durch festes Einfügen benutzt folgende Druckaufbereitungszeichen: + (Plus), - (Minus), CR (Kredit), DB (Debet) und $ / € (Währungszeichen).
2. Nur ein Währungszeichen und nur eines der Vorzeichen-Aufbereitungssymbole (+, -, CR, DB) dürfen in einer Maskenzeichenfolge angegeben werden.
3. Dem Zeichen für das Währungszeichen darf nur ein Plus oder Minus vorangehen; ansonsten muss es das am weitesten links stehende Zeichen sein.
4. Werden die Symbole CR oder DB angegeben, müssen sie die am weitesten rechts stehenden Zeichenstellen darstellen.
5. Werden die Symbole Plus oder Minus angegeben, müssen sie entweder die am weitesten links oder am weitesten rechts stehenden Zeichen sein.
6. Bei der Druckaufbereitung durch festes Einfügen belegt das Einfügungszeichen dieselbe Zeichenstelle im druckaufbereiteten Datenelement wie in der Maskenzeichenfolge.
7. Werden die Symbole CR oder DB verwendet, so stellen sie zwei Zeichenstellen dar, die zur Größe des Datenelements gerechnet werden. Alle Druckaufbereitungszeichen zur Druckaufbereitung durch festes Einfügen werden zur Größe des Datenelements gerechnet.
8. Die Vorzeichen-Aufbereitungssymbole liefern die Ergebnisse abhängig vom Wert des Datenelements (siehe Tabelle 15).

Druckaufbereitungszeichen in der Maskenzeichenfolge	Ergebnis bei Datenelement positiv oder null	Ergebnis bei Datenelement negativ
+ - CR DB	+ Leerzeichen 2 Leerzeichen 2 Leerzeichen	– – CR DB

Tabelle 15: Vorzeichen-Aufbereitungssymbole und ihre Ergebnisse

Beispiel 7-18

Maskenzeichenfolge des Empfangsfeldes	Übertragene Daten*)	Druckaufbereitetes Ergebnis
+999.99 +999.99 -999.99	+123&45 -123&45 +123&45	+123.45 -123.45 123.45

-999.99	-123&45	-123.45
$999.99CR	+123&45	$123.45
$999.99CR	-123&45	$123.45CR
$999.99DB	+123&45	$123.45
$999.99DB	-123&45	$123.45DB

*)& bezeichnet die Stelle des Rechendezimalpunkts, der bei Übertragung nicht erscheint.

Druckaufbereitung durch gleitendes Einfügen

Syntaxregeln

1. Die Druckaufbereitung durch gleitendes Einfügen benutzt das Währungszeichen und die Vorzeichen-Aufbereitungssymbole (+ oder -) als Druckaufbereitungszeichen. Diese Zeichen schließen sich innerhalb einer Maskenzeichenfolge gegenseitig aus.

   Die Druckaufbereitung durch gleitendes Einfügen wird durch die Angabe einer Folge von mindestens zwei der zugelassenen Einfügungszeichen in der Maskenzeichenfolge erreicht. Diese Zeichen stellen die am weitesten links stehenden numerischen Zeichen dar, in die die Einfügungszeichen gleiten können. Alle einfachen Einfügungszeichen (/, , (Komma), B, 0), die in der Zeichenfolge der gleitenden Einfügungszeichen eingebettet sind oder unmittelbar rechts an diese Zeichenfolge anschließen, sind Teil der gleitenden Zeichenfolge.
2. In einer Maskenzeichenfolge gibt es nur zwei Darstellungsarten zur Druckaufbereitung durch gleitendes Einfügen:
   - Einige oder alle führenden numerischen Zeichenstellen links vom Dezimalpunkt werden durch ein Einfügungszeichen dargestellt.
   - Alle numerischen Zeichenstellen der Maskenzeichenfolge werden durch ein Einfügungszeichen dargestellt.
3. Das Ergebnis der Druckaufbereitung durch gleitendes Einfügen hängt von der Darstellung in der Maskenzeichenfolge ab:
   - Werden die Einfügungszeichen nur links vom Dezimalpunkt angegeben, so wird ein einfaches Einfügungszeichen in die Zeichenstelle gebracht, die unmittelbar dem Dezimalpunkt vorangeht oder vor einer von Null verschiedenen Ziffer steht, die als erste von links in der Maskenzeichenfolge auftritt und innerhalb der durch die Einfügungszeichenkette dargestellten Daten liegt. Die Zeichenstellen, die dem Einfügungszeichen vorangehen, werden durch Leerzeichen ersetzt.
   - Werden alle numerischen Zeichenstellen in der Maskenzeichenfolge dargestellt, so ist das Ergebnis vom Wert des Datenelements abhängig. Ist der Wert des Datenelements Null, so enthält das ganze Datenelement Leerzeichen. Ist der Wert des Datenelements ungleich Null, so ist das Ergebnis dasselbe, als würden die Einfügungszeichen nur links vom Dezimalpunkt auftreten.
4. Jedes gleitende Einfügungszeichen wird zur Größe des Datenelements gerechnet.
5. Um ein Abschneiden von Zeichenstellen zu verhindern, muss der Programmierer bei der Bildung der Maskenzeichenfolge des Empfangsfeldes auf Folgendes achten:
   - Die Mindestgröße der Maskenzeichenfolge muss gleich sein der Anzahl der nicht-gleitenden Einfügungszeichen, die im Empfangsfeld zur Druckaufbereitung dienen, plus einem gleitenden Einfügungszeichen.

Beispiel 7-19

Maskenzeichenfolge des Empfangsfeldes	Übertragene Daten*)	Druckaufbereitetes Ergebnis
$$$$.99	123&12	$123.12
$$$$.99	3&12	$3.12
$$$$.99	&12	$.12

$,$$$.99	123&12	$123.12
$,$$$.99	3&12	$3.12
$,$$$.99	&12	$.12
+,+++.99	123&12	+123.12
+,+++.++	123&12	+123.12
$,$$$.99	-123&12	$123.12
-,---.99	-123&12	-123.12
-,---.99	123&12	'BLANK'123.12
$$,$$$.99	1234&56	$1,234.56
+,+++,999.99	-123456&78	-123,456.78
+,+++,+++.++	000&00	(Leerzeichen)

*)& bezeichnet die Stelle des Rechendezimalpunkts, der bei Übertragung nicht erscheint.

Druckaufbereitung durch Nullenunterdrückung und durch Ersetzen

Syntaxregeln

1. Die Druckaufbereitung durch Unterdrückung von führenden Nullen in numerischen Zeichenstellen benutzt als Unterdrückungszeichen das Zeichen Z oder das Zeichen * (Stern) in der Maskenzeichenfolge. Diese Zeichen schließen sich innerhalb einer Maskenzeichenfolge gegenseitig aus. Wird das Zeichen Z verwendet, so ist das Ersetzungszeichen ein Leerzeichen. Wird das Zeichen * verwendet, so ist das Ersetzungszeichen ein *.
2. Die Druckaufbereitung durch Nullenunterdrückung und Ersetzen wird durch die Angabe von einem oder mehreren der erlaubten Zeichen (* oder Z) in der Maskenzeichenfolge erreicht. Die Zeichen * oder Z stellen führende numerische Zeichenstellen dar, die mit den Ersetzungszeichen aufgefüllt werden sollen, falls diese Zeichenstellen in dem Datenelement Nullen enthalten. Alle einfachen Einfügungszeichen (/, , (Komma), B, 0), die in diese Zeichenfolge eingebettet sind oder unmittelbar rechts an diese Folge anschließen, sind Teil der Zeichenfolge; jedes dieser einfachen Einfügungszeichen wird wie ein * oder Z behandelt, bis ein Zeichen mit dem Wert ungleich Null auftritt. Die einfachen Einfügungszeichen (/, , (Komma), B, 0) und die festen Einfügungszeichen ($,+, -) links von einer solchen Zeichenfolge sind nicht von den Regeln der Nullenunterdrückung/Ersetzen betroffen.
3. In einer Maskenzeichenfolge gibt es zwei Darstellungsarten für Druckaufbereitung durch Nullenunterdrückung:
   - Einige oder alle führende numerischen Zeichenstellen links vom Dezimalpunkt werden durch Unterdrückungszeichen dargestellt.
   - Alle numerischen Zeichenstellen in der Maskenzeichenfolge werden durch Unterdrückungszeichen dargestellt.
4. Das Ergebnis der Druckaufbereitung durch Nullenunterdrückung und Ersetzen hängt von der Darstellung in der Maskenzeichenfolge ab.
   - Werden die Unterdrückungszeichen nur links vom Dezimalpunkt angegeben, so werden in dem Datenelement alle führenden Nullen, welche in ihren Zeichenstellen durch Unterdrückungszeichen dargestellt sind, durch das Ersetzungszeichen ersetzt. Die Unterdrückung wird beim Auftreten der ersten Ziffer in der Unterdrückungszeichenkette, die ungleich Null ist, oder beim Dezimalpunkt beendet.
   - Werden alle numerischen Zeichenstellen in der Maskenzeichenfolge durch Unterdrückungszeichen dargestellt und ist der Wert des Datenelements ungleich Null, so ist das Ergebnis dasselbe, als würden die Unterdrückungszeichen nur links vom Dezimalpunkt auftreten. Ist der Inhalt des Datenelements Null und das Unterdrückungszeichen Z, wird das ganze Datenelement durch Leerzeichen ersetzt.Ist der Wert des Datenelements Null und das Unterdrückungszeichen *, werden alle Zeichen in dem Datenelement

außer dem Dezimalpunkt, durch Sterne ersetzt; in diesem Fall hat die Druckaufbereitung durch Nullenunterdrückung Vorrang gegenüber der BLANK WHEN ZERO-Klausel, falls letztere angegeben wurde.

5. Jedes Unterdrückungszeichen wird zur Größe des Datenelements gerechnet.

Beispiel 7-20

Maskenzeichenfolge des Empfangsfeldes	Übertragene Daten*)	Druckaufbereitetes Ergebnis
ZZZZ.ZZ	0000&00	'BLANK''BLANK''BLANK''BLANK''BLANK''BLANK''BLANK' (Leerzeichen)
****.**	0000&00	****.**
ZZZZ.99	0000&00	'BLANK''BLANK''BLANK''BLANK'.00
****.99	0000&00	****.00
ZZZZ.ZZ	+135&00	'BLANK'135.00
$**,***.**BDB	-2135&00	$*2,135.00'BLANK'DB
$BB****,**.99BBCR	-2135&00	$'BLANK''BLANK'**21,35.00'BLANK''BLANK'CR

*)& bezeichnet die Stelle des Rechendezimalpunkts, der bei Übertragung nicht erscheint.

7.3.16 REDEFINES-Klausel

Funktion

Die REDEFINES-Klausel erlaubt dem Programmierer, für einen Speicherbereich mehrere Beschreibungen anzugeben.

Format

```
stufennummer  [datenname-1 | FILLER] REDEFINES datenname-2
```

(stufennummer und datenname-1 bzw. FILLER sind nicht Teil der REDEFINES-Klausel; sie werden hier nur zur Verdeutlichung angegeben.)

Syntaxregeln

1. Wird die REDEFINES-Klausel verwendet, so muss sie unmittelbar auf datenname-1 folgen.
2. Die Stufennummern von datenname-1 und datenname-2 müssen identisch, dürfen aber nicht 66 oder 88 sein.
3. Die Länge des Datenfeldes von datenname-1 muss kleiner oder gleich der Länge des Datenfeldes von datenname-2 sein, wenn die zugehörige Stufennummer ungleich 01 ist. Auf der Stufe 01 gibt es keine solche Beschränkung.
4. datenname-2 kann gekennzeichnet, aber nicht subskribiert oder indiziert werden.
5. Die Datenerklärung für datenname-2 darf keine OCCURS-Klausel enthalten, jedoch darf datenname-2 einem Datenfeld untergeordnet sein, das eine OCCURS-Klausel enthält. In diesem Fall darf die Bezugnahme auf datenname-2 in der REDEFINES-Klausel nicht subskribiert oder indiziert werden. Ein Datenfeld, das datenname-2 untergeordnet ist, darf eine OCCURS-Klausel ohne die DEPENDING ON-Angabe enthalten (siehe „OCCURS-Klausel").
6. datenname-1 oder ein Datenfeld, das datenname-1 untergeordnet ist, darf eine OCCURS-Klausel ohne die DEPENDING ON-Angabe enthalten. Enthält datenname-1 eine OCCURS-Klausel, so wird die Größe von datenname-1 durch Multiplikation der Länge eines Tabellenelementes mit der Anzahl der Tabellenelemente errechnet.
7. Die REDEFINES-Klausel darf in der FILE SECTION nicht in Erklärungen der Stufe 01 verwendet werden; eine implizite Neubelegung ist dort auf der Stufe 01 automatisch gegeben.
8. Mehrfache Neubelegungen des Speicherbereichs sind erlaubt, sie müssen sich jedoch alle auf den Datennamen in der Originalerklärung beziehen.
9. Erklärungen, die einen Speicherbereich neu beschreiben, dürfen mit Ausnahme der Erklärungen für Bedingungsnamen keine VALUE-Klausel enthalten.
10. Zwischen den Erklärungen von datenname-2 und datenname-1 dürfen keine Erklärungen liegen, die kleinere Stufennummern haben als datenname-1 und datenname-2.
11. Die REDEFINES-Klausel darf sowohl für ein Datenfeld, das einem neubelegten Datenfeld untergeordnet ist, als auch für ein Datenfeld, das einem Datenfeld, welches eine REDEFINES-Klausel enthält, untergeordnet ist, angegeben werden.
12. Die REDEFINES-Klausel darf nicht innerhalb einer Typdeklaration stehen.
13. Die REDEFINES-Klausel darf nicht für Daten der Klassen objekt oder zeiger sowie für Datengruppen, die solche Daten enthalten, angegeben werden.
14. datenname-2 darf nicht von den Klassen objekt oder zeiger oder einer Datengruppe, die solche Daten enthält oder eine stark typisierte Datengruppe sein.
15. datenname-2 darf nicht mit der ANY LENGTH-Klausel definiert sein.
16. Die REDEFINES-Klausel darf nicht in der Satzbeschreibung einer XML-organisierten Datei angegeben werden.

Allgemeine Regeln

1. datenname-1 ist der Name des mit der Neubelegung verbundenen Datenbereichs. datenname-2 ist der Name der ersten Erklärung des Datenbereichs, der neu belegt werden soll.
2. Die Neubelegung beginnt mit datenname-2 und endet, wenn eine Stufennummer kleiner als die von datenname-2 oder gleich der von datenname-2 ist.
3. Wird ein Bereich neu belegt, bleiben alle Erklärungen des Bereichs in Wirkung. Zum Beispiel: Wenn A und B zwei getrennte Datenfelder sind, die denselben Speicherbereich belegen, könnten die Prozeduranweisungen MOVE ALPHA TO A oder MOVE BETA TO B an irgendeiner Stelle im Programm ausgeführt werden. Im ersten Fall würde ALPHA nach A übertragen werden und würde die Darstellung annehmen, die in der Beschreibung von A angegeben wurde. Im zweiten Falle würde BETA in denselben physischen Bereich übertragen werden und würde die Darstellung annehmen, die in der Beschreibung von B angegeben wurde. Würden beide MOVE-Anweisungen unmittelbar hintereinander in der angegebenen Reihenfolge ausgeführt werden, würde der Wert von ALPHA durch den Wert von BETA überlagert werden. Die Neubelegung eines Bereiches löscht jedoch keine Daten und ersetzt keine vorangegangene Beschreibung.
4. Die Übertragung eines Datenfeldes von A nach B, wobei B die Neubelegung von A ist, ist gleichermaßen die Übertragung von einem Datenfeld zu sich selbst. Das Ergebnis einer solchen Übertragung ist nicht vorhersagbar. Dasselbe passiert bei einer umgekehrten Übertragung: wenn A nach B übertragen wird und A B neu belegt.
5. Die Verwendung von Datenfeldern, definiert durch die PICTURE- und USAGE-Klausel, innerhalb eines Bereichs kann neu festgelegt werden. Die Änderung der Verwendung eines Bereichs durch die REDEFINES-Klausel verändert jedoch keine vorhandenen Daten.
6. Wird die SYNCHRONIZED-Klausel in einer Datenerklärung angegeben, die einen vorhergehenden Bereich neu belegt, muss der Benutzer darauf achten, dass der Bereich, der neu belegt wird, auf Halbwort-, Wort- und Doppelwortgrenze ausgerichtet wird.

Beispiel 7-21

```
02  ALPHA.
    03  A-1 PICTURE X(3).
    03  A-2 PICTURE X(2).
02  BETA REDEFINES ALPHA PICTURE 9(5).
02  GAMMA.
```

BETA ist datenname-1; ALPHA ist datenname-2. BETA belegt den ALPHA zugeordneten Bereich neu (d.h. den Bereich, der von A-1 und A-2 belegt ist). Die Neubelegung beginnt bei BETA und endet bei der nächsten Stufennummer 02 (die Nummer, die GAMMA vorangeht).

Beispiel 7-22

für mehrfache Neubelegungen:

```
   02 ALPHA PICTURE 9(3).
   02 BETA REDEFINES ALPHA PICTURE X(3).
   02 GAMMA REDEFINES ALPHA PICTURE A(3).
```

Beispiel 7-23

```
01 BEISPIEL-1.
   02  ERST-ERKLAERUNG PICTURE 99 VALUE 12.
```

```
    02  ZWEIT-ERKLAERUNG REDEFINES ERST-ERKLAERUNG
        USAGE COMPUTATIONAL PICTURE S9(4).
```

In diesem Beispiel ist ERST-ERKLAERUNG eine 2-Byte vorzeichenlose externe Dezimalzahl mit dem Wert 12, d.h. der Inhalt dieser 2 Bytes ist sedezimal X'F1F2'.
ZWEIT-ERKLAERUNG ist ebenfalls eine Zahl und belegt dieselben Bytes; aber diese Zahl hat nicht den Wert 12. Die Daten in diesen beiden Bytes (X'F1F2') bleiben durch die Neubelegung unverändert; und da ZWEIT-ERKLAERUNG eine Binärzahl mit Vorzeichen ist, haben diese Daten den Wert -3598.

Beispiel 7-24

```
01 BEISPIEL-2.
   02 ERST-ERKLAERUNG.
      03 ALPHA                     PICTURE X(3).
      03 BETA                      PICTURE X(5).
      03 GAMMA REDEFINES BETA      PICTURE 9(5).
      03 FILLER                    PICTURE X(10).
   02 ZWEIT-ERKLAERUNG REDEFINES ERST-ERKLAERUNG PICTURE X(18).
```

In diesem Beispiel wird eines der Datenfelder, das ERST-ERKLAERUNG untergeordnet ist, neu belegt: GAMMA REDEFINES BETA. Das ist erlaubt und wird durch die Tatsache, dass ERST-ERKLAERUNG selbst später durch ZWEIT-ERKLAERUNG neu belegt wird, nicht blockiert.

Beispiel 7-25

```
01 BEISPIEL-3.
   02  ERST-ERKLAERUNG PICTURE S9(7).
   02  ZWEIT-ERKLAERUNG REDEFINES ERST-ERKLAERUNG.
       03  A-1 PICTURE A.
       03  N-1 REDEFINES A-1 PICTURE 9.
       03  FILLER PICTURE X(6).
```

In diesem Beispiel wird eines der Datenfelder, das ZWEIT-ERKLAERUNG untergeordnet ist, neu belegt; N-1 REDEFINES A-1. Das ist erlaubt und wird durch die Tatsache, dass ZWEIT-ERKLAERUNG selbst eine Neubelegung ist, nicht blockiert.

7.3.17 RENAMES-Klausel

Funktion

Die RENAMES-Klausel erlaubt eine andere, sich möglicherweise überlappende Gruppierung von Datenelementen. Diese Klausel weist einem oder mehreren Datenfeldern, die durch die Datensatzbeschreibung festgelegt worden sind, einen neuen Namen zu. Anders als die REDEFINES-Klausel belegt die RENAMES-Klausel bestehende Datenerklärungen nicht neu, sondern erlaubt, dass Daten unter anderem Namen zugänglich und/oder gruppiert werden, wobei die vorher erklärte Datenbeschreibung erhalten bleibt.

Format

```
66 datenname-1 RENAMES datenname-2 [ {THRU | THROUGH} datenname-3 ]
```

(stufennnummer 66 und datenname-1 sind nicht Teil der RENAMES-Klausel; sie werden hier nur zur Verdeutlichung angegeben.)

Syntaxregeln

1. Alle Einträge der RENAMES-Klausel, die sich auf Datenfelder innerhalb eines gegebenen Datensatzes beziehen, müssen unmittelbar der letzten Datenbeschreibung des Datensatzbeschreibungseintrags folgen.
2. datenname-2 muss datenname-3 in der Datensatzbeschreibung vorangehen. Nach jeder Neubenennung muss der Anfangspunkt des durch datenname-3 beschriebenen Bereichs logisch hinter dem Anfangspunkt des durch datenname-2 beschriebenen Bereichs liegen.
3. datenname-2 und datenname-3 müssen Namen von Datenelementen oder Gruppen von Datenelementen im zugehörigen logischen Datensatz sein. Sie dürfen nicht derselbe Datenname sein.
4. Der Anfang des durch datenname-3 beschriebenen Bereichs darf nicht links vom Anfang des durch datenname-2 beschriebenen Bereichs liegen. Das Ende des durch datenname-3 beschriebenen Bereichs muss rechts vom Ende des durch datenname-2 beschriebenen Bereichs liegen. datenname-3 kann aus diesem Grund datenname-2 nicht untergeordnet sein. Weder datenname-2 und datenname-3 noch die dazwischen liegenden Daten dürfen von der Klasse objekt oder der Klasse zeiger sein. Weder datenname-2 noch datenname-3 darf aus einer Typdeklaration stammen.
5. Keines der Datenfelder innerhalb des Bereichs von datenname-2 und datenname-3, darf, wenn angegeben, eine variable Größe, wie in den OCCURS-Klauseln beschrieben, haben (siehe „OCCURS-Klausel" mit DEPENDING ON-Angabe).
6. datenname-1 darf nicht als Kennzeichner verwendet werden und darf nur durch die Namen von 01-, SD- und FD-Einträgen gekennzeichnet werden.
7. datenname-2 und datenname-3 dürfen gekennzeichnet werden.
8. Weder datenname-2 noch datenname-3 dürfen eine OCCURS-Klausel in ihrer Datenerklärung haben, noch dürfen sie einem Datenfeld untergeordnet sein, das in seiner Datenerklärung eine OCCURS-Klausel enthält.
9. Die RENAMES-Klausel darf sich weder auf eine andere Erklärung der Stufennummer 66 noch eine Erklärung der Stufennummern 77, 88 und 01 beziehen.
10. Die RENAMES-Klausel darf nicht innerhalb einer Typdeklaration angegeben werden.
11. Die RENAMES-Klausel darf nicht in der Satzbeschreibung einer XML-organisierten Datei angegeben werden.

Allgemeine Regeln

1. Es können mehrere RENAMES-Klauseln für einen Datensatz geschrieben werden.
2. datenname-1 bezeichnet eine andere Erklärung eines oder mehrerer Datenfelder.
3. datenname-2 oder datenname-3 bezeichnet das Datenfeld oder die Datenfelder, die neu benannt werden.
4. Wird datenname-3 angegeben, ist datenname-1 eine Datengruppe, die alle Datenelemente einschließt:

- beginnend bei datenname-2 (falls dies ein Datenelement ist); oder beginnend bei dem ersten Datenelement innerhalb von datenname-2 (falls dies eine Datengruppe ist) und
- endend bei datenname-3 (falls dies ein Datenelement ist) oder endend bei dem letzten Datenelement innerhalb von datenname-3 (falls dies eine Datengruppe ist).

5. Wenn datenname-3 nicht angegeben ist, kann datenname-2 entweder eine Datengruppe oder ein Datenelement sein. Wenn datenname-2 eine Datengruppe ist, wird datenname-1 als Datengruppe behandelt; wenn datenname-2 ein Datenelement ist, wird datenname-1 als Datenelement behandelt.

Beispiel 7-26

Das folgende Beispiel zeigt, wie man die RENAMES-Klausel in einem Programm verwenden kann:

```
01  EINGABE-SATZ.
    02  ARTIKEL-1.
        03  ARTIKEL-NR              PIC 99.
        03  PREIS                   PIC 9999.
    02  ARTIKEL-2.
        03  ARTIKEL-NR              PIC 99.
        03  PREIS                   PIC 9999.
    02  ARTIKEL-3.
        03  ARTIKEL-Nr.             PIC 99.
        03  PREIS                   PIC 9999.
    66  ART-EINS RENAMES ARTIKEL-1.
    66  ART-ZWEI RENAMES ARTIKEL-1 THRU ARTIKEL-2.
    66  ART-DREI RENAMES ARTIKEL-1 THRU ARTIKEL-3.
```

In diesem Fall würde jede Bezugnahme auf ART-EINS die Datengruppe ARTIKEL-1, jede Bezugnahme auf ART-ZWEI die Datengruppen ARTIKEL-1 und ARTIKEL-2, jede Bezugnahme auf ART-DREI die Datengruppen ARTIKEL-1, ARTIKEL-2 und ARTIKEL-3 ansprechen.

7.3.18 SIGN-Klausel

Funktion

Die SIGN-Klausel beschreibt die Position und die Darstellungsart des Rechenvorzeichens für numerische Datenfelder.

Format

```
[SIGN IS] { LEADING | TRAILING } [SEPARATE CHARACTER]
```

Syntaxregeln

1. Die SIGN-Klausel darf nur für eine numerische Datenerklärung, die in der PICTURE-Klausel mit dem Symbol „S“ beschrieben ist, oder für eine Datengruppe, die mindestens eine solche Datenerklärung enthält, angegeben werden.
2. Die Datenerklärungen müssen explizit oder implizit mit USAGE IS DISPLAY beschrieben sein.
3. Wenn eine SIGN-Klausel für ein Gruppenfeld oder ein numerisches Datenelement angegeben ist, das einem Gruppenfeld untergeordnet ist, für das ebenfalls eine SIGN-Klausel vorliegt, dann hat die SIGN-Klausel des untergeordneten Feldes den Vorrang.
4. Ist die CODE-SET-Klausel angegeben, muss jede mit Vorzeichen versehene Datenerklärung mit der SIGN IS SEPARATE-Klausel beschrieben sein.
5. Die SIGN-Klausel gibt die Position und Darstellungsart des Rechenvorzeichens an. Wird sie für eine Datengruppe angegeben, gilt sie für jede numerische Datenerklärung innerhalb dieser Gruppe. Sie wird nur für numerische Datenerklärungen verwendet, die in der Maskenzeichenfolge das Symbol „S“ enthalten. Das „S“ zeigt lediglich das Vorhandensein, nicht jedoch die Darstellungsart des Rechenvorzeichens an.
6. Eine numerische Datenerklärung, deren Maskenzeichenfolge ein „S“ enthält, für die aber die SIGN-Klausel nicht angegeben wurde, hat ein Rechenvorzeichen. Die Darstellung und Position werden durch das Symbol „S“ jedoch nicht spezifiziert (Darstellung des Rechenvorzeichens siehe „USAGE-Klausel").

Allgemeine Regeln

1. Ist die SEPARATE CHARACTER-Angabe nicht vorhanden, gelten folgende Regeln:
   a. Das Symbol „S“ in der Maskenzeichenfolge wird nicht zur Größe des Datenfeldes gerechnet.
   b. Es wird angenommen, dass das Rechenvorzeichen im Platz der führenden (LEADING) bzw. der letzten (TRAILING) Ziffer des numerischen Datenelementes enthalten ist. Dabei entspricht die TRAILING-Angabe der Standardannahme dieses Compilers.
   c. Für den Compiler ist das Rechenvorzeichen für positiv das Halbbyte C, für negativ das Halbbyte D (siehe nachfolgendes Beispiel).
2. Ist die SEPARATE CHARACTER-Angabe vorhanden, gelten folgende Regeln:
   a. Das Symbol „S“ in einer Maskenzeichenfolge wird zur Größe des Datenfeldes gerechnet.
   b. Es wird angenommen, dass das Rechenvorzeichen im Platz der führenden (LEADING) bzw. der letzten (TRAILING) Ziffer des numerischen Datenelementes enthalten ist. Diese Zeichenposition ist keine Ziffernposition.
   c. Die Rechenvorzeichen für positiv und negativ sind standardmäßig „+“ bzw. „-“.
3. Jede numerische Datenerklärung, die in der Maskenzeichenfolge das Symbol „S“ enthält, ist eine mit Vorzeichen versehene Datenerklärung. Wird eine SIGN-Klausel für eine solche Erklärung angegeben und ist eine Konvertierung für arithmetische Operationen oder für Vergleiche notwendig, erfolgt diese Konvertierung automatisch.

Beispiel 7-27

```
IDENTIFICATION DIVISION.
PROGRAM-ID. VZ.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FELD1          PIC S999   SIGN IS LEADING SEPARATE.
01  GRUPPE1        USAGE IS DISPLAY.
    02  FELD2      PIC S9(5)  SIGN IS TRAILING SEPARATE.
    02  FELD3      PIC X(15).
    02  FELD4      PIC S99    SIGN IS LEADING.
01  FELD5          PIC S9(9)  SIGN IS TRAILING.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    MOVE ZEROES TO FELD1 FELD2 FELD3 FELD4 FELD5.
    MOVE 3 TO FELD4.
    MOVE -2 TO FELD5.
    MOVE FELD4 TO FELD2.
    MOVE FELD2 TO FELD3.
    MOVE FELD5 TO FELD4.
    DISPLAY "FELD1 = " FELD1 UPON T.
    DISPLAY "FELD2 = " FELD2 UPON T.
    DISPLAY "FELD3 = " FELD3 UPON T.
    DISPLAY "FELD4 = " FELD4 UPON T.
    DISPLAY "FELD5 = " FELD5 UPON T.
    STOP RUN.
```

Inhalt aller Felder nach der ersten MOVE-Anweisung:

FELD1 dezimal	+	0	0	0											
hexadezimal	4E	F0	F0	F0											
FELD2 dezimal	0	0	0	0	0	+									
hexadezimal	F0	F0	F0	F0	F0	4E									
FELD3 dezimal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
hexadezimal	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0	F0
FELD4 dezimal	0+														0
hexadezimal	C0	F0													
FELD5 dezimal	0	0	0	0	0	0	0	0	0+						
hexadezimal	F0	F0	F0	F0	F0	F0	F0	F0	C0						

nach der zweiten MOVE-Anweisung:

FELD4 dezimal	+	3													
hexadezimal	C0	F3													

nach der dritten MOVE-Anweisung:

FELD5 dezimal	0	0	0	0	0	0	0	0	2-						
hexadezimal	F0	F0	F0	F0	F0	F0	F0	F0	D2						

nach der vierten MOVE-Anweisung:

FELD2 dezimal	0	0	0	0	3	+									
hexadezimal	F0	F0	F0	F0	F3	4E									

nach der fünften MOVE-Anweisung:

FELD3 dezimal	0	0	0	0	3										
hexadezimal	F0	F0	F0	F0	F3	40	40	40	40	40	40	40	40	40	40

nach der sechsten MOVE-Anweisung:

FELD4 dezimal															2

	0-
hexadezimal	D0 F2

7.3.19 SYNCHRONIZED-Klausel

Funktion

Die SYNCHRONIZED-Klausel dient zur Ausrichtung eines Datenelements an einer vorgegebenen Grenze des Arbeitsspeichers der Datenverarbeitungsanlage, um eine effiziente Ausführung von arithmetischen Operationen für dieses Datenelement zu gewährleisten.

Als Erweiterung zum Standard erlaubt der Compiler, dass die SYNCHRONIZED-Klausel auf Gruppenebene angegeben werden kann; dies bewirkt eine Ausrichtung der dieser Gruppe untergeordneten Datenelemente.

Format

```
{SYNCHRONIZED | SYNC} [LEFT | RIGHT]
```

Syntaxregeln

1. SYNC ist die Abkürzung für SYNCHRONIZED.
2. LEFT und RIGHT werden als Kommentare behandelt.
3. Die SYNCHRONIZED-Klausel darf nicht für Daten der Klassen national, objekt und zeiger angegeben werden.

Allgemeine Regeln

1. Durch die Angabe der SYNCHRONIZED-Klausel zur Ausrichtung eines Datenelements ist es manchmal notwendig, dass der Compiler Füllfeld-Bytes einfügt. Füllfeld-Bytes sind unbenutzte Zeichenstellen, die in einem Datensatz unmittelbar vor dem Datenfeld, das eine Ausrichtung verlangt, eingefügt werden. Diese Füllfeld-Bytes zählen zur Länge der Datengruppe, die das ausgerichtete Datenelement enthält.
2. Die eigentliche Grenze zur Ausrichtung eines Datenelementes hängt von der jeweiligen USAGE-Klausel des Datenelements ab.
3. Wird für binäre Datenfelder oder interne Gleitpunktdatenfelder die SYNCHRONIZED-Klausel **nicht** angegeben, so werden keine Füllfeld-Bytes angelegt. Werden jedoch Rechenoperationen auf diese Felder ausgeführt, so generiert der Compiler die notwendigen Anweisungen, um die Datenfelder in Hilfsfelder, die die notwendige Ausrichtung für die Rechenoperation haben, zu übertragen.
4. Wird die SYNCHRONIZED-Klausel für eine Datengruppe angegeben, so werden alle Datenelemente mit USAGE COMPUTATIONAL, COMPUTATIONAL-5, BINARY, COMPUTATIONAL-1 oder COMPUTATIONAL-2 so ausgerichtet, als wäre die SYNCHRONIZED-Klausel in ihren Datenbeschreibungen angegeben worden.
5. Wird die SYNCHRONIZED-Klausel angegeben, so werden folgende Aktionen ausgeführt:
   - Für ein Datenfeld mit USAGE COMPUTATIONAL, COMPUTATIONAL-5 oder BINARY:
     a. Für den Bereich S9 bis S9(4) in der PICTURE-Klausel wird das Datenfeld an einer Halbwortgrenze (2 Bytes) ausgerichtet.
     b. Für den Bereich S9(5) bis S9(31) in der PICTURE-Klausel wird das Datenfeld an einer Wortgrenze (4 Bytes) ausgerichtet.
   - Für ein Datenfeld mit USAGE COMPUTATIONAL-1 wird das Datenfeld an einer Wortgrenze ausgerichtet.
     Für ein Datenfeld mit USAGE COMPUTATIONAL-2 wird das Datenfeld an einer Doppelwortgrenze (8 Bytes) ausgerichtet.

     Anmerkungen:

     - Für ein Datenfeld mit USAGE DISPLAY oder COMPUTATIONAL-3 bzw. PACKED-DECIMAL wird die SYNCHRONIZED-Klausel als Kommentar betrachtet, da in diesen Fällen keine Ausrichtung notwendig ist.

- Für ein Datenfeld mit USAGE INDEX wird die SYNCHRONIZED-Klausel ebenfalls als Kommentar betrachtet, da in diesem Fall immer eine Ausrichtung auf Wortgrenze erfolgt.

Was Sie im Einzelnen bei der Ausrichtung von Datenfeldern beachten müssen, ist in Tabelle 16 zusammengefasst.

USAGE	SYNC erlaubt	Ausrichtung ohne SYNC	Ausrichtung mit SYNC
BINARY, COMP, COMP-5	ja	B	HW[1] W[2]
COMP-1	ja	B	W
COMP-2	ja	B	DW
COMP-3, PACKED-DECIMAL	ja[3]	B	B
DISPLAY	ja[3]	B	B
INDEX	ja[3]	W	W
NATIONAL	nein	B	-
OBJECT REFERENCE	nein	DW	-
POINTER	nein	W	-
PROGRAM-POINTER	nein	W	-

Tabelle 16: Ausrichtung von Datenfeldern

B	Ausrichtung auf Byte-Grenze	W	Ausrichtung auf Wort-Grenze
HW	Ausrichtung auf Halbwort-Grenze	DW	Ausrichtung auf Doppelwort-Grenze

[1)] 1-4 Ziffern

[2)] >=5 Ziffern

[3)]Angabe zulässig, jedoch ohne Einfluss auf die Ausrichtung

6. Wird die SYNCHRONIZED-Klausel innerhalb einer Tabelle (beschrieben durch die OCCURS-Klausel) angegeben, wird jedes Tabellenelement ausgerichtet. Dies ist unter „Ausrichtung durch Einschieben von Füllfeld-Bytes“ beschrieben.
7. Wird eine SYNCHRONIZED-Klausel im Zusammenhang mit einer REDEFINES-Klausel angegeben, so muss der Programmierer sicherstellen, dass das Datenelement, das neu belegt werden soll, ausgerichtet ist (siehe Beispiel 7-28).
8. Die SYNCHRONIZED-Klausel ändert nicht die Länge eines Datenelements. Jeder unbenutzte Internspeicherplatz (Füllfeld-Bytes) zählt zur Größe der Gruppe, zu der dieses Datenelement gehört, und muss in der Internspeicherbelegung berücksichtigt werden, wenn die Gruppe als Objekt in einer REDEFINES-Klausel angegeben wurde (siehe Beispiel 7-29).
9. Alle Satzbeschreibungen (01-Stufen) in allen Kapiteln des Datenteils beginnen an Doppelwortgrenzen.
10. Werden Datensätze, die Datenelemente mit der SYNCHRONIZED-Klausel enthalten, geblockt, so muss der Benutzer die notwendigen Füllfeld-Bytes hinzufügen, um eine richtige Ausrichtung nach dem ersten Datensatz innerhalb des Blockes zu gewährleisten. Dies ist unter „Ausrichtung durch Einschieben von Füllfeld-Bytes“ beschrieben.
11. Zum Zwecke der Ausrichtung von Datenelementen mit USAGE COMPUTATIONAL, COMPUTATIONAL-5, BINARY, COMPUTATIONAL-1, COMPUTATIONAL-2 im LINKAGE-Kapitel werden alle Elemente der 01-Stufe als auf Doppelwortgrenze ausgerichtet angenommen. Infolgedessen muss der Benutzer sicherstellen,

dass, wenn er eine CALL-Anweisung schreibt, diese Operanden in der USING-Angabe entsprechend ausgerichtet sind.

Beispiel 7-28

Im folgenden Beispiel muss A an einer Wortgrenze stehen:

```
    02  A  PICTURE X(4).
    02  B  REDEFINES A PICTURE S9(9) USAGE BINARY SYNC.
```

Beispiel 7-29

```
01 SATZ.
   02 A.
      03 G PICTURE X(5).
      03 H PICTURE S9(9) SYNC USAGE BINARY.
   02 B REDEFINES A.
      03 I PICTURE X(12).
```

Hierbei belegt das Datenelement G 5 Bytes und das Datenelement H 4 Bytes.

Die SYNCHRONIZED- und USAGE-Klausel zeigt an, dass das Datenelement H an Wortgrenze ausgerichtet wird, deshalb gehen dem Datenelement H 3 Füllfeld-Bytes voraus. Da das Datenfeld A im ganzen 12 Bytes belegt, muss das Subjekt der REDEFINES-Klausel, das Datenfeld B, ebenfalls 12 Bytes belegen.

7.3.20 TYPE-Klausel

Funktion

Die TYPE-Klausel legt fest, dass die Datenbeschreibung durch eine Typerklärung definiert wird.

Format

```
TYPE TO typname-1
```

Syntaxregeln

1. Einer Datenbeschreibung mit TYPE-Klausel darf weder eine untergeordnete Datenbeschreibung noch ein Bedingungsname folgen.
2. Die aktuelle Datenbeschreibung darf weder insgesamt noch teilweise mit Hilfe der RENAMES-Klausel umbenannt werden.
3. Die aktuelle Datenbeschreibung darf weder implizit noch explizit ganz oder teilweise redefiniert werden.
4. Übergeordnete Datengruppen dürfen keine SIGN-Klausel, keine USAGE-Klausel und keine GROUP-USAGE-Klausel enthalten.
5. Wenn typname-1 die STRONG-Angabe enthält, muss die aktuelle Datenbeschreibung entweder die Stufennummer 01 haben oder muss einer Typbeschreibung mit der Angabe STRONG untergeordnet sein.
6. Ist die TYPE-Klausel in einer Datenbeschreibung mit Stufennummer 77 angegeben, so muss typname-1 ein Datenelement beschreiben.
7. Eine Datenbeschreibung darf neben der TYPE-Klausel folgende Klauseln enthalten: BASED-, EXTERNAL-, GLOBAL-, OCCURS- und TYPEDEF-Klausel.

Allgemeine Regeln

1. Die TYPE-Klausel legt fest, dass die aktuelle Datenbeschreibung durch typname-1 bestimmt wird. Die TYPE-Klausel wirkt so, als wäre die Datenbeschreibung von typname-1 an der Stelle erfolgt, wo die TYPE-Klausel steht, mit Ausnahme von Stufennummer, Name und der GLOBAL- und TYPEDEF-Klausel, die bei typname-1 angegeben ist.
2. Wenn typname-1 eine Datengruppe beschreibt, gilt:
   a. Die aktuelle Datenbeschreibung ist eine Datengruppe, deren untergeordnete Elemente denselben Namen, dieselbe Beschreibung und Hierarchie haben wie die untergeordneten Elemente von typname-1.
   b. Die Stufennummern der Datenbeschreibungen, die der aktuellen Datenbeschreibung untergeordnet sind, werden wenn notwendig angepasst, um die Hierarchie von typename-1 zu erhalten.
   c. Stufennummern in der resultierenden Hierarchie können dadurch 49 überschreiten.
   d. Die aktuelle Datenbeschreibung wird auf Doppelwortgrenze ausgerichtet.

7.3.21 TYPEDEF-Klausel

Funktion

Die TYPEDEF-Klausel legt fest, dass die Datenbeschreibung eine Typdeklaration ist.

Format

```
01 typname-1 IS TYPEDEF [STRONG]
```

Syntaxregeln

1. Die TYPEDEF-Klausel darf nur in einer Datenbeschreibung mit Stufennummer 01 verwendet werden. Die TYPEDEF-Klausel muss unmittelbar als erste Klausel stehen.
2. Weder die aktuelle Datenbeschreibung, noch die Beschreibung eines untergeordneten Datenfeldes darf direkt oder indirekt ein Datenfeld vom typname-1 sein.
3. Wenn die aktuelle Datenbeschreibung ein Datenelement ist, ist die STRONG-Angabe nicht erlaubt.
4. Die Angaben TYPEDEF und REDEFINES dürfen nicht zusammen verwendet werden.
5. Eine Typdeklaration darf keine OCCURS-Klausel enthalten.
6. Datentypen, die einer stark typisierten Typdeklaration untergeordnet sind, müssen selbst stark typisiert oder elementar sein.

Allgemeine Regeln

1. Wird die TYPEDEF-Klausel angegeben, so ist die Datenbeschreibung eine Typdeklaration. Untergeordnete Datenbeschreibungen und Bedingungsnamen sind Teil der Typdeklaration. Die Datennamen dieser Beschreibungen können nur als untergeordnete Beschreibungen von Datengruppen referenziert werden, die den Typnamen benutzen. Wenn mehr als eine Datengruppe den Typnamen referenziert, ist die Qualifizierung mit dem Namen der Gruppe notwendig. Wenn es keine Datengruppe mit dieser Eigenschaft gibt, verweist jede Referenz eines Datennamens untergeordnet zur aktuellen Typdeklaration auf andere Datenbeschreibungen, soweit es diese gibt, oder ist eine ungültige Referenz.
2. Einer Typdeklaration wird kein Speicher zugewiesen.
3. Die GLOBAL-Klausel bezieht sich auf den Gültigkeitsbereich des Typnamens. Alle anderen Klauseln der Datenbeschreibung und untergeordneter Datenbeschreibungen beziehen sich auf Daten, die mit Hilfe der Typbeschreibung definiert werden.
4. Weitere Einzelheiten und Einschränkungen sind im Abschnitt „Datentypen" beschrieben.

7.3.22 USAGE-Klausel

Funktion

Die USAGE-Klausel legt fest, in welchem Datenformat ein Datenelement im Internspeicher der Datenverarbeitungsanlage abgespeichert wird.

Format

```
[USAGE IS] {  BINARY
            | COMPUTATIONAL
            | COMP
            | COMPUTATIONAL-1
            | COMP-1
            | COMPUTATIONAL-2
            | COMP-2
            | COMPUTATIONAL-3
            | COMP-3
            | COMPUTATIONAL-4
            | COMP-4
            | COMPUTATIONAL-5
            | COMP-5
            | DISPLAY
            | INDEX
            | NATIONAL
            | OBJECT REFERENCE [  interfacename-1
                               | [FACTORY OF] ACTIVE-CLASS
                               | [FACTORY OF] klassenname-1 [ONLY]
                              ]
            | PACKED-DECIMAL
            | POINTER [TO typname-1]
            | PROGRAM-POINTER
          }
```

Syntaxregeln

1. COMP ist die Abkürzung für COMPUTATIONAL.
   COMP-1 ist die Abkürzung für COMPUTATIONAL-1.
   COMP-2 ist die Abkürzung für COMPUTATIONAL-2.
   COMP-3 ist die Abkürzung für COMPUTATIONAL-3.
   COMP-5 ist die Abkürzung für COMPUTATIONAL-5.
2. Wird die USAGE-Klausel für ein Datenelement oder eine Datengruppe nicht angegeben, so wird
   a. USAGE NATIONAL angenommen, wenn die explizite oder implizite PICTURE Maskenzeichenfolge das Zeichen N enthält
   b. USAGE DISPLAY in allen anderen Fällen angenommen.
3. Für die Beschreibung der verschiedenen Datenkategorien siehe Kapitel „Einführung in die COBOL-Sprache" .

Allgemeine Regeln

1. Die USAGE-Klausel kann auf jeder Datenbeschreibungsstufe angegeben werden. Wird die USAGE-Klausel auf Gruppenebene angegeben, so gilt sie für alle Datenelemente dieser Gruppe.
2. Die Angabe der USAGE-Klausel eines Datenelements darf nicht im Widerspruch stehen zur Angabe der USAGE-Klausel der Gruppe, zu der das Datenelement gehört.

3. Ein Datenelement, das mit USAGE BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, COMPUTATIONAL-5 oder PACKED-DECIMAL beschrieben wurde, stellt einen Wert dar, der in arithmetischen Operationen verwendet wird, und muss deshalb numerisch sein. Wird für eine Datengruppe eine dieser Angaben gemacht, so beziehen sich diese Angaben nur auf die Datenelemente dieser Gruppe; die Datengruppe selbst darf bei Rechenoperationen nicht verwendet werden.
4. Die USAGE-Klausel ist für die Verwendung eines Datenelementes ohne Bedeutung. Einige Anweisungen des Prozedurteils können jedoch die für ihre Operanden zulässigen USAGE-Klausen einschränken.
5. Die interne Darstellung der numerischen Datenfelder ist in Tabelle 17 im Abschnitt "COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe" gezeigt.

7.3.23 DISPLAY-Angabe

Syntaxregeln

1. Die verschiedenen Arten der Datenelemente mit der DISPLAY-Angabe unterscheiden sich durch die verschiedenen Kategorien der Maskenzeichenfolge in der PICTURE-Klausel.

Allgemeine Regeln

1. Die DISPLAY-Angabe zeigt an, dass das Datenfeld im EBCDIC-Format abgespeichert wird, d.h. jede Zeichenstelle ist durch 1 Byte dargestellt.
2. Externe dezimale Datenfelder werden intern wie folgt dargestellt:

   Jede Ziffer einer Zahl wird durch ein Byte dargestellt. Die vier höchstwertigen Bits eines jeden Bytes stellen den Zonenteil. Der Zonenteil des niedrigstwertigen bzw. höchstwertigen (je nach SIGN-Klausel) Bytes enthält das Vorzeichen (falls ein solches vorhanden ist). Die vier niedrigstwertigen Bits enthalten den Wert der Ziffer.

   Die maximale Länge eines externen dezimalen Datenfeldes ist 31 Ziffern.
3. Externe Gleitpunktdatenfelder bestehen aus einer Mantisse, die den dezimalen Teil der Zahl darstellt, und einem Exponenten, der die Basis 10 hat.

   Der Wert eines externen Gleitpunktdatenfeldes errechnet sich aus der Multiplikation der Mantisse mit dem Exponenten, Mantisse $*10^{Exponent}$.
   Der Absolutbetrag eines Gleitpunktdatenfeldes muss größer als $5.4 * 10^{-79}$ sein und darf $7.2 * 10^{75}$ nicht übersteigen.

   Wird ein externes Gleitpunktdatenfeld als ein numerischer Operand verwendet, so wird es zur Programmausführungszeit geprüft und in ein internes Gleitpunkdatenfeld umgewandelt. In dieser Form wird es in arithmetischen Operationen verwendet (siehe die Angaben zu COMPUTATIONAL-1 und COMPUTATIONAL-2).

Beispiel 7-30

Datenformate für USAGE IS DISPLAY

Datenkategorie	Wert	Masken-zeichenfolge	Interne Darstellung*)									
alphabetisch	ABCD	AAAA	C1	C2	C3	C4						
alphanumerisch	A1B2	XXXX	C1	F1	C2	F2						
alphanumerisch-druckaufbereitet	123AB	XXBXXX	F1	F2	40	F3	C1	C2				
numerisch-druckaufbereitet	54321	99,999	F5	F4	6B	F3	F2	F1				
numerisch												
extern dezimal	+1234	9999	F1	F2	F3	F4						
	+6879	S9999	F6	F8	F7	C9						
	-6879	S9999	F6	F8	F7	D9						
extern Gleitpunkt	6879	+99.99E-99	4E	F6	F8	4B	F7	F9	C5	40	F0	F2
	.6879	+99.99E-99	4E	F6	F8	4B	F7	F9	C5	60	F0	F2

*) Jedes Kästchen stellt ein Byte dar.

7.3.24 NATIONAL-Angabe

Syntaxregeln

1. Ein Datenelement, das eine USAGE NATIONAL-Klausel enthält, oder einer Datengruppe mit USAGE NATIONAL-Klausel untergeordnet ist, muss mit seiner PICTURE Zeichenmaske ein nationales Datenfeld beschreiben.
2. Die GROUP-USAGE-Klausel darf nicht gleichzeitig mit der NATIONAL-Angabe verwendet werden.

Allgemeine Regeln

1. Die NATIONAL-Angabe zeigt an, dass das Datenfeld im UTF-16-Format abgespeichert wird, d.h. jede Zeichenstelle ist durch 2 Bytes dargestellt.

> ¡ Für COBOL sind alle nationalen Zeichen gleich lang. „Surrogate Pairs“ werden daher als 2 Zeichen betrachtet.

Beispiel 7-31

Datenformat für USAGE IS NATIONAL

Datenkategorie	Wert	Maskenzeichenfolge	Interne Darstellung*)					
national	ABC	NNN	00	41	00	42	00	43

*)Jedes Kästchen stellt 1 Byte dar.

7.3.25 BINARY-Angabe oder COMPUTATIONAL-Angabe oder COMPUTATIONAL-5-Angabe

Syntaxregeln

1. Die Angaben beschreiben binäre Datenfelder.
2. Die PICTURE-Klausel eines binären Datenfeldes darf nur aus 9en, dem Rechenvorzeichen S, dem impliziten Dezimalpunkt V und einem oder mehreren P bestehen (siehe „PICTURE-Klausel").
3. Die Datenfelder werden in einem Halbwort, Wort oder Doppelwort abgespeichert und nur dann ausgerichtet, wenn die SYNCHRONIZED-Klausel angegeben wurde.

Allgemeine Regeln

1. Wird ein mit USAGE IS BINARY beschriebenes Datenfeld als Empfangsfeld verwendet, so wird geprüft, ob der in dieses Datenfeld zu übertragende Wert den nach der Maskenzeichenfolge der PICTURE-Klausel maximal möglichen Wert übersteigt. Ist dies der Fall, so wird der Wert durch Verkürzung angepasst. Diese Prüfung und eventuelle Verkürzung wird für ein Empfangsfeld, das mit USAGE IS COMPUTATIONAL oder COMPUTATIONAL-5 beschrieben ist, nicht durchgeführt.
2. In einer DISPLAY-Anweisung und zur Überprüfung der Überlaufbedingung wird auch bei Feldern mit USAGE IS COMPUTATIONAL oder COMPUTATIONAL-5 die in der PICTURE-Klausel angegebene Maskenzeichenfolge berücksichtigt.
3. Die Speicherbelegung für binäre Datenfelder variiert, abhängig von der Anzahl der Dezimalziffern, die in der PICTURE-Klausel angegeben wurde, wie folgt:

Dezimale Ziffern in der PICTURE-Klausel	Speicherbelegung in Bytes	Ausrichtung
1-4 5-9 10-18	2 4 8	Halbwort Wort Wort
19-31	16	Wort

4. Das am weitesten links stehende Bit eines binären Datenfeldes stellt das Rechenvorzeichen, die restlichen Bits stellen den Wert dar.

Beispiele für die BINARY- bzw. COMPUTATIONAL- bzw. COMPUTATIONAL-5-Angabe siehe Tabelle 17, „Interne Darstellung von numerischen Datenfeldern“ im Abschnitt "COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe".

7.3.26 COMPUTATIONAL-1-Angabe

Syntaxregeln

1. Diese Angabe beschreibt interne Gleitpunktdatenfelder. Ein solches Datenfeld ist gleichbedeutend mit einem externen Gleitpunktdatenfeld bezüglich seiner Kapazität und seinem Verwendungszweck (siehe „Datenklassen“ auf "Konzept der maschinenunabhängigen Datenbeschreibung").
2. Für ein COMPUTATIONAL-1-Datenfeld darf keine PICTURE-Klausel angegeben werden.
3. Die COMPUTATIONAL-1-Angabe zeigt an, dass ein Datenfeld in Gleitpunktformat mit einfacher Genauigkeit abgespeichert wird.
4. Ein COMPUTATIONAL-1-Datenfeld hat eine Länge von 4 Bytes und wird an Wortgrenze ausgerichtet, falls die SYNCHRONIZED-Klausel angegeben wurde.

Allgemeine Regeln

1. Ein COMPUTATIONAL-1-Datenfeld wird im Speicher wie folgt dargestellt:

   S ist hier das Vorzeichen der Mantisse.

   Charakteristik = Exponent + 32

2. Ein internes Gleitpunktdatenfeld einfacher Genauigkeit erlaubt eine Darstellung mit einer Genauigkeit von 7 Dezimalziffern.
3. Für den Wert, der in einem COMPUTATIONAL-1 Datenfeld dargestellt werden kann, gilt:

   Wert = 0 oder der Absolutbetrag des Wertes liegt zwischen $5.4 * 10^{-79}$ und $7.2 * 10^{75}$.

Beispiele für die COMPUTATIONAL-1-Angabe siehe Tabelle 17, „Interne Darstellung von numerischen Datenfeldern“ im Abschnitt "COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe".

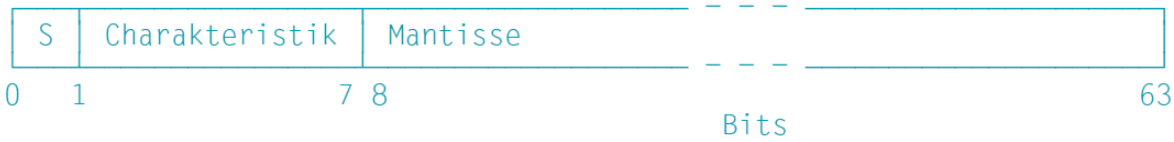
7.3.27 COMPUTATIONAL-2-Angabe

Syntaxregeln

1. Diese Angabe beschreibt interne Gleitpunktdatenfelder. Ein solches Datenfeld ist gleichbedeutend mit einem externen Gleitpunktdatenfeld bezüglich seiner Kapazität und seinem Verwendungszweck (siehe „Datenklassen“ auf "Konzept der maschinenunabhängigen Datenbeschreibung").
2. Für ein COMPUTATIONAL-2-Datenfeld darf keine PICTURE-Klausel angegeben werden.
3. Die COMPUTATIONAL-2-Angabe zeigt an, dass ein Datenfeld im Gleitpunktformat mit doppelter Genauigkeit abgespeichert wird.
4. Ein COMPUTATIONAL-2-Datenfeld hat eine Länge von 8 Bytes und wird an Doppelwortgrenze ausgerichtet, falls die SYNCHRONIZED-Klausel angegeben wurde.

Allgemeine Regeln

1. Ein COMPUTATIONAL-2-Datenfeld wird im Speicher wie folgt dargestellt:



   S ist hier das Vorzeichen der Mantisse.

   Charakteristik = Exponent + 64

2. Ein internes Gleitpunktdatenfeld doppelter Genauigkeit erlaubt eine Darstellung mit einer Genauigkeit von 16 Dezimalziffern.
3. Für den Wert, der in einem COMPUTATIONAL-2-Datenfeld dargestellt werden kann, gilt:

   Wert = 0 oder der Absolutbetrag des Wertes liegt zwischen $5.4 * 10^{-79}$ und $7.2 * 10^{75}$.

Beispiele für die COMPUTATIONAL-2-Angabe siehe Tabelle 17 im Abschnitt "COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe".

7.3.28 COMPUTATIONAL-3-Angabe oder PACKED-DECIMAL-Angabe

Syntaxregeln

1. Die Angaben von COMPUTATIONAL-3 und PACKED-DECIMAL sind gleichbedeutend.
2. Die Angaben zeigen an, dass das Datenfeld in internem dezimalem Format (also in gepackter Form) abgespeichert wird.
3. Die PICTURE-Klausel eines COMPUTATIONAL-3- bzw. PACKED-DECIMAL-Datenfeldes darf nur aus 9en, dem Rechenvorzeichen S, dem Rechendezimalpunkt V und einem oder mehreren P's bestehen (siehe „PICTURE-Klausel").

Allgemeine Regel

1. Interne dezimale Datenfelder werden mit 2 Ziffern pro Byte dargestellt; das Vorzeichen ist in den vier niedrigstwertigen Bits des niedrigstwertigen Bytes enthalten.

   Für interne dezimale Datenfelder, deren PICTURE-Klausel kein S enthält, entspricht die Darstellung dem Absolutwert der Zahl.

Beispiele für die COMPUTATIONAL-3- bzw. PACKED-DECIMAL-Angabe siehe Tabelle 17.

Form	Masken-zeichen-folge	USAGE- und SIGN-Angabe	Wert in externer Darstellung	Wert in interner Darstellung [4]	Anzahl benötigter Bytes
Extern dezimal (entpackt)	9999	DISPLAY	1234	F1F2F3F4	1 Byte / Ziffer
	S9999		+1234	F1F2F3C4 [1][2]	
	S9999		-1234	F1F2F3D4 [1][2]	
	S9999	DISPLAY	1234+	F1F2F3C4	
		SIGN TRAILING	1234 -	F1F2F3D4	
	S9999	DISPLAY	1234+	F1F2F3F44E	+ 1 Byte für Vorzeichen
		SIGN TRAILING SEPARATE	1234 -	F1F2F3F460	
	S9999	DISPLAY	+1234	C1F2F3F4	
		SIGN LEADING	-1234	D1F2F3F4	
	S9999	DISPLAY	+1234	4EF1F2F3F4	+ 1 Byte für Vorzeichen
		SIGN LEADING SEPARATE	- 1234	60F1F2F3F4	
Intern dezimal (gepackt)	9999	COMP-3 oder PACKED-DECIMAL	+1234	01234F [2]	2 Ziffern pro Byte, bis auf das niedrigstwertige Byte, das eine Ziffer und das
	9999		-1234	01234F [2]	
	S9999		+1234	01234C [2]	

	S9999		-1234	01234D [2]	Vorzeichen enthält
Binär	S9999	BINARY oder COMP oder COMP-5	+1234	04D2	2 bei 1-4 Ziffern
					4 bei 5-9 Ziffern
					8 bei 10-18 Ziffern
					16 bei 19-31 Ziffern
	S9999		- 1234	FB2E	2
ExternGleitpunkt	+99.99E-99	DISPLAY	+12.34E+2	4EF1F26BF3F4C540F0F2	1 Byte pro Zeichen
InternGleitpunkt	KeineAngabeerlaubt	COMP-1	+12.34E+2	434D2000	4
	KeineAngabeerlaubt	COMP-2	- 12.34E-2	C01F972474538EF3	8

Tabelle 17: Interne Darstellung von numerischen Datenfeldern

[1] Byte pro Ziffer, bis auf das niedrigwertigste Byte, das im ersten Halbbyte das Vorzeichen, im zweiten Halbbyte die letzte Ziffer enthält

[2] Darstellung des Vorzeichens

F =nichtabdruckbares Pluszeichen (wird als absoluter Wert betrachtet)
C =interne Darstellung des Pluszeichens
D =interne Darstellung des Minuszeichens

[3] siehe Regeln für Datenfelder

[4] jedes Zeichen (Buchstabe/Ziffer) stellt ein halbes Byte dar

7.3.29 INDEX-Angabe

Ein Datenelement, das mit der USAGE-Klausel mit INDEX-Angabe beschrieben ist, heißt Indexdatenfeld. Solch ein Datenfeld (das nicht unbedingt mit einer Tabelle verknüpft zu sein braucht) kann benutzt werden, um Werte von Indizes für eine spätere Verwendung sicherzustellen. Einem Indexdatenfeld wird durch die SET-Anweisung der Wert eines Index zugewiesen. Der Wert eines Indexdatenfeldes ist nicht die Tabellenelementnummer.

Allgemeine Regeln

1. Die USAGE-Klausel mit INDEX-Angabe kann auf jeder Stufe angegeben werden. Ist eine Datengruppe mit einer USAGE-Klausel mit INDEX-Angabe beschrieben, sind alle Datenelemente in der Datengruppe Indexdatenfelder, die Datengruppe selbst ist kein Indexdatenfeld.
2. Auf ein Indexdatenfeld kann nur in einer SEARCH- oder SET-Anweisung, einer Vergleichsbedingung, der USING-Angabe der PROCEDURE DIVISION-Überschrift oder der USING-Angabe einer CALL-Anweisung direkt Bezug genommen werden.
3. Ein Indexdatenfeld darf keine Bedingungsvariable sein.
4. Ein Indexdatenfeld kann Teil einer Datengruppe sein, auf die in einer MOVE-Anweisung oder in einer Ein-/Ausgabe-Anweisung Bezug genommen wird. Der Inhalt der Indexdatenfelder wird jedoch bei Ausführung solcher Anweisungen nicht konvertiert.
5. SYNCHRONIZED-, PICTURE- oder VALUE-Klauseln können zur Erklärung von Gruppen oder Datenelementen nicht verwendet werden, die mit der USAGE-Klausel mit INDEX-Angabe beschrieben sind. Der Compiler gestattet jedoch die Anwendung der SYNCHRONIZED-Klausel mit der USAGE-Klausel mit INDEX-Angabe.

Beispiel 7-32

```
02  ALPHA PICTURE X(9) OCCURS 5 INDEXED BY A-NAME.
...
77  A-INDEX USAGE IS INDEX.
...
    SET A-NAME TO 3.
    ...
    SET A-INDEX TO A-NAME.
```

Hier wird das Indexdatenfeld A-INDEX auf den aktuellen Wert des Index A-NAME gesetzt, d.h. Tabellenelementnummer (3) minus 1 mal Länge des Eintrags (9) = 18.

7.3.30 OBJECT REFERENCE-Angabe

Syntaxregeln

1. Die USAGE OBJECT REFERENCE-Klausel sollte nur auf Stufe 01 oder innerhalb einer Typerklärung mit dem Attribut STRONG angegeben werden. Abweichungen davon werden als Erweiterung zugelassen.
2. Die USAGE OBJECT REFERENCE-Klausel darf in Datendefinitionen nicht auf Gruppenebene angegeben werden.
3. Die USAGE OBJECT REFERENCE-Klausel darf nicht in der FILE SECTION angegeben werden.
4. Die ACTIVE -CLASS- Angabe ist nur in einer Fabrik(Factory)-, Objekt- oder Methodendefinition zulässig.
5. Die USAGE OBJECT REFERENCE-Klausel darf nicht in Strukturen angegeben werden, in denen auf Stufe 01 die EXTERNAL- oder DYNAMIC-Klausel verwendet wird.
6. Für Datenfelder, die mit einer USAGE OBJECT REFERENCE-Klausel definiert sind, darf keine VALUE-Klausel angegeben werden.
7. Datenfelder, für die eine USAGE OBJECT REFERENCE-Klausel gilt, dürfen weder eine REDEFINES-Klausel haben, noch darf direkt von solch einer Klausel auf sie Bezug genommen werden. Außerdem dürfen Datenfelder auch nicht indirekt in einer redefinierten oder redefinierenden Struktur enthalten sein.
8. Datenfelder und Strukturen, für die eine USAGE OBJECT REFERENCE-Klausel gilt, dürfen nicht mit einem anderen Namen benannt werden (RENAMES-Klausel).
9. Für Gruppen, die Datenfelder mit einer USAGE OBJECT REFERENCE-Klausel definiert haben, darf auf Gruppenebene keine VALUE-Klausel angegeben werden.
10. Für Datenfelder mit einer USAGE OBJECT REFERENCE-Klausel, darf kein Bedingungsname (Stufennummer 88) angegeben werden.
11. Das in der KEY-Angabe einer OCCURS-Klausel angegebene Datenfeld, darf nicht mit einer USAGE OBJECT REFERENCE-Klausel definiert sein.

Allgemeine Regeln

Ein mit der USAGE OBJECT REFERENCE-Klausel beschriebenes Datenfeld ist eine Objektreferenz. Es gelten folgende Regeln:

1. Wenn keine der wahlweisen Angaben definiert ist, ist dieses Datenelement eine universelle Objektreferenz. Sie kann auf jedes mögliche Objekt verweisen.
2. interfacename bzw. klassenname müssen die Namen der umfassenden Schnittstellen- bzw. Klassendefinition sein oder im REPOSITORY Paragrafen entsprechend aufgeführt sein.
3. Ist interfacename-1 angegeben, dann verweist die Objektreferenz auf ein Objekt, dessen Schnittstelle zu interface-1 konform ist.
4. Ist klassenname-1 angegeben, so verweist diese Objektreferenz auf ein Objekt der Klasse, die durch klassenname -1 bezeichnet ist oder auf eine Unterklasse davon. Es gelten hierfür folgende Regeln:
   a. Ist die ONLY nicht angegeben, aber FACTORY, so verweist diese Objektreferenz auf das **Fabrikobjekt** der entsprechenden Klasse bzw. einer Unterklasse davon. Ist FACTORY nicht angegeben, so verweist die Objektreferenz auf ein **Objekt** dieser Klasse bzw. einer Unterklasse davon.
   b. Ist die ONLY angegeben, gelten dieselben Regeln wie unter a), außer dass kein Bezug auf eine Unterklasse möglich ist.
5. Ist ACTIVE-CLASS angegeben, dann verweist die Objektreferenz auf ein Objekt der Klasse des Objekts, für das die Methode aufgerufen wurde, in der die Objektreferenz definiert ist. Ist FACTORY angegeben, bezieht sich die Objektreferenz auf das **Fabrikobjekt** dieser Klasse, andernfalls auf ein **Objekt** dieser Klasse.

7.3.31 POINTER-Angabe

Die POINTER-Angabe in der USAGE-Klausel dient dazu, Daten der Kategorie datenzeiger zu definieren.

Syntaxregeln

1. Die USAGE POINTER-Klausel sollte nur auf Stufe 01 oder innerhalb einer Typerklärung mit dem Attribut STRONG angegeben werden. Abweichungen davon werden als Erweiterung zugelassen.
2. Die USAGE POINTER-Klausel darf in Definitionen nicht auf Gruppenebene angegeben werden.
3. Die USAGE POINTER-Klausel darf nicht in der FILE SECTION angegeben werden.
4. Die USAGE POINTER-Klausel darf nicht in Strukturen angegeben werden, in denen auf Stufe 01 die EXTERNAL- oder DYNAMIC-Klausel verwendet wird.
5. Für Datenfelder, die mit einer USAGE POINTER-Klausel definiert sind, darf keineVALUE-Klausel angegeben werden.
6. Datenfelder, für die eine USAGE POINTER-Klausel gilt, dürfen weder eineREDEFINES-Klausel haben, noch darf direkt von solch einer Klausel auf sie Bezug genommen werden. Außerdem dürfen solche Datenfelder auch nicht indirekt in einer redefinierten oder redefinierenden Struktur enthalten sein.
7. Datenfelder und Strukturen, für die eine USAGE POINTER-Klausel gilt, dürfen nicht mit einem anderen Namen benannt werden (RENAMES-Klausel).
8. Für Gruppen, die Datenfelder mit einer USAGE POINTER-Klausel definiert haben, darf auf Gruppenebene keine VALUE-Klausel angegeben werden.
9. Für Datenfelder mit einer USAGE POINTER-Klausel darf kein Bedingungsname (Stufennummer 88) angegeben werden.
10. Das in der KEY-Angabe einer OCCURS-Klausel angegebene Datenfeld darf nicht mit einer USAGE POINTER-Klausel definiert sein.
11. Ist typname-1 angegeben, so muss die Datenerklärung auch eine TYPEDEF-Klausel enthalten.

Allgemeine Regel

1. Ein mit der USAGE POINTER-Klausel beschriebenes Datenfeld ist ein 4 Byte langes Datenfeld der Kategorie datenzeiger, das die Adresse eines Datenfeldes enthalten kann. Der Anfangswert entspricht NULL. Ist typname-1 angegeben, handelt es sich um einen typbezogenen Zeiger. Ein typbezogener Zeiger hat als Inhalt nur den vordefinierten Wert NULL oder die Adresse eines typisierten Datenfeldes vom Typ typname-1.

7.3.32 PROGRAM-POINTER-Angabe

Die PROGRAM-POINTER-Angabe in der USAGE-Klausel dient dazu, Daten der Kategorie programzeiger zu definieren.

Syntaxregeln

1. Die USAGE PROGRAM-POINTER-Klausel sollte nur auf Stufe 01 angegeben werden. Abweichungen davon werden als Erweiterung zugelassen.
2. Die USAGE PROGRAM-POINTER-Klausel darf in Definitionen nicht auf Gruppenebene angegeben werden.
3. Die USAGE PROGRAM-POINTER-Klausel darf nicht in der File Section angegeben werden.
4. Die USAGE PROGRAM-POINTER-Klausel darf nicht in Strukturen angegeben werden, in denen auf Stufe 01 die EXTERNAL- oder DYNAMIC-Klausel verwendet wird.
5. Für Datenfelder, die mit einer USAGE PROGRAM-POINTER-Klausel definiert sind, darf keine VALUE-Klausel angegeben werden.
6. Datenfelder, für die eine USAGE PROGRAM-POINTER-Klausel gilt, dürfen weder eine REDEFINES-Klausel haben, noch darf direkt von solch einer Klausel auf sie Bezug genommen werden. Außerdem dürfen solche Datenfelder auch nicht indirekt in einer redefinierten oder redefinierenden Struktur enthalten sein.
7. Datenfelder und Strukturen, für die eine USAGE PROGRAM-POINTER-Klausel gilt, dürfen nicht mit einem anderen Namen benannt werden (RENAMES-Klausel).
8. Für Gruppen, die Datenfelder mit einer USAGE PROGRAM-POINTER-Klausel definiert haben, darf auf Gruppenebene keine VALUE-Klausel angegeben werden.
9. Für Datenfelder mit einer USAGE PROGRAM-POINTER-Klausel, darf kein Bedingungsname (Stufennummer 88) angegeben werden.
10. Das in der KEY-Angabe einer OCCURS-Klausel angegebene Datenfeld darf nicht mit einer USAGE PROGRAM-POINTER-Klausel definiert sein.

Allgemeine Regel

1. Ein mit der USAGE PROGRAM-POINTER-Klausel beschriebenes Datenfeld ist 4 Byte langes ein Datenfeld der Kategorie programmzeiger, das die Adresse eines Programms enthalten kann. Der angenommene Anfangswert entspricht NULL.

7.3.33 VALUE-Klausel

Funktion

In der WORKING-STORAGE SECTION und der LOCAL-STORAGE SECTION bestimmt die VALUE-Klausel den Anfangswert eines Datenfeldes. In der REPORT SECTION
bestimmt die VALUE-Klausel den Wert eines druckfähigen Datenfeldes. In der WORKING-STORAGE SECTION, der LOCAL-STORAGE SECTION sowie in der FILE SECTION und der LINKAGE SECTION haben die VALUE-Klauseln bei der Ausführung einer INITIALIZE-Anweisung mit der BY VALUE-Angabe eine Bedeutung.

Bei Bedingungsnamen bestimmt die VALUE-Klausel den Wert oder einen Bereich von Werten, die diesem Bedingungsnamen zugeordnet sind.

Format 1 der VALUE-Klausel wird angegeben, um den Anfangswert eines Datenfeldes oder den Wert eines druckfähigen Datenfeldes zu bestimmen oder den Wert des Sendefelds für die INITIALIZE-Anweisung mit der BY VALUE Angabe festzulegen.

Format 2 der VALUE-Klausel wird nur angegeben, um den Wert oder einen Bereich von Werten, der einem Bedingungsnamen zugeordnet ist, zu bestimmen. Durch den wahlweisen Zusatz „WHEN SET TO FALSE IS LITERAL-4“ wird der Wert festgelegt, auf den das zugehörige Datum bei Ausführung der Anweisung „SET bedingungsname TO FALSE“ gesetzt wird.

Format 3 dient dazu, Tabellenelemente mit Anfangswerten zu versehen.

Format 1

```
VALUE IS literal
```

Syntaxregeln

1. Das angegebene Literal kann durch eine figurative Konstante ersetzt werden.
2. Ein numerisches Literal muss eine Größe haben, die innerhalb der in der PICTURE-Klausel angegebenen Stellenanzahl liegt, und darf keinen Wert haben, der das Abschneiden von Ziffern ungleich Null erfordern würde.
3. Ein nichtnumerisches Literal darf die in der PICTURE-Klausel angegebene Größe nicht überschreiten.
4. Ist die Kategorie des Datenfeldes alphabetisch, alphanumerisch, alphanumerisch druckaufbereitet oder numerisch druckaufbereitet, muss das Literal in der VALUE-Klausel alphanumerisch sein.
   Die Länge alphanumerischer Literale in der VALUE-Klausel eines Datenelements darf nicht größer sein als die Länge, die durch eine explizit angegebene PICTURE-Klausel festgelegt ist.
   Die Länge alphanumerischer Literale in der VALUE-Klausel einer alphanumerischen Gruppe darf nicht größer sein als die Länge der Gruppe.
5. Ein numerisches Literal mit Vorzeichen muss einer PICTURE-Klausel mit numerischer Maskenzeichenfolge mit Vorzeichen zugeordnet sein.
6. Ist die Kategorie des Datenfeldes numerisch, muss das Literal in der VALUE-Klausel numerisch sein. Der Wert des Literals muss im Bereich der durch die PICTURE-Klausel beschriebenen Werte liegen. Alle Ziffern an Stellen, die einem Maskenzeichensymbol P entsprechen, müssen 0 sein.
7. Wird die VALUE-Klausel in einer Erklärung auf Gruppenebene angegeben, so muss das Literal eine figurative Konstante oder ein nichtnumerisches Literal sein. Innerhalb der Datengruppe darf die VALUE-Klausel für keine der untergeordneten Stufen angegeben werden.
8. Die VALUE-Klausel darf nicht für eine Datengruppe angegeben werden, deren Datenfelder mit JUSTIFIED, SYNCHRONIZED oder USAGE (außer USAGE DISPLAY oder USAGE NATIONAL) beschrieben sind.
9. Für eine stark typisierte Datengruppe darf keine VALUE-Klausel angegeben werden.
10. Die VALUE-Klausel darf für externe Gleitpunktdatenfelder nicht angegeben werden.

11. Ist die Kategorie des Datenfeldes national, müssen alle Literale in der VALUE-Klausel national sein.
    Die Länge nationaler Literale in der VALUE-Klausel eines Datenelements darf nicht größer sein als die Länge, die durch eine explizit angegebene PICTURE-Klausel festgelegt ist.
    Die Länge nationaler Literale in der VALUE-Klausel einer nationalen Gruppe darf nicht größer sein als die Länge der Gruppe.

Allgemeine Regeln

1. Wenn eine VALUE-Klausel in der Erklärung eines Datenfeldes angegeben ist, das mit einem Datenfeld variabler Länge verknüpft ist, wird beim Setzen des Anfangswertes so vorgegangen, als hätte das Datenfeld der DEPENDING ON-Angabe der OCCURS-Klausel für das Datenfeld variabler Länge den maximalen Wert. Ein Datenfeld ist mit einem Datenfeld variabler Länge verknüpft, wenn einer der folgenden Fälle vorliegt:
   a. Es ist ein Gruppenfeld, das ein Datenfeld variabler Länge enthält.
   b. Es ist ein Datenfeld variabler Länge.
   c. Es ist ein Datenfeld, das einem Datenfeld variabler Länge untergeordnet ist.
2. Wird die VALUE-Klausel in einer Erklärung auf Gruppenebene angegeben, wird der ganze Bereich der Datengruppe auf den Anfangswert gesetzt, ohne Rücksicht auf die einzelnen Datenelemente oder Datengruppen, die in der Datengruppe enthalten sind.
3. Das Literal wird so in dem Datenfeld ausgerichtet, als wäre das Datenfeld alphanumerisch beschrieben worden. Dabei hat die JUSTIFIED-Klausel keinen Einfluss und es findet keine Editierung statt.
4. Ein Datenfeld wird auf einen Anfangswert gesetzt, unabhängig davon, ob eine BLANK WHEN ZERO-Klausel oder JUSTIFIED-Klausel angegeben wurde.
5. Eine VALUE-Klausel in einer Datenerklärung, die eine OCCURS-Klausel enthält oder einer solchen Datenerklärung untergeordnet ist, bewirkt, dass jede Wiederholung des Datenfeldes den angegebenen Anfangswert erhält.
6. In der WORKING-STORAGE SECTION und in der LOCAL-STORAGE SECTION kann die VALUE-Klausel angegeben werden, um den Anfangswert irgendeines Datenfeldes zu bestimmen; in diesem Fall bewirkt die Klausel, dass das Datenfeld zu Beginn des Programmablaufs den angegebenen Wert annimmt. Wird die VALUE-Klausel in der Beschreibung eines Datenfeldes nicht angegeben, so ist der Anfangswert dieses Datenfeldes undefiniert.
7. Ein Datenfeld der Klassen objekt oder zeiger darf keine VALUE-Klausel haben; es ist immer mit NULL (undefiniert) initialisiert.

Beispiel 7-33

```
77 FELD PICTURE IS AA VALUE IS "AA"
```

Der Inhalt von FELD ist hier auf den Anfangswert AA gesetzt.

Format 2

```
{VALUE | VALUES} {IS | ARE} {literal-1 [{THRU | THROUGH} literal-2]} ...
                            [WHEN SET TO FALSE IS literal-4]
```

Syntaxregeln

1. Die VALUE-Klausel im Format 2 darf nur in Verbindung mit Bedingungsnamen (Stufennummer 88) verwendet werden.
2. Die Stufennummer 88 gilt für Erklärungen von Bedingungsnamen, die einer Bedingungsvariablen zugeordnet sind; diese Erklärungen nennt man Bedingungsnamenerklärungen. Eine Bedingungsvariable ist ein Datenfeld, das von einer oder mehreren Bedingungsnamenerklärungen gefolgt ist. Ein Bedingungsname beschreibt einen Wert oder einen Bereich von Werten, die zur Ausführungszeit des

Programms als Inhalt der Bedingungsvariable abgefragt werden sollen. Ein Bedingungsname kann während der Programmausführung „wahr“ oder „falsch“ sein.
Ein Bedingungsname ist kein Datenfeld und benötigt keinen Speicherplatz (siehe „Bedingungsnamen-Bedingung").

Bedingungsnamen dürfen nicht auf ein Datenfeld folgen, für das eine ANY LENGTH- Klausel angegeben ist. Bedingungsnamen dürfen nicht auf eine Typdefinition mit der Angabe STRONG oder auf eine Datengruppe, die einer Typdefinition mit der Angabe STRONG untergeordnet ist, folgen.

3. Die angegebenen Literale können durch figurative Konstanten ersetzt werden.
4. Alle numerischen Literale müssen die Größe haben, die innerhalb der in der PICTURE-Klausel des zugehörigen Datenelements (Bedingungsvariable) angegebenen Stellenzahl liegt und dürfen keinen Wert haben, der das Abschneiden von Ziffern ungleich Null erfordern würde.
5. Nichtnumerische Literale dürfen die in der PICTURE-Klausel des zugehörigen Datenelements (Bedingungsvariable) angegebene Größe nicht überschreiten.
6. Ist die Kategorie des Datenfeldes alphabetisch, alphanumerisch, alphanumerisch druckaufbereitet oder numerisch druckaufbereitet müssen alle Literale in der VALUE-Klausel alphanumerisch sein.

   Die Länge alphanumerischer Literale in der VALUE-Klausel eines Datenelements darf nicht größer sein als die Länge, die durch eine explizit angegebene PICTURE-Klausel festgelegt ist.
7. Ist die Kategorie des Datenfeldes numerisch, müssen alle Literale in der VALUE-Klau-sel numerisch sein. Der Wert des Literals muss im Bereich der durch die PICTURE-Klausel beschriebenen Werte liegen. Alle Ziffern an Stellen, die einem Maskenzeichensymbol P entsprechen, müssen 0 sein.
8. Ein numerisches Literal mit Vorzeichen muss einer PICTURE-Klausel mit numerischer Maskenzeichenfolge mit Vorzeichen zugeordnet sein.
9. Wird die THRU-/THROUGH-Angabe verwendet, muss das Literal vor THRU/THROUGH kleiner als das Literal nach THRU/THROUGH sein.
10. Die THRU-/THROUGH-Angabe weist dem angegebenen Bedingungsnamen einen Bereich von Werten zu.
11. literal-4 darf keinem literal-1 gleich sein.
12. Für jedes Paar literal-1 und literal-2 muss gelten: literal-4 muss kleiner als literal-1 oder größer als literal-2 sein.
13. Ist die Kategorie des Datenfeldes national, müssen alle Literale in der VALUE-Klausel national sein.
    Die Länge nationaler Literale in der VALUE-Klausel eines Datenelements darf nicht größer sein als die Länge, die durch eine explizit angegebene PICTURE-Klausel festgelegt ist.

Allgemeine Regeln

1. Die VALUE-Klausel darf für externe Gleitpunktdatenfelder und für Daten der Klassen objekt oder zeiger nicht angegeben werden.
2. Die VALUE-Klausel darf nicht für Datenfelder angegeben werden, deren Größe explizit oder implizit variabel ist.
3. Die VALUE-Klausel darf nicht im Widerspruch zu den sonstigen Klauseln in der Datenerklärung eines Datenfeldes oder in der Datenerklärung innerhalb der Hierarchie eines Datenfeldes stehen. Folgende Regeln gelten:
4. Der Wert wird so in dem Datenfeld ausgerichtet, als wäre das Datenfeld alphanumerisch beschrieben worden.
5. Format 2 der VALUE-Klausel darf nur in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und in der LINKAGE SECTION angegeben werden. Sie darf nicht in der REPORT SECTION angegeben werden.
6. Die FALSE-Angabe in der VALUE-Klausel ist nur dann von Bedeutung, wenn der Bedingungsname in einer SET-TO-FALSE-Anweisung benutzt wird.

Beispiel 7-34

```
02 STAEDTE PICTURE 9.
   88  BERLIN VALUE 1.
   88  HAMBURG VALUE 2.
   88  MUENCHEN VALUE 3.
   88  KOELN VALUE 4.
```

STAEDTE ist hier die Bedingungsvariable, und BERLIN, HAMBURG, MUENCHEN und KOELN sind die Bedingungsnamen.
Würde die Anweisung IF MUENCHEN GO TO TEST-C im Befehlsteil geschrieben werden, so würde der Wert der Bedingungsvariablen STAEDTE mit 3 verglichen werden; diese Anweisung wäre gleichbedeutend mit der Anweisung
IF STAEDTE IS EQUAL TO 3 GO TO TEST-C.

Beispiel 7-35

```
02  ZEITALTER PICTURE 99.
    88  ZWANZIGER VALUE 20 THRU 29.
    88  DREISSIGER VALUE 30 THRU 39.
```

Würde die Anweisung IF ZWANZIGER... im Befehlsteil geschrieben werden, so würde der Wert der Bedingungsvariablen ZEITALTER auf 20 und 29 verglichen werden. Diese Anweisung wäre gleichbedeutend mit IF ZEITALTER NOT < 20 AND NOT > 29...

Beispiel 7-36

```
02 WOCHEN-TAG PIC X(3).
   88  ANFANG-WOCHE VALUE "MON" "DIE" "MIT".
   88  ENDE-WOCHE   VALUE "DON" "FRE".
   88  FREIER-TAG   VALUE "SAM" "SON".
```

Wird die Anweisung IF ANFANG-WOCHE... im Befehlsteil geschrieben, dann wird die Bedingungsvariable WOCHEN-TAG mit „MON“, „DIE“ und „MIT“ verglichen. Diese Anweisung wäre gleichbedeutend mit IF WOCHEN-TAG IS EQUAL TO "MON" OR "DIE" OR "MIT" ...

Format 3

```
{{VALUE | VALUES} [FROM ({subskript-1}...)] [IS | ARE]
       {literal-2}... [REPEATED {ganzzahl-1 TIMES | TO END}]} ...
```

Syntaxregeln

1. Alle numerischen Literale in der VALUE-KLausel eines Datenfeldes müssen die Größe haben, die innerhalb der in der PICTURE-Klausel angegebenen Stellenanzahl liegt, und dürfen keinen Wert haben, der das Abschneiden von Ziffern ungleich Null erfordern würde.
2. Nichtnumerische Literale in der VALUE-Klausel eines Datenfeldes dürfen die in der PICTURE-Klausel angegebene Größe nicht überschreiten.
3. Ist die Kategorie des Datenfeldes alphabetisch, alphanumerisch, alphanumerisch druckaufbereitet oder numerisch druckaufbereitet, müssen alle Literale in der VALUE-Klausel alphanumerische Literale sein.
   Die Länge alphanumerischer Literale in der VALUE-Klausel eines Datenelements darf nicht größer sein als die Länge, die durch eine explizit angegebene PICTURE-Klausel festgelegt ist.

Die Länge alphanumerischer Literale in der VALUE-Klausel einer alphanumerischen Gruppe darf nicht größer sein als die Länge der Gruppe.

4. Wird die VALUE-Klausel in einer Erklärung auf Gruppenebene angegeben, so muss das Literal eine figurative Konstante oder ein nichtnumerisches Literal sein; in diesem Fall ist der ganze Bereich der Datengruppe auf den Anfangswert gesetzt, ohne Rücksicht auf die einzelnen Datenelemente oder Datengruppen, die in der Datengruppe enthalten sind. Innerhalb der Datengruppe darf die VALUE-Klausel für keine der untergeordneten Stufen angegeben werden.
5. Die VALUE-Klausel darf nicht für eine Datengruppe angegeben werden, deren Daten felder mit JUSTIFIED, SYNCHRONIZED oder USAGE (außer USAGE DISPLAY oder USAGE NATIONAL) beschrieben sind. -
6. Die Datenerklärung muss eine OCCURS-Klausel enthalten oder muss einer Datener klärung untergeordnet sein, die eine OCCURS-Klausel enthält.
7. Subskript-1 muss ein ganzzahliges Literal sein. Wenn alle Subskripte den Wert 1 haben, brauchen sie nicht angegeben zu werden; andernfalls müssen alle Subskripte, die sich auf ein einzelnes Tabellenelement beziehen, angegeben werden. -
8. Die Anzahl der mit Anfangswerten zu versehenden Tabellenelemente ist wie folgt fest gelegt:
   a. Wenn ganzzahl-1 nicht angegeben ist, gilt die Anzahl der Wiederholungen von literal-2.
   b. Wenn ganzzahl-1 angegeben ist, gilt die Anzahl der Wiederholungen von literal-2 multipliziert mit ganzzahl-1.

   Die Anzahl der mit Anfangswerten zu versehenden Tabellenelemente darf nicht die maximale Anzahl der Elemente vom Bezugspunkt bis zum Tabellenende überschreiten. -
9. Wenn mehrere VALUE-Klauseln vom Format 3 in einer Datenerklärung verwendet werden, gilt:
   a. Die TO END-Angabe darf nur einmal gemacht werden.
   b. Ein spezifiziertes Tabellenelement darf nur einmal berücksichtigt werden.
10. Ist die Kategorie des Datenfeldes national, müssen alle Literale in der VALUE-Klausel national sein.
    Die Länge nationaler Literale in der VALUE-Klausel eines Datenelements darf nicht größer sein als die Länge, die durch eine explizit angegebene PICTURE-Klausel festgelegt ist.
    Die Länge nationaler Literale in der VALUE-Klausel einer nationalen Gruppe darf nicht größer sein als die Länge der Gruppe.

Allgemeine Regeln

1. In einer Tabelle können alle Formate der VALUE-Klausel verwendet werden.
2. Wenn mehr als eine VALUE-Klausel sich auf dasselbe Tabellenelement beziehen, wird - innerhalb derselben Datenerklärung - derjenige Wert dem Tabellenelement zugewiesen, der von der letzten VALUE-Klausel der Datenerklärung vorgegeben wurde.
3. Eine VALUE-Klausel vom Format 3 belegt ein Tabellenelement mit dem Wert von literal-2. Das mit dem Anfangswert belegte Tabellenelement ist bezeichnet durch subskript-1. Zusammenhängende Tabellenelemente werden der Reihe nach initialisiert, je nach den aufeinanderfolgenden Werten von literal-2. Zusammenhängende Tabellenelemente beziehen sich auf das durch 1 inkrementierte Subskript, das die kleinste Tabellendimension repräsentiert.Wenn ein Bezug auf ein Subskript, bevor es inkrementiert wird, gleich ist mit der größten Anzahl von Bezügen, die in der zugehörigen OCCURS-Klausel angegeben wurde, wird dieses Subskript auf 1 gesetzt, und das Subskript für die nächste Tabellendimension wird um 1 vergrößert.
4. Wird REPEATED angegeben, werden alle Werte von literal-2 wiederverwendet in der Reihenfolge, in der sie angegeben wurden.Wird TO END angegeben, wird die Wiederverwendung durchgeführt, bis das Ende der Tabelle erreicht ist. Wenn ganzzahl-1 TIMES angegeben ist, werden alle Werte von literal-2 der Reihe nach wiederverwendet, abhängig von ganzzahl-1.Ist REPEATED nicht angegeben, werden die Werte von literal-2 nur einmal der Reihe nach verwendet.

5. Wenn eine VALUE-Klausel in der Erklärung eines Datenfeldes angegeben ist, das mit einem Datenfeld variabler Wiederholungen verknüpft ist, wird beim Setzen des Anfangswertes so vorgegangen, als hätte das Datenfeld der DEPENDING ON-Angabe der OCCURS-Klausel für das Datenfeld variable Wiederholung den Wert maximaler Wiederholungen. Ein Datenfeld ist mit einem Datenfeld variabler Wiederholungen verknüpft, wenn einer der folgenden Fälle gilt:
   a. Es ist ein Gruppenfeld, das ein Datenfeld variabler Wiederholungen enthält.
   b. Es ist ein Datenfeld variabler Wiederholungen.
   c. Es ist ein Datenfeld, das einem Datenfeld variabler Wiederholungen untergeordnet ist.
6. Die VALUE-Klausel darf nicht im Widerspruch zu den sonstigen Klauseln in der Datenerklärung eines Datenfeldes oder in der Datenerklärung innerhalb der Hierarchie eines Datenfeldes stehen. Folgende Regeln gelten:

   Ist die Kategorie des Datenfeldes numerisch, müssen alle Literale in der VALUE-Klau sel numerisch sein.

   Das Literal wird standardmäßig in dem Datenfeld ausgerichtet. Das Literal wird so in das Datenfeld ausgerichtet, als wäre das Datenfeld alphanume- risch beschrieben worden. Dabei hat die JUSTIFIED-Klausel keinen Einfluss und es findet keine Editierung statt.

   Ein Datenfeld wird auf einen Anfangswert gesetzt, unabhängig davon, ob eine BLANK WHEN ZERO-Klausel oder JUSTIFIED-Klausel angegeben wurde.

Example 7-37

```
IDENTIFICATION DIVISION.
  PROGRAM-ID. TAB.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
      TERMINAL IS T.
  DATA DIVISION.
 *****
  WORKING-STORAGE SECTION.
  01 FIELD1.
     02 A OCCURS 20.
       03 B OCCURS 4.
         49 PIC X(01)
            VALUE FROM (5 2) IS "1" "2" "3"
            REPEATED 4.
 *
  01 FIELD2.
     02 Z PIC 99.
     02 A OCCURS 1 TO 78 DEPENDING ON Z.
        49 PIC X VALUE "x".
 *
  01 FIELD3.
     02 A OCCURS 20
          VALUE FROM (1) IS "ab" "c"
          REPEATED 10 TIMES.
        03 B OCCURS 4.
          49 PIC X.
  PROCEDURE DIVISION.
  MAIN SECTION.
  P1.
      MOVE 78 TO Z.
      DISPLAY FIELD1 UPON T.
      DISPLAY FIELD2 UPON T.
```

```
        DISPLAY FIELD3 UPON T.
        STOP RUN.
```

Es ergibt sich folgende Belegung:

FIELD1:	B(5,2) = "1"	B(6,1) = "1"	B(7,1) = "2"	B(8,1) = "3"
	B(5,3) = "2"	B(6,2) = "2"	B(7,2) = "3"	
	B(5,4) = "3"	B(6,3) = "3"	B(7,3) = "1"	
		B(6,4) = "1"	B(7,4) = "2"	

Alle anderen Tabellenelemente sind nicht belegt.

FIELD2	78 mal "x"	
FIELD3:	A(1) = "ab'BLANK"BLANK"'	A(2) = "c'BLANK"BLANK"BLANK"'
	A(3) = "ab'BLANK"BLANK"'	A(4) = "c'BLANK"BLANK"BLANK"'
	...	...
	A(19) = "ab'BLANK"BLANK"'	A(20) = "c'BLANK"BLANK"BLANK"'

8 PROCEDURE DIVISION

In diesem Kapitel werden folgende Themen behandelt:

- EXIT PARAGRAPH-Anweisung
- EXIT PERFORM-Anweisung
- EXIT PROGRAM-Anweisung
- EXIT SECTION-Anweisung
- FREE-Anweisung
- GOBACK-Anweisung
- GO TO-Anweisung
- IF-Anweisung
- INITIALIZE-Anweisung
- INSPECT-Anweisung
- INVOKE-Anweisung
- MERGE-Anweisung
- MOVE-Anweisung
- MULTIPLY-Anweisung
- OPEN-Anweisung
- PERFORM-Anweisung
- RAISE-Anweisung
- READ-Anweisung
- RELEASE-Anweisung
- RESUME-Anweisung
- RETURN-Anweisung
- REWRITE-Anweisung
- SEARCH-Anweisung
- SET-Anweisung
- SORT-Anweisung
- START-Anweisung
- STOP-Anweisung
- STRING-Anweisung
- SUBTRACT-Anweisung
- UNSTRING-Anweisung
- USE-Anweisung
- WRITE-Anweisung

8.1 Allgemeine Beschreibung

Die PROCEDURE DIVISION eines Programms oder einer Methode enthält die wesentlichen Instruktionen zur Lösung eines Datenverarbeitungsproblems.

Die PROCEDURE DIVISION in einer Objekt-, Fabrik(Factory)- oder Schnittstellendefinition enthält die aufrufbaren Methoden (bei Schnittstellen nur die Methoden-Prototypen), die entsprechend für ein Objekt, ein Fabrikobjekt oder für eine Schnittstelle gelten.

COBOL-Instruktionen werden als Anweisungen geschrieben.

Eine **Anweisung** ist eine syntaktisch zulässige Kombination von Wörtern und Symbolen, die mit einem COBOL-Wort beginnt.

Beispiel für eine Anweisung:

```
MOVE A TO B
```

Mehrere Anweisungen können zur Bildung eines Programmsatzes zusammengestellt werden, Gruppen von Programmsätzen können Paragrafen bilden, die wiederum Kapitel bilden können.

Eine COBOL-Anweisung nimmt normalerweise auf vom Programmierer definierte Daten oder Prozeduren Bezug, indem sie Datennamen oder Prozedurnamen benutzt. Bezugnahmen auf benutzerdefinierte Namen müssen eindeutig sein (siehe unter „Kennzeichnung").

Eine logische Untermenge der Quelleinheit, bestehend aus einem oder mehreren aufeinanderfolgenden Paragrafen oder aus einem oder mehreren aufeinanderfolgenden Kapiteln des Prozedurteils, heißt Prozedur. Ein Prozedurname ist ein Wort, das zur Bezugnahme auf einen Paragrafen oder ein Kapitel benutzt wird; es besteht aus einem Paragrafennamen (der durch einen Kapitelnamen gekennzeichnet sein kann) oder aus einem Kapitelnamen.

Es gibt zwei Arten von Prozeduren innerhalb der PROCEDURE DIVISION:

- Prozedurvereinbarungen (DECLARATIVES), die nicht innerhalb der normalen Anweisungsfolge des Prozedurteils ausgeführt werden können und
- Prozeduren, die die normal auszuführenden Anweisungen enthalten, wenn keine besonderen Ausnahmesituationen vorhanden sind.

Die Ausführung der Quelleinheit beginnt mit der ersten Anweisung in der PROCEDURE DIVISION nach den Prozedurvereinbarungen. Anweisungen werden in der Reihenfolge ausgeführt, in der sie zur Übersetzungszeit vorliegen, außer wenn die Regeln für eine Anweisung eine andere Reihenfolge angeben.

Wird Programmsegmentierung verwendet, muss der Programmierer die gesamte PROCEDURE DIVISION in benannte Kapitel einteilen. Programmsegmentierung wird im Kapitel „Segmentierung" behandelt.

8.1.1 Struktur

Allgemeines Format

Format 1 (mit Sections)

```
PROCEDURE DIVISION [USING-Angabe] [RETURNING datenname-2].

[ DECLARATIVES.
 {kapitelname SECTION.
      USE-Anweisung.
 [paragrafenname.
      [programmsatz]...]...}...
 END DECLARATIVES.]

 {kapitelname SECTION [segmentnummer].
 [paragrafenname.
     [programmsatz]... ]... }...
```

Format 2 (ohne Sections)

```
PROCEDURE DIVISION [USING-Angabe] [RETURNING datenname -2].

[programmsatz]...
[paragrafenname.
     [programmsatz]...]...
```

wobei die **USING-ANGABE** wie folgt definiert ist:

```
USING {[BY REFERENCE] {[OPTIONAL] datenname-1}... | BY VALUE {datenname-1}... }...
```

Format 3

Gilt nur für eine Objekt-,Fabrik(Factory)- oder Schnittstellendefinition.

```
PROCEDURE DIVISION.

[{ methodendefinition}...]
```

Allgemeine Regeln

1. Der Überschrift der PROCEDURE DIVISION eines Programms oder einer Methode folgt wahlweise der Prozedurvereinbarungsteil, der Prozedurvereinbarungen (DECLARATIVES) enthält, auf die wiederum normal auszuführende Prozeduren folgen. Jede dieser Prozeduren wird aus Anweisungen, Programmsätzen, Paragrafen und/oder Kapiteln in einem syntaktisch gültigen Format gebildet.
2. Für die Beschreibung des Prozedurvereinbarungsteils siehe Abschnitt „DECLARATIVES".
3. Falls Kapitel innerhalb der PROCEDURE DIVISION verwendet werden, muss Format 1 benutzt werden, sonst kann Format 2 verwendet werden.
4. Ein Kapitel besteht aus einer Kapitelüberschrift (Kapitelname, gefolgt von dem Wort SECTION, gefolgt von einem Punkt; wird Programmsegmentierung gewünscht, kann ein Leerzeichen und eine Segmentnummer, gefolgt von einem Punkt, hinter dem Wort SECTION eingefügt werden) gefolgt von keinem, einem oder mehreren aufeinanderfolgenden Paragrafen. Ein Kapitel endet unmittelbar vor dem nächsten Kapitel oder am Ende der PROCEDURE DIVISION oder im Prozedurvereinbarungsteil der PROCEDURE DIVISION unmittelbar vor dem nächsten Kapitel bzw. bei den Schlüsselwörtern END DECLARATIVES.

Mehrfach definierte Kapitelnamen bzw. mehrfach definierte Paragrafennamen innerhalb eines Kapitels werden vom Compiler nicht als Fehler gemeldet, wenn sie nicht angesprochen werden.

5. Ein **Paragraf** besteht zumindest aus einem Paragrafennamen, gefolgt von einem Punkt sowie einem Leerzeichen. Ein oder mehrere Programmsätze können sich anschließen. Ein Paragraf endet unmittelbar vor dem nächsten Paragrafen bzw. Kapitel oder am Ende der PROCEDURE DIVISION oder im Prozedurvereinbarungsteil der PROCEDURE DIVISION unmittelbar vor dem nächsten Paragrafen bzw. Kapitel bzw. bei den Schlüsselwörtern END DECLARATIVES.
6. Wenn ein Paragraf in einem Kapitel liegt, müssen alle Paragrafen in Kapiteln liegen.
7. Ein **Programmsatz** wird aus einer oder mehreren Anweisungen gebildet, die wahlweise durch Semikolon, Leerzeichen oder Komma getrennt werden können, und wird durch einen Punkt, gefolgt von Leerzeichen, beendet.
8. Eine **Anweisung** besteht aus einer syntaktisch gültigen Kombination von Wörtern und Symbolen und muss mit einem COBOL-Verb beginnen.

8.2 PROCEDURE DIVISION-Überschrift

Funktion

Innerhalb eines aufgerufenen Programms (Unterprogramms) oder einer Methode bestimmt die Prozedurteilüberschrift die Standard-Einsprungstelle. Wahlweise können Datennamen angegeben werden, wenn vom aufrufenden Programm Daten als Parameter übergeben werden.

Format 1

```
PROCEDURE DIVISION [USING-Angabe] [RETURNING datenname-2] .
```

wobei die USING-Angabe wie folgt definiert ist:

```
USING {[BY REFERENCE] {[OPTIONAL] datenname-1}... | BY VALUE { datenname -1}...
}...
```

Format 2

Für Objekt-, Fabrik(Factory)- oder Schnittstellendefinition.

```
PROCEDURE DIVISION.
```

Syntaxregeln

1. Die USING-Angabe darf nur geschrieben werden, wenn das aufgerufene Programm von einer CALL-Anweisung oder eine Methode von einer INVOKE-Anweisung aufgerufen wird und die CALL-/INVOKE-Anweisung in der aufrufenden Quelleinheit eine
   USING-Angabe enthält. Für Aufrufe als Unterprogramm von „fremdsprachigen Programmen“ aus siehe [2] CRTE-Handbuch.
2. Die RETURNING-Angabe darf nur geschrieben werden, wenn das aufgerufene Programm von einer CALL-Anweisung oder eine Methode von einer INVOKE-Anweisung aufgerufen wird und die CALL- /INVOKE-Anweisung in der aufrufenden Quelleinheit eine RETURNING-Angabe enthält.
3. Jeder in der USING- bzw. -RETURNING-Angabe der PROCEDURE DIVISION-Überschrift angegebene Datenname muss in der LINKAGE SECTION der Quelleinheit, die diese Überschrift enthält, definiert sein und die Stufennummer 01 oder 77 haben.
   Die Datenbeschreibung von datenname-1 bzw. datenname-2 darf keine REDEFINES-oder BASED-Klausel enthalten.
4. Jeder datenname-1, der mit der BY VALUE-Angabe definiert wird, darf nur ein Datenelement von der Klasse numerisch, objekt oder zeiger bezeichnen.
5. Die RETURNING-Angabe kann in einer Methoden-Definition, einer Programm- oder Programmprototyp-Definition spezifiziert werden.
6. datenname-2 darf nicht identisch sein mit datenname-1.

Allgemeine Regeln

1. Die Standard-Einsprungstelle innerhalb eines aufgerufenen Programms oder einer aufgerufenen Methode wird durch die PROCEDURE DIVISION-Überschrift bestimmt. Um die Verknüpfung von einer aufrufenden Quelleinheit zu dieser Einsprungstelle herzustellen, muss die aufrufende Quelleinheit eine CALL- bzw. INVOKE-Anweisung enthalten. Der Name in dieser CALL- bzw. INVOKE-Anweisung muss mit dem Namen im PROGRAM-ID bzw. METHOD-ID Paragrafen der IDENTIFICATION DIVISION des aufgerufenen Programms bzw. der aufgerufenen Methode übereinstimmen.
2. Die USING-Angabe bewirkt, dass sich während des Ablaufs datenname-1 der PROCEDURE DIVISION-Überschrift im aufgerufenen Programm/Methode und bezeichner-2 bzw. bezeichner-5 in der USING-

Angabe der CALL- bzw. INVOKE-Anweisung in der aufrufenden Quelleinheit auf dieselben Daten beziehen, die in gleicher Weise für die aufgerufene und für die aufrufende Quelleinheit verfügbar sind. Die Gleichheit der Namen ist nicht erforderlich.

In der PROCEDURE DIVISION-Überschrift des aufgerufenen Programms bzw. der aufgerufenen Methode darf ein Datenname nur ein einziges Mal vorkommen; in der USING-Angabe der CALL- bzw. INVOKE-Anweisung darf derselbe Bezeichner hingegen öfter angegeben werden.

3. Im aufgerufenen Programm /Methode werden die Operanden der USING-Angabe entsprechend ihren in der LINKAGE SECTION angegebenen Datenbeschreibungen behandelt.
4. Eine Quelleinheit kann zur Ausführungszeit sowohl als aufgerufene als auch als aufrufende Quelleinheit ablaufen. Einen Sonderfall bildet die erste Quelleinheit im Ablauf (die auf Systemebene gestartet wird); sie darf in der PROCEDURE DIVISION-Überschrift **keine** USING-Angabe enthalten.

Beispiel 8-1

```
Aufrufendes Programm:

IDENTIFICATION DIVISION
PROGRAM-ID. A-PROG.
...
WORKING-STORAGE SECTION.
01  ALPHA ...
01  BETA ...
77  GAMMA ...
...
PROCEDURE DIVISION.
...
    CALL "B-PROG" USING ALPHA BETA GAMMA.        (1)
...

Aufgerufenes Programm:

IDENTIFICATION DIVISION.
PROGRAM-ID. B-PROG.
...
LINKAGE SECTION.
01  DELTA ...
01  EPSILON ...
77  THETA ...
...
PROCEDURE DIVISION USING DELTA EPSILON THETA.    (1)
...
```

(1) Die Parameter der USING-Angaben sind paarweise aufeinander bezogen, d.h. ALPHA und DELTA, BETA und EPSILON, GAMMA und THETA beziehen sich jeweils auf dasselbe Datenfeld.

ℹ Details zur Parameterübergabe zwischen ILCS-Programmen siehe Handbuch „CRTE“ [2].

8.3 DECLARATIVES

Funktion

Die DECLARATIVES-Unterabteilung ist ein wahlweise anzugebender Teil der PROCEDURE DIVISION. Er enthält Gruppen von Prozeduren, die nicht innerhalb der normalen Folge von Anweisungen der PROCEDURE DIVISION ausgeführt werden, sondern nur dann, wenn eine besondere Bedingung auftritt.

Prozedurvereinbarungen werden zur Ausführung folgender Funktionen verwendet:

- Ein-/Ausgabe-Kennsatz-Behandlung
- Behandlung von Ein-/Ausgabe-Fehlern
- Spezielle Listenprogramm-Funktionen

Format (Allgemeines Format in der PROCEDURE DIVISION)

```
PROCEDURE DIVISION [USING {datenname-1}... ].

[ DECLARATIVES.
 {kapitelname SECTION.
      USE-Anweisung.
 [paragrafenname.
      [programmsatz]...]...}...
 END DECLARATIVES.]

  {kapitelname SECTION [segmentnummer].
  [paragrafenname.
      [programmsatz]...]...}...
```

Syntaxregeln

1. Prozedurvereinbarungen müssen an den Anfang der PROCEDURE DIVISION gestellt werden, im Anschluss an das Schlüsselwort DECLARATIVES und gefolgt von einem Punkt und einem Leerzeichen. Prozedurvereinbarungen werden durch das Schlüsselwort END DECLARATIVES abgeschlossen, gefolgt von einem Leerzeichen.
2. Wie im allgemeinen Format der PROCEDURE DIVISION aufgeführt, muss die DECLARATIVES-Unterabteilung in Kapitel eingeteilt werden. Diese Kapitel heißen Prozedurvereinbarungskapitel. Jedes Prozedurvereinbarungskapitel enthält eine Gruppe von zugehörigen Prozeduren, denen eine Kapitelüberschrift vorangeht, auf welche unmittelbar eine USE-Anweisung mit anschließendem Punkt und Leerzeichen folgt.
3. Die USE-Anweisung bezeichnet die Art der Prozedurvereinbarungsprozeduren entsprechend den oben erwähnten drei Funktionen. Die Formate der USE-Anweisung sind ausführlich beschrieben ab "USE-Anweisung " und "USE BEFORE REPORTING-Anweisung " („Listenprogramm“: USE BEFORE REPORTING-Anweisung).
4. Die USE-Anweisung selbst wird niemals ausgeführt, sondern sie definiert die Bedingungen, unter denen die Prozedurvereinbarungen im zugehörigen Kapitel ausgeführt werden.

8.4 Arithmetische Ausdrücke

Funktion

Arithmethische Ausdrücke erlauben es dem Anwender, arithmetische Operationen miteinander zu kombinieren.

Format

Ein **arithmetischer Ausdruck** kann einer der folgenden sein:

- ein Bezeichner eines numerischen Datenelements
- ein numerisches Literal
- zwei arithmetische Ausdrücke, getrennt durch einen arithmetischen Operator sowie ein von Klammern eingeschlossener arithmetischer Ausdruck

Jedem arithmetischen Ausdruck kann ein unäres Plus (+) oder ein unäres Minus (-) vorangehen.

Arithmetische Operatoren

Die folgenden Operatoren können in arithmetischen Ausdrücken verwendet werden:

Binärer Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
* *	Potenzierung

Unärer Operator	Bedeutung
+	gleiche Wirkung wie Multiplikation mit dem numerischen Literal +1
-	gleiche Wirkung wie Multiplikation mit dem numerischen Literal -1

Einem binären arithmetischen Operator muss laut Standard immer ein Leerzeichen vorausgehen und nachfolgen.
Der Compiler erlaubt jedoch, dass alle diese Operatoren, außer den Additions- und Subtraktionsoperatoren, ohne die sie einschließenden Leerzeichen verwendet werden können. Dem Subtraktionsoperator (-) muss immer ein Leerzeichen vorausgehen und folgen.

Dem Additionsoperator (+) muss ein Leerzeichen folgen, wenn er vor einem vorzeichenlosen numerischen Literal steht. Beide Operatoren dürfen unmittelbar einer runden Klammer vorausgehen oder nachfolgen.
Einem unären + muss ein Leerzeichen folgen, wenn es vor einem vorzeichenlosen Literal steht. Einem unären - muss immer ein Leerzeichen folgen.

Regeln für die Bildung und Auflösung von Ausdrücken

1. Ein arithmetischer Ausdruck kann nur mit einer linken Klammer, einem unären +, einem unären -, einem Bezeichner oder einem Literal beginnen und kann nur mit einer rechten Klammer oder einer Variablen (bezeichner oder literal) enden.
2. Linke und rechte Klammern eines arithmetischen Ausdrucks müssen paarweise auftreten.

3. In Tabelle 18 ist die zulässige Kombination von Operatoren, Variablen und Klammern in arithmetischen Ausdrücken zusammengestellt.

Erstes Zeichen	**Zweites Zeichen**				
	bezeichner, literal	arithmetischer operator	unärer operator	(	)
bezeichner,literal	-	P	-	-	P
arithmetischer operator	P	-	P	P	-
unärer operator	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

Tabelle 18: Zulässige Symbol-Paare in arithmetischen Ausdrücken

P bedeutet, dass die zwei Zeichen nacheinander auftreten können (erlaubtes Paar),

- bedeutet, dass die zwei Zeichen nicht nacheinander auftreten dürfen (ungültiges Paar).

4. Klammern können in arithmetischen Ausdrücken verwendet werden, um die Reihenfolge anzugeben, in der die Elemente berechnet werden sollen.
5. Ausdrücke innerhalb von Klammern werden zuerst berechnet. Wenn geschachtelte Klammern benutzt werden, so erfolgt die Auflösung von der innersten zur äußersten Klammer.
6. Wenn keine Klammern benutzt werden oder geklammerte Ausdrücke gleichwertig sind, werden folgende Präzedenzregeln (= Stufen der Rangordnung) bei der Auflösung angewendet:
   a. Unäres Plus und Minus (zuerst aufgelöst)
   b. Potenzierung
   c. Multiplikation und Division
   d. Addition und Subtraktion (zuletzt aufgelöst)
7. Wenn aufeinanderfolgende Operationen die gleiche Stufe der Rangordnung haben, werden sie von links nach rechts aufgelöst.

Allgemeine Regeln

1. Klammern werden entweder benutzt, um logische Mehrdeutigkeit da zu vermeiden, wo aufeinanderfolgende Operationen auf der gleichen Stufe der Rangordnung erscheinen, oder um die normale Reihenfolge der Ausführung zu verändern.
2. Arithmetische Ausdrücke werden in arithmetischen und in bedingten Anweisungen verwendet.

Beispiel 8-2

Ausdruck:	A + (B - C) * D	
Auflösung des Ausdrucks:	1. B - C	(Ergebnis wird x genannt)
	2. x * D	(Ergebnis wird y genannt)
	3. A + y	(Endergebnis)

Beispiel 8-3

Ausdruck:	A + ((B / C) + (D ** E) * F) - G	
Auflösung des Ausdrucks:	1. B / C	(Ergebnis wird z genannt)

2. D ** E (Ergebnis wird x genannt)

3. x * F (Ergebnis wird y genannt)

4. z + y (Ergebnis wird a genannt)

5. A + a (Ergebnis wird b genannt)

6. b - G (Endergebnis)

8.5 Bedingungen

Allgemeine Beschreibung

Eine Bedingung ermöglicht dem Programm, zwischen zwei verschiedenen Ablaufzweigen in Abhängigkeit eines Tests zu wählen. Das Ergebnis des Tests ist einer der Werte „wahr“ oder „falsch“. Es wird zwischen einfachen und zusammengesetzten Bedingungen unterschieden.

Einfache Bedingungen

1. Bedingungsnamen-Bedingung
2. Klassenbedingung
3. Schalterzustandsbedingung
4. Vergleichsbedingung
5. Vorzeichenbedingung
6. omitted-argument-Bedingung

Jede dieser Bedingungen kann durch Klammerpaare eingeschlossen werden.

Zusammengesetzte Bedingungen

Zusammengesetzte Bedingungen werden gebildet, indem einfache und/oder zusammengesetzte Bedingungen durch die logischen Operatoren AND und OR miteinander verknüpft werden oder durch den Operator NOT negiert werden.

8.5.1 Bedingungsnamen-Bedingung

Funktion

Die Bedingungsnamen-Bedingung bewirkt, dass eine Bedingungsvariable geprüft wird, um zu entscheiden, ob ihr Wert gleich einem der Werte ist, die zu einem bestimmten Bedingungsnamengehören (für zusätzliche Information siehe „VALUE-Klausel").

Format

```
bedingungsname
```

Syntaxregeln

1. bedingungsname gibt den Bedingungsnamen an, der bei der Prüfung benutzt werden soll.
2. Ist bedingungsname nur ein einziger Wert zugeordnet, so ist die zugehörige Prüfung nur dann wahr, wenn der dem Bedingungsnamen entsprechende Wert gleich dem der zugehörigen Bedingungsvariablen ist.
3. Falls bedingungsname Wertbereiche zugeordnet sind, wird die Bedingungsvariable geprüft, um zu entscheiden, ob ihr Wert in diesen Bereich, einschließlich der Endwerte, fällt.
4. Die Bedingungsnamen-Bedingung ist eine verkürzte Form der Vergleichsbedingung (siehe Beispiel). Siehe auch „SET-Anweisung“, Format 4 ("SET-Anweisung ").

Beispiel 8-4

```
02  ZAHLUNGS-ART PICTURE  9.
    88  STUENDLICH   VALUE 1.
    88  WOECHENTLICH VALUE 2.
    88  MONATLICH    VALUE 3.
         ...
    IF  STUENDLICH GO TO STUNDEN-PROZEDUR..
```

Hier ist ZAHLUNGS-ART eine Bedingungsvariable und STUENDLICH, WOECHENTLICH und MONATLICH sind Bedingungsnamen. Wenn der derzeitige Wert von ZAHLUNGS-ART gleich 1 ist, ist das Ergebnis der Prüfung in der IF-Anweisung wahr. Im anderen Fall ist das Ergebnis falsch.
Wie oben beschrieben, ist die Bedingungsnamen-Bedingung eine verkürzte Form der Vergleichsbedingung. Die folgende Anweisung, die eine Vergleichsbedingung enthält, ist mit der obigen IF-Anweisung gleichbedeutend:

```
IF ZAHLUNGS-ART = 1 GO TO STUNDEN-PROZEDUR.
```

8.5.2 Klassenbedingung

Funktion

Die Klassenbedingung bestimmt, ob ein Operand numerisch, alphabetisch, alphabetisch in Klein- oder in Großbuchstaben ist oder ob er nur Zeichen eines Zeichenvorrats enthält, der mit klassenname in der CLASS-Klausel des SPECIAL-NAMES-Paragrafen vereinbart wurde.

Format

```
bezeichner IS [NOT] {NUMERIC | ALPHABETIC | ALPHABETIC-LOWER | ALPHABETIC-UPPER |
klassenname}
```

Syntaxregeln

1. bezeichner muss ein Datenfeld sein, das implizit oder explizit mit USAGE DISPLAY oder USAGE NATIONAL oder COMPUTATIONAL-3 bzw. PACKED-DECIMAL beschrieben ist.
2. Alle in den Prüfungen zulässigen Kategorien von Bezeichnern sind in Tabelle 19 aufgeführt:

Prüfung	zulässig für Bezeichner der Kategorien
[NOT] ALPHABETIC [NOT] ALPHABETIC-LOWER [NOT] ALPHABETIC-UPPER	alphabetisch alphanumerisch alphanumerisch druckaufbereitet numerisch druckaufbereitet national
[NOT] NUMERIC	alphanumerisch alphanumerisch druckaufbereitet numerisch druckaufbereitet numerisch national
klassenname	alphabetisch alphanumerisch alphanumerisch druckaufbereitet numerisch druckaufbereitet numerisch (nur USAGE DISPLAY)

Tabelle 19: Zulässige Formate der Klassenbedingung

Allgemeine Regeln

1. bezeichner gibt das zu prüfende Datenfeld an.
2. NUMERIC, ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER und klassenname (gegebenenfalls durch NOT verneint) geben an, welche Eigenschaft überprüft werden soll.
3. Ist bezeichner ein null-längiges Datenfeld, so ist der Wert der Klassenbedingung ohne NOT-Angabe „falsch “.
4. bezeichner wird als numerisch erkannt, wenn
   a. seine Kategorie numerisch ist,
      - bezeichner mit USAGE DISPLAY beschrieben ist und er nur das seiner Beschreibung entsprechende Vorzeichen und die Ziffern 0 bis 9 enthält.
      - bezeichner mit USAGE COMP-3 oder USAGE PACKED-DECIMAL beschrieben ist und sein Inhalt numerisch ist.

Ist in der Maskenzeichenfolge kein Vorzeichen beschrieben, dann ist in der internen Darstellung nur F als Vorzeichen erlaubt.
Ist ein Vorzeichen beschrieben, so finden Sie im Abschnitt „SIGN-Klausel" nähere Informationen.

b. seine Kategorie nicht numerisch ist und sein Inhalt nur aus den Ziffern 0 bis 9 besteht.

5. bezeichner wird als alphabetisch erkannt, wenn sein Inhalt aus einer beliebigen Kombination der Zeichen A-Z und/oder a-z und Leerzeichen besteht.
6. bezeichner wird als alphabetisch in Kleinbuchstaben (ALPHABETIC-LOWER) erkannt, wenn sein Inhalt aus einer Kombination der Kleinbuchstaben a-z und Leerzeichen besteht.
7. bezeichner wird als alphabetisch in Großbuchstaben (ALPHABETIC-UPPER) erkannt, wenn sein Inhalt aus einer Kombination der Großbuchstaben A-Z und Leerzeichen besteht.
8. bezeichner entspricht dem klassennamen, wenn sein Inhalt nur aus einer Kombination derjenigen Zeichen besteht, die durch die Definition von klassenname im SPECIAL-NAMES-Paragrafen festgelegt wurden.

8.5.3 Schalterzustandsbedingung

Funktion

Die Schalterzustandsbedingung untersucht die Stellung eines Benutzer- und Prozessschalters. Der angegebene Herstellername und sein zugehöriger ON- oder OFF-Wert muss im SPECIAL-NAMES-Paragrafen der ENVIRONMENT DIVISION aufgeführt sein.

Format

```
bedingungsname
```

Syntaxregeln

1. Das Ergebnis eines Tests ist wahr, wenn der Schalter auf der dem Bedingungsnamen entsprechenden Stellung steht.
2. Der Zustand eines Schalters kann mit einer SET-Anweisung, Format 3, verändert werden (siehe „SET-Anweisung“, Format 3, Seite "SET-Anweisung").

8.5.4 Vergleichsbedingung

Funktion

Eine Vergleichsbedingung bewirkt einen Vergleich zwischen zwei Operanden, die auch ein Literal, ein Index oder ein arithmetischer Ausdruck sein können.

Format

```
{  bezeichner-1                vergleichsoperator  {  bezeichner-2
 | literal-1                                         | literal-2
 | arithmetischer-ausdruck-1                        | arithmetischer-ausdruck-2
 | index-1                                          | index-2
}                                                  }
```

Syntaxregeln

1. Der erste und der zweite Operand einer Vergleichsbedingung dürfen nicht beide Literale sein. Es dürfen auch nicht beide Operanden gleichzeitig NULL sein.
2. Der Vergleichsoperator muss einer der in Tabelle 20 aufgeführten Operatoren sein. Es muss ihm je ein Leerzeichen vorausgehen und folgen.

Operator	Bedeutung
IS [NOT] GREATER THAN IS [NOT] >	[Nicht] größer als
IS [NOT] LESS THAN IS [NOT] <	[Nicht] kleiner als
IS [NOT] EQUAL TO IS [NOT] =	[Nicht] gleich
IS GREATER THAN OR EQUAL TO IS > =	Größer oder gleich
IS LESS THAN OR EQUAL TO IS <=	Kleiner oder gleich

Tabelle 20: Vergleichsoperatoren

   Die Sonderzeichen >, <, = sind hier nicht unterstrichen, damit sie nicht mit anderen Sonderzeichen verwechselt werden.
3. Der Vergleichsoperator bezeichnet die Art des Vergleichs, der in einer Vergleichsbedingung ausgeführt werden soll.
4. Sind in einer Vergleichsbedingung Daten der Klassen objekt oder zeiger beteiligt, dürfen nur Vergleichsoperatoren mit der Bedeutung „gleich“ oder „nicht gleich“ verwendet werden.
5. Sind in einer Vergleichsbedingung Daten der Klassen objekt oder zeiger beteiligt, müssen der linke und rechte Operand von der gleichen Kategorie sein.
6. Der Vergleich von stark typisierten Datengruppen ist nicht erlaubt. Nur die Datenelemente aus stark typisierten Datengruppen können miteinander verglichen werden.
7. Die zulässigen Vergleiche und die Art des Vergleichs sind in Tabelle 21 angegeben:.

Rechter Operand / **Linker Operand**	Gruppe	Alphabetisch	Alphanumerisch	National	Numerisch	Expression	Indexname	Indexdatenfeld	Objekt	Zeiger (Pointer)
Gruppe	an	an	an	-	an	-	-	-	-	-
Alphabetisch	an	an	an	ant	-	-	-	-	-	-
Alphanumerisch	an	an	an	ant	anu,i,d	-	-	-	-	-
National	-	ant	ant	nt	nnu,i	-	-	-	-	-
Numerisch	an	-	anu,i,d	nnu,i	nu	nu	ix2,i	-	-	-
Expression	-	-	-	-	nu	nu	ix2,i	-	-	-
Indexname	-	-	-	-	ix2,i	ix2,i	ix1	ix3	-	-
Indexdatenfeld	-	-	-	-	-	-	ix3	ix3	-	-
Objekt	-	-	-	-	-	-	-	-	ob	-
Zeiger (Pointer)	-	-	-	-	-	-	-	-	-	pt

Tabelle 21: Zulässige Vergleiche verschiedener Operanden[1)]

1) Tabelleneinträge:

i = nur für ganze Zahlen (integer)

d = bei alphanumerischen Literalen nur für numerische Datenfelder mit USAGE DISPLAY erlaubt

- = Vergleich ist nicht erlaubt

nu = Vergleich von numerischen Operanden

an = Vergleich von alphanumerischen Operanden

nt = Vergleich von nationalen Operanden

ix = Vergleich von Indexnamen oder Indexdatenfeldern

ob = Vergleich von Objektreferenzen

pt = Vergleich von Zeigern (Pointer)

ant = Vergleich von alphanumerischen und nationalen Operanden

anu = Vergleich von alphanumerischen und ganzzahligen numerischen Operanden

nnu = Vergleich von nationalen und ganzzahligen numerischen Operanden

Vergleich von numerischen Operanden (nu)

Wenn zwei numerische Operanden verglichen werden, wird dazu ihr algebraischer Wert unabhängig von dem verwendeten Datenformat (USAGE-Klausel) verwendet. Die Länge, d.h. die Anzahl der Ziffern, die sie enthalten, ist nicht entscheidend.
Numerische Operanden ohne Vorzeichen werden bei Vergleichen als positiv betrachtet.Null wird als eindeutiger Wert betrachtet, unabhängig vom Vorzeichen.

Beispiel 8-5

-50 ist kleiner als +5
+75 ist größer als +5
-100 ist kleiner als -10
-0 ist gleich +0

Vergleich von alphanumerischen mit ganzzahligen numerischen Operanden (anu)

Der numerische Operand muss ein ganzzahliges Literal oder ein ganzzahliges Datenfeld mit USAGE DISPLAY sein. Der hier beschriebene Compiler lässt jedoch auch andere Datenformate zu.

a. Ist der alphanumerische Operand ein **Datenelement** oder ein alphanumerisches Literal, wird der numerische Operand so behandelt, als ob er entsprechend den Regeln der MOVE-Anweisung zu einem alphanumerischen Datenelement übertragen wurde. Die Länge des alphanumerischen Datenfeldes entspricht der Ziffernanzahl in der Beschreibung des numerischen Datenfeldes, wobei Skalenstellenzeichen P nicht mitgezählt werden.
b. Ist der alphanumerische Operand eine **Datengruppe**, wird der numerische Operand so behandelt, als ob er entsprechend den Regeln der MOVE-Anweisung zu einer Datengruppe mit derselben Größe wie das numerische Datenfeld übertragen wurde.

In beiden Fällen wird anschließend das gedachte Datenfeld mit dem alphanumerischen Operanden verglichen. Dies geschieht entsprechend den Regeln für den Vergleich von alphanumerischer Operanden (an).

Vergleich von nationalen mit ganzzahligen numerischen Operanden (nnu)

Der numerische Operand muss ein ganzzahliges Literal oder ein ganzzahliges Datenfeld sein. Er wird so behandelt, als ob er entsprechend den Regeln der MOVE-Anweisung zu einem nationalen Datenelement übertragen wurde. Die Länge des nationalen Datenfeldes entspricht der Ziffernanzahl in der Beschreibung des numerischen Datenfeldes, wobei Ska lenstellenzeichen P nicht mitgezählt werden.

Anschließend wird das gedachte nationalen Datenfeld mit dem nationalen Operanden verglichen. Dies geschieht entsprechend den Regeln für den Vergleich von nationalen Operanden (nt).

Vergleich von alphanumerischen Operanden (an)

Wenn zwei alphanumerische Operanden verglichen werden, wird der Vergleich entsprechend der Sortierfolge der PROGRAM COLLATING SEQUENCE durchgeführt (siehe „OBJECT-COMPUTER-Paragraf").

Es gilt zwei Fälle zu beachten:

a. Vergleich von Operanden gleicher Länge

   Das Programm vergleicht die Zeichen entsprechender Zeichenstellen miteinander. Es wird an der höchstwertigen Stelle begonnen (das heißt linksbündig) und der Vergleich so lange fortgesetzt, bis entweder zwei ungleiche Zeichen auftreten oder bis das Ende der Operanden erreicht ist.
   Sind alle Zeichen paarweise gleich, werden die Operanden als gleich betrachtet. Zwei Operanden mit der Länge Null sind ebenfalls gleich.
   Tritt ein Paar von ungleichen Zeichen auf, so wird festgestellt, welches Zeichen eine höhere Ordnungszahl innerhalb der Sortierfolge hat. Der Operand, der das höhere Zeichen enthält, wird als der größere Operand betrachtet.

   Beispiel 8-6

linker Operand	rechter Operand	keine COLLATING SEQUENCE oderCOLLATING SEQUENCE NATIVE	COLLATING SEQUENCE ALPHATAB (siehe Beispiel 6-5 im Abschnitt "ALPHABET-Klausel")
+-*	ABC	'kleiner' Das 1. Zeichen linker Operand „+“ ist kleiner als das 1. Zeichen rechter Operand „A“.	'größer' Das 1.Zeichen linker Operand „+“ ist größer als das 1. Zeichen rechter Operand „A“, weil „A“ in der Sortierfolge ALPHATAB die Ordnungszahl 1 hat.
RADE	RABE		

		’größer’ Das 3. Zeichen linker Operand „D“ ist größer als das 3. Zeichen rechter Operand „B“.	’gleich’ Das 3.Zeichen linker Operand „D“ ist gleich dem 3. Zeichen rechter Operand „B“, weil „D“ und „B“ in der Sortierfolge ALPHATAB beide die Ordnungszahl 1 haben.
XYZ	XYZ	’gleich’ Alle Zeichen sind paarweise gleich.	’gleich’ Alle Zeichen sind paarweise gleich.

b. Vergleich von Operanden ungleicher Länge

Der Vergleich verläuft so, als ob der kürzere Operand rechts mit so vielen alphanumerischen Leerzeichen aufgefüllt wäre, so dass die Operanden die gleiche Länge haben.

Es sind dann die beschriebenen Regeln aus a) „Vergleich von Operanden gleicher Länge" anzuwenden.

Beispiel 8-7

linker Operand	**rechter Operand**	**keine COLLATING SEQUENCE oderCOLLATING SEQUENCE NATIVE**	**COLLATING SEQUENCE ALPHATAB (siehe Beispiel 6-5 im** Abschnitt "ALPHABET-Klausel" **)**
SMITH	SMITHY	’kleiner’ Der linke Operand wird mit Leerzeichen zu „SMITH'BLANK'“ aufgefüllt. Das 6. Zeichen linker Operand „'BLANK'“ ist kleiner als das 6. Zeichen rechter Operand „Y“.	’kleiner’ Der linke Operand wird mit Leerzeichen zu „SMITH'BLANK'“ aufgefüllt. Das 6. Zeichen linker Operand „'BLANK'“ ist kleiner als das 6. Zeichen rechter Operand „Y“.
AB	ABBA	’kleiner’ Der linke Operand wird mit Leerzeichen zu „AB'BLANK''BLANK'“ aufgefüllt. Das 3. Zeichen linker Operand „'BLANK'“ ist kleiner als das 3. Zeichen rechter Operand „B“.	’größer’ Der linke Operand wird mit Leerzeichen zu „AB'BLANK''BLANK'“ aufgefüllt. Das 3. Zeichen linker Operand „'BLANK'“ ist größer als das 3. Zeichen rechter Operand „B“, weil „B“ in der Sortierfolge ALPHATAB die Ordnungszahl 1 hat.

Vergleich von nationalen Operanden (nt)

Es gilt zwei Fälle zu beachten:

a. Vergleich von Operanden gleicher Länge

Das Programm vergleicht die Zeichen entsprechender Zeichenstellen miteinander. Es wird an der höchstwertigen Stelle begonnen (das heißt linksbündig) und der Vergleich so lange fortgesetzt, bis entweder zwei ungleiche Zeichen auftreten oder bis das Ende der Operanden erreicht ist.
Sind alle Zeichen paarweise gleich, werden die Operanden als gleich betrachtet. Zwei Operanden mit der Länge Null sind ebenfalls gleich.
Tritt ein Paar von ungleichen Zeichen auf, so wird festgestellt, welches Zeichen eine höhere Ordnungszahl innerhalb der nationalen Sortierfolge hat. Der Operand, der das höhere Zeichen enthält, wird als der größere Operand betrachtet.

b. Vergleich von Operanden ungleicher Länge

Der Vergleich verläuft so, als ob der kürzere Operand rechts mit so vielen nationalen Leerzeichen aufgefüllt wäre, so dass die Operanden die gleiche Länge haben.

Es sind dann die beschriebenen Regeln aus a) „Vergleich von Operanden gleicher Länge" anzuwenden.

Vergleich von alphanumerischen mit nationalen Operanden (ant)

Der alphanumerische Operand wird so behandelt, als ob er entsprechend den Regeln der MOVE-Anweisung zu einem nationalen Datenelement übertragen und konvertiert wurde. Das nationale Datenfeld hat genau so viele Zeichenstellen wie das alphanumerischen Datenfeld.

Anschließend wird das gedachte Datenfeld mit dem nationalen Operanden verglichen. Dies geschieht entsprechend den Regeln für den Vergleich von nationalen Operanden (nt).

Beispiel 8-8

linker, alphanumerischer Operand	rechter, nationalerOperand	Ergebnis des Vergleichs
123	XYZ	'kleiner' Der linke Operand wird in die nationale Darstellung konvertiert. Das 1. Zeichen linker Operand „1“ ist in der nationalen Sortierfolge kleiner als das 1. Zeichen rechter Operand „X“.
AB	ABBA	'kleiner' Der linke Operand wird in die nationale Darstellung konvertiert und mit Leerzeichen zu „AB'BLANK''BLANK'“ aufgefüllt. Das 3. Zeichen linker Operand „'BLANK'“ ist in der nationalen Sortierfolge kleiner als das 3. Zeichen rechter Operand „B“.

Vergleich von Indexnamen oder Indexdatenfelder (ix)

1. Sind beide Operanden Indexnamen, dann werden die den beiden Indizes entsprechenden Tabellenelementnummern miteinander verglichen (ix1).
2. Ist ein Operand numerisch (nur ganzzahlig), so wird diese Zahl als Tabellenelementnummer angesehen und mit der Tabellenelementnummer verglichen, die dem Indexnamen entspricht (ix2).
3. Ist ein Operand ein Indexdatenfeld, so werden die aktuellen Werte der Felder (ohne Umrechnung zu Tabellenelementnummern!) verglichen (ix3).

Vergleich von Objektreferenzen (ob)

Die Bedingung „gleich“ ist nur dann wahr, wenn bezeichner-1 und bezeichner-2 das gleiche Objekt referenzieren.

Vergleich von Zeigern (pt)

Die Bedingung „gleich“ ist nur dann wahr, wenn bezeichner-1 und bezeichner-2 die gleiche Speicherzelle referenzieren.

8.5.5 Vorzeichen-Bedingung

Funktion

Die Vorzeichen-Bedingung entscheidet, ob der algebraische Wert eines numerischen Operanden (das heißt ein als numerisch beschriebenes Datenfeld) kleiner, größer oder gleich Null ist.

Format

```
{bezeichner | arithmetischer-ausdruck} IS [NOT] {POSITIVE | NEGATIVE | ZERO}
```

Syntaxregel

1. bezeichner muss sich auf ein numerisches Datenfeld beziehen.

Allgemeine Regeln:

1. bezeichner oder arithmetischer-ausdruck bezeichnet den zu prüfenden Operanden.
2. POSITIVE, NEGATIVE oder ZERO bezeichnen die durchzuführende Prüfung.
3. Ein Operand ist dann POSITIVE, wenn sein Wert größer Null ist, NEGATIVE, wenn sein Wert kleiner Null ist, und ZERO, wenn sein Wert gleich Null ist.

8.5.6 OMITTED-ARGUMENT-Bedingung

Die OMITTED-ARGUMENT-Bedingung entscheidet, ob ein Argument für ein Programm oder eine Methode bereitgestellt wurde.

Format

```
datenname-1 IS [NOT] OMITTED
```

Syntaxregeln

1. datenname-1 muss in der LINKAGE SECTION mit Stufennummer 01 oder 77 definiert sein. Die Datenerklärung für datenname-1 darf keine REDEFINES-Klausel enthalten.
2. datenname-1 muss in der USING-Angabe der PROCEDURE DIVISION-Überschrift des Programmelementes angegeben sein, das diese Bedingung enthält.

Allgemeine Regeln

1. Das Ergebnis der Überprüfung von IS OMITTED ist dann wahr , wenn
   a. für den formalen Parameter datenname-1 des aufgerufenen Programms bzw. der aufgerufenen Methode als aktueller Wert OMITTED angegeben worden ist,
   b. für den formalen Parameter datenname-1 kein aktueller Wert angegeben wurde; das ist nur für Parameter am Ende der USING-Liste möglich, wenn auch für alle nachfolgenden Parameter kein Wert angegeben wurde.
2. Ist NOT angegeben, dann kehrt sich der gemäß Punkt 1.) ermittelte Wahrheitswert um.

8.5.7 Zusammengesetzte Bedingungen

Funktion

Eine zusammengesetzte Bedingung besteht aus einer Kombination von zwei oder mehr einfachen Bedingungen.

Format

```
bedingung {AND | OR} [NOT] bedingung [{AND | OR} [NOT] bedingung] ...
```

Syntaxregeln

1. bedingung bezeichnet eine einfache Bedingung.
2. Klammern können innerhalb einer zusammengesetzten Bedingung benutzt werden, um die Lesbarkeit zu verbessern oder um die normale Ablauffolge zu verändern.
3. Die einfachen Bedingungen innerhalb einer zusammengesetzten Bedingung sind durch logische Operatoren voneinander getrennt, entsprechend den angegebenen Regeln. Den logischen Operatoren muss ein Leerzeichen vorausgehen und eines folgen.
4. In einer zusammengesetzten Bedingung dürfen max. 60 einfache Bedingungen stehen.
5. Die logischen Operatoren und ihre Bedeutung sind in Tabelle 22 aufgeführt.

Operator	Bedeutung	Erläuterung
OR	Logisch inklusives Oder (einer oder beide)	Der Ausdruck A OR B ist wahr, wenn A wahr ist oder wenn B wahr ist oder beide, A und B, wahr sind.
AND	Logische Verknüpfung (beide)	Der Ausdruck A AND B ist nur dann wahr, wenn beide, A und B, wahr sind.
NOT	Logische Verneinung	Der Ausdruck NOT A ist nur dann wahr, wenn A falsch ist.

Tabelle 22: Logische Operatoren

6. Tabelle 23 zeigt, auf welche Weise Bedingungen und logische Operatoren kombiniert werden dürfen.

Erstes Symbol	Zweites Symbol					
	einfache Bedingung	OR	AND	NOT	(	)
einfache Bedingung	-	P	P	-	-	P
OR	P	-	-	P	P	-
AND	P	-	-	P	P	-
NOT	P	-	-	-	P	-
(	P	-	-	P	P	-
)	-	P	P	-	-	P

Tabelle 23: Zulässige Symbol-Paare von Bedingungen und logischen Operatoren [1)]

[1)] P bedeutet, dass die beiden Symbole aufeinander folgen dürfen.

7. **Präzedenzregeln für die Ausdrucksauflösung**
   Die Auflösung zusammengesetzter Bedingungen beginnt mit dem innersten Klammernpaar und wird bis zum äußersten Paar fortgesetzt. Wenn die Reihenfolge der Auflösung nicht durch Klammern bestimmt ist, wird der Ausdruck nach folgenden Präzedenzregeln (Rangordnungsstufen) aufgelöst:
   - Arithmetische Ausdrücke

- Vergleichsoperatoren
- NOT-Bedingungen
- AND und die dazugehörigen Bedingungen werden von links nach rechts aufgelöst.
- OR und die dazugehörigen Bedingungen werden zuletzt aufgelöst, ebenfalls von links nach rechts.
- Wenn aufeinanderfolgende Ausdrücke die gleiche Rangordnungsstufe haben, werden sie von links nach rechts aufgelöst.

Beispiel 8-9

Man betrachte den Ausdruck:

```
A IS NOT GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE
```

Er wird so aufgelöst, als wären folgende Klammern gegeben:

```
(A IS NOT GREATER THAN B) OR (((A+B) IS EQUAL TO C) AND (D IS POSITIVE)).
```

Beispiel 8-10

Tabelle 24 zeigt einige Beziehungen zwischen logischen Operatoren und einfachen Bedingungen.

Operanden	Wert von A[1)]	Wahr	Falsch	Wahr	Falsch
	Wert von B[1)]	Wahr	Wahr	Falsch	Falsch
Kombinationen	NOT A	Falsch	Wahr	Falsch	Wahr
	A AND B	Wahr	Falsch	Falsch	Falsch
	A OR B	Wahr	Wahr	Wahr	Falsch
	NOT (A AND B)	Falsch	Wahr	Wahr	Wahr
	NOT A AND B	Falsch	Wahr	Falsch	Falsch
	NOT (A OR B)	Falsch	Falsch	Falsch	Wahr
	NOT A OR B	Wahr	Wahr	Falsch	Wahr

Tabelle 24: Resultate der logischen Operatoren

[1)] A und B stehen für Bedingungen.

8.5.8 Indirekte Subjekte und Vergleichsoperatoren

Funktion

Sind innerhalb einer zusammengesetzten Bedingung keine Klammern angegeben, können alle Vergleichsbedingungen, mit Ausnahme der ersten, auf folgende Weise abgekürzt werden:

- Das Subjekt der Vergleichsbedingung kann wegfallen.
- Das Subjekt und der Vergleichsoperator der Vergleichsbedingung können wegfallen.

Der Compiler erlaubt jedoch Klammern, und zwar in Vergleichssubjekten und -objekten, die arithmetische Ausdrücke sind, und um die Reihenfolge der Auswertung der logischen Operatoren AND und OR zu beeinflussen.

Format des indirekten Subjekts

```
...subjekt vergleichsoperator objekt {AND | OR} [NOT] vergleichsoperator objekt...
```

Format des indirekten Subjekts und indirekten Vergleichsoperators

```
...subjekt vergleichsoperator objekt {AND | OR} [NOT] objekt...
```

Syntaxregeln

1. Innerhalb einer Folge von Vergleichsbedingungen können beide Abkürzungsformen benutzt werden. Die Verwendung dieser Abkürzungen hat die gleiche Auswirkung, als ob das weggelassene Subjekt durch das zuletzt aufgeführte Subjekt bzw. der weggelassene Vergleichsoperator durch den zuletzt aufgeführten Vergleichsoperator ersetzt würde.
2. „NOT“ in einer abgekürzten zusammengesetzten Vergleichsbedingung wird folgendermaßen interpretiert:
   a. Folgt dem Wort NOT einer der Vergleichsoperatoren
   GREATER, >, LESS, <, EQUAL, =,
   so gilt „NOT“ als Teil des jeweiligen Vergleichsoperators.
   b. Folgt dem Wort NOT einer der übrigen Vergleichsoperatoren, so gilt „NOT“ als logischer Operator zur Verneinung der jeweiligen Vergleichsbedingung.
3. Ein „NOT“ vor einer öffnenden Klammer wirkt bis zur entsprechenden schließenden Klammer (siehe Beispiel 8-15).

Beispiel 8-11

Indirekte Subjekte

```
IF X = Y OR > W OR < Z
```

ist gleichbedeutend mit:

```
IF X = Y OR X > W OR X < Z
```

In diesem Beispiel ist das indirekte Subjekt das zuletzt genannte Subjekt, also X.

Beispiel 8-12

Indirekte Subjekte und Vergleichsoperatoren

```
IF X = Y OR Z OR W
```

ist gleichbedeutend mit:

```
IF X = Y OR X = Z OR X = W
```

In diesem Beispiel ist das indirekte Subjekt das zuletzt genannte Subjekt, also X und der indirekte Operator ist der zuletzt genannte Operator, also =.

Beispiel 8-13

Indirektes Subjekt und indirektes Subjekt mit Vergleichsoperator

```
X = Y AND > Z OR A
```

ist gleichbedeutend mit:

```
X = Y AND X > Z OR X > A
```

Da X hier das einzige Subjekt ist, wird es in den beiden einfachen Bedingungen eingesetzt. Der zuletzt genannte Operator, nämlich >, wird in der dritten einfachen Bedingung eingesetzt.

Beispiel 8-14

```
A > B AND NOT > C OR D
```

ist gleichbedeutend mit:

```
A > B AND NOT A > C OR NOT A > D
```

```
oder ((A > B) AND (A NOT > C)) OR (A NOT > D)
```

Beispiel 8-15

```
A NOT = "A" AND NOT ("B" OR NOT "C")
```

ist gleichbedeutend mit:

```
A NOT = "A" AND NOT (A NOT = "B" OR NOT A NOT = "C")
```

oder

```
A NOT = "A" AND A = "B" AND A NOT = "C"
```

oder

```
A = "B".
```

8.6 Arithmetische Anweisungen

Syntaxregeln

1. Alle in arithmetischen Anweisungen verwendeten Bezeichner müssen als numerische Daten im Datenteil definiert sein.
2. Alle Literale, die in arithmetischen Anweisungen verwendet werden, müssen numerisch sein. Es können Gleitpunktliterale sein.
3. Die Maximalgröße eines jeden Operanden (Bezeichner oder Literal) beträgt 31 Dezimalziffern.
4. Die Maximalgröße aller Ergebnisse nach der Dezimalpunktausrichtung beträgt 31 Dezimalziffern.
5. Werden mehrere Operanden, die in einer arithmetischen Anweisung auftreten, ihrem Dezimalpunkt entsprechend in einem hypothetischen Datenfeld „überlagert“, dann darf die dafür benötigte Größe dieses Datenfeldes (gemeinsame Stellenzahl) 31 Dezimalziffern nicht überschreiten (siehe „ADD-Anweisung" und „SUBTRACT-Anweisung").
6. Maximal 100 Operanden können in einer arithmetischen Anweisungen bzw. einem arithmetischen Ausdruck angegeben werden. Die Anzahl der öffnenden und schließenden Klammern () darf 250 nicht übersteigen.
7. Im Format eines Datenfeldes, das an einer Rechnung beteiligt ist (z.B. als ein Summand, Subtrahend oder Multiplikator), dürfen keine Zeichen zur Druckaufbereitung auftreten. Vorzeichen und Rechendezimalpunkte gelten nicht als Zeichen zur Druckaufbereitung.
8. Bezeichner, die nur dazu verwendet werden, das Resultat einer arithmetischen Anweisung aufzunehmen (z. B. der in der GIVING-Angabe benutzte Bezeichner) können numerisch druckaufbereitete Felder sein (siehe „GIVING-Angabe").
9. Bedingungsnamen dürfen nicht als Operanden auftreten.

Allgemeine Regeln

1. Die Datenbeschreibung der Operanden muss nicht gleich sein; notwendige Konvertierung und Dezimalpunktausrichtung werden während der gesamten Rechnung durchgeführt (siehe „MOVE-Anweisung “, Regeln für numerische Übertragung, "MOVE-Anweisung ").
2. Wenn das Sende- und Empfangsfeld einer arithmetischen Anweisung oder in einer INSPECT-, MOVE-, SET-, STRING- oder UNSTRING-Anweisung den gleichen Internspeicherplatz belegt (d.h. wenn sich die Operanden überlagern), ergeben sich bei der Ausführung der Anweisung unvorhersagbare Ergebnisse.
3. Das Ergebnis ist ebenfalls unvorhersagbar, wenn der Bezeichner zur Programmausführungszeit andere als numerische Daten enthält.

   > ¡ Wenn die Eingabeoperanden für arithmetische Anweisungen keine gültigen numerischen Daten enthalten, kann zur Programmausführungszeit ein Datenfehler auftreten.

4. Die folgenden Regeln werden für die Berechnung von Exponentialausdrücken verwendet:
   a. Ist die Basis eines Exponentialausdrucks Null, muss der Exponent einen Wert größer Null haben. Andernfalls tritt eine Überlauf-Bedingung auf.
   b. Hat ein Exponentialausdruck eine positive und eine negative reelle Lösung, ist der Ergebniswert positiv.
   c. Ergibt die Berechnung keine reelle Zahl, tritt eine Überlauf-Bedingung auf.
5. Bei Auflösung von arithmetischen Anweisungen generiert der Compiler eine Reihe von arithmetischen Operationen. Abhängig von der Beziehung der verschiedenen Operanden zueinander generiert der Compiler ein oder mehrere Zwischenergebnisfelder. Diese Zwischenergebnisfelder werden so lange aufgehoben, bis sie zur Lösung des Endergebnisses der Anweisung benötigt werden.

   Die folgende Tabelle 25 zeigt die Anzahl der ganzzahligen und der Dezimalziffern, die je nach ausgeführter Operation im Ergebnisfeld aufgehoben werden. Aus dieser Tabelle kann für eine gegebene Anweisung die optimale Operandengröße für die gewünschte Genauigkeit ermittelt werden. Anhand der Dezimalstellen in

jedem der Operanden und durch Bezugnahme auf die in der Tabelle angegebenen Formeln kann der Programmierer die genaue Stellenanzahl bestimmen, die der Compiler dem Ergebnisfeld zur Verfügung stellen wird. Jedoch wird beim Absetzen des Ergebnisses in das Ergebnisfeld Dezimalpunktausrichtung durchgeführt, so dass dies ebenfalls für die Bestimmung der Genauigkeit des Ergebnisses wichtig ist.

Hat einer der Operanden das Datenformat COMPUTATIONAL oder COMPUTATIONAL-5, so gelten spezielle Regeln, die in der Tabelle 25 nicht dargestellt werden können.

Art derAnweisung	**Operation**	**Dezimalstellen im Zwischenergebnis (d)**	**Ganzzahl-Stellen im Zwischenergebnis (i)**
Arithmetisch	Addition oderSubtraktion (+) oder (-)	MAX (Ad, Bd)	MAX (Ai+1, Bi+1)
	Multiplikation (*)	Ad+Bd	Ai+Bi
	Division (/)	MAX (Fd+1, Ad)	Ai+Bd
IF oder PERFORM	Addition oderSubtraktion (+) oder (-)	MAX (Ad, Bd)	MAX(Ai+1, Bi+1)
	Multiplikation(*)	Ad+Bd	Ai+Bi
	Division (/)	Ad	Ai+Bd

Tabelle 25: Berechnung der ganzzahligen Stellen und der Dezimalziffernstellenin Zwischenergebnissen

i	= errechnete ganzzahlige Stellen
Ai	= ganzzahlige Stellen im ersten Operanden
Bi	= ganzzahlige Stellen im zweiten Operanden
d	= errechnete Dezimalstellen
Ad	= Dezimalstellen im ersten Operanden
Bd	= Dezimalstellen im zweiten Operanden
Fd	= Dezimalstellen im Endergebnis
MAX	= der jeweils größere Wert der angegebenen Operanden

Enthält das Ergebnisfeld (i+d) mehr als 31 Stellen, dann wird in den meisten Fällen mit Gleitpunktarithmetik gerechnet.

Ferner werden in Gleitpunktarithmetik durchgeführt:

- Potenzierung
- Divisionen, die in Argumenten von internen Standard-Funktionen angegeben sind

8.7 Angaben in Anweisungen

In diesem Kapitel werden folgende Themen behandelt:

- CORRESPONDING-Angabe
- GIVING-Angabe
- ROUNDED-Angabe
- ON SIZE ERROR-Angabe

8.7.1 CORRESPONDING-Angabe

Die CORRESPONDING-Angabe ermöglicht es dem Anwender, mit einer Anweisung Operationen auf verschiedene Datenelemente gleichen Namens in verschiedenen Datengruppen auszuführen.

1. CORR ist die Abkürzung von CORRESPONDING.
2. Alle Bezeichner müssen sich auf Datengruppen beziehen. Bezieht sich ein Bezeichner auf eine nationale Datengruppe, so wird diese nicht wie ein Datenelement verarbeitet, sondern wie eine Gruppe.
3. Paare von Datenfeldern entsprechen sich, wenn die nachstehend beschriebenen Bedingungen erfüllt sind; alle anderen Datenfelder werden bei der Operation nicht berücksichtigt.
4. Beide Datenfelder haben den gleichen Namen und gleiche Qualifizierung, bis zu, aber nicht unbedingt einschließlich, bezeichner-1 und bezeichner-2.
5. Keines der Datenfelder ist mit FILLER erklärt.
6. Mindestens eines der Datenfelder ist im Falle von MOVE CORRESPONDING ein Datenelement und die daraus resultierende MOVE-Anweisung ist gültig entsprechend den Regeln der MOVE-Anweisung; beide Datenfelder sind Datenelemente im Falle von ADD CORRESPONDING oder SUBTRACT CORRESPONDING.
7. Ein Datenfeld, das bezeichner-1 oder bezeichner-2 untergeordnet ist und das mit einer REDEFINES-Klausel, RENAMES-Klausel, OCCURS-Klausel, USAGE IS INDEX-Klausel definiert oder von der Klasse objekt oder der Klasse zeiger ist, wird ignoriert; alle Felder, die diesen Feldern untergeordnet sind, werden ebenfalls ignoriert. bezeichner1 oder bezeichner-2 können jedoch mit REDEFINES-Klauseln oder OCCURS-Klauseln definiert werden oder Datenfeldern untergeordnet sein, die mit REDEFINES-Klauseln oder OCCURS-Klauseln definiert sind.
8. Auf Bezeichner, die einer Teilfeldselektion unterzogen werden, kann die CORRESPONDING-Angabe nicht angewendet werden.

Beispiel 8-16

In diesem Beispiel werden Datenelemente in ARBEITER-SATZ von entsprechenden Feldern in LOHN-KONTROLLE abgezogen.

Anweisung in der PROCEDURE DIVISION:

```
    SUBTRACT CORRESPONDING ARBEITER-SATZ FROM LOHN-KONTROLLE.
```

Einträge in der DATA DIVISION:

```
01  ARBEITER-SATZ.
    02  ARBEITER-NR.
        03  ARBEITS-ORT...
        03  STECHUHR-NR.
            04  SCHICHT-
KENNZAHL...
            04  KONTROLL-
NR...
    02  LOHN.
        03  ARBEITSSTUNDEN...
        03  AUSZAHLUNG...
    02  STEUER-SATZ...
    02  ABZUEGE...
```

```
01  LOHN-KONTROLLE.
    02  ARBEITER-NR.
        03  STECHUHR-NR...
        03  FILLER...
    02  ABZUEGE.
        03  STEUER-SATZ...
        03  STEUER-
BETRAG...
        03  VORSCHUSS...
    02  LOHN.
        03
ARBEITSSTUNDEN...
        03  AUSZAHLUNG...
    02  NETTO-ZAHLUNG...
    02  ARBEITER-NAME.
        03  SCHICHT-
KENNZAHL...
```

Entsprechend den Regeln für die CORRESPONDING-Angabe finden folgende Subtraktionen statt:

1. Operand	2. Operand
ARBEITSSTUNDEN OF LOHN OF ARBEITER-SATZ	ARBEITSSTUNDEN OF LOHN OF LOHN-KONTROLLE
AUSZAHLUNG OF LOHN OF ARBEITER-SATZ	AUSZAHLUNG OF LOHN OF LOHN-KONTROLLE

Die folgenden Felder werden aus den aufgeführten Gründen nicht subtrahiert.

Feld	Begründung
ARBEITER-NR	Feld ist in keiner der beiden Gruppen ein Element
ARBEITS-ORT OF ARBEITER-NR OF ARBEITER-SATZ	Name kommt nicht in LOHN-KONTROLLE vor
STECHUHR-NR OF ARBEITER-NR	Feld ist in einer Gruppe kein Element
SCHICHT-KENNZAHL OF STECHUHR-NR OF ARBEITER-NR OF ARBEITER-SATZ	Qualifizierung ist nicht identisch in LOHN-KONTROLLE
KONTROLL-NR OF STECHUHR-NR OF ARBEITER-NR OF ARBEITER-SATZ	Name kommt in LOHN-KONTROLLE nicht vor
LOHN	Name ist in keiner der beiden Gruppen ein Element
STEUER-SATZ OF ARBEITER-SATZ	Qualifizierung ist nicht identisch inLOHN-KONTROLLE
ABZUEGE	Feld ist in einer Gruppe kein Element

8.7.2 GIVING-Angabe

Syntaxregeln

1. Der dem Wort GIVING folgende Bezeichner kann ein numerisch druckaufbereitetes Element sein, da er nicht selbst an der Rechnung beteiligt ist.
2. Wenn GIVING angegeben ist, wird das Ergebnis der arithmetischen Operation dem angegebenen Bezeichner zugewiesen.
3. Das in bezeichner abgespeicherte Resultat ersetzt den vorherigen Inhalt von bezeichner. Es ist daher nicht notwendig, den Bezeichner auf Null zu setzen.

Beispiel 8-17

```
ADD A B GIVING C.
```

Die Summe A + B wird C zugewiesen, A und B werden nicht verändert.

8.7.3 ROUNDED-Angabe

Syntaxregeln

1. Falls nach der Dezimalpunktausrichtung die Anzahl der Stellen nach dem Komma im Ergebnis einer arithmetischen Operation größer ist als die Stellenanzahl, die im Ergebnisbezeichner dafür vorgesehen wurde, werden Stellen entsprechend der vorgesehenen Stellenanzahl des Ergebnisbezeichners abgeschnitten. Soll gerundet werden, so wird der absolute Wert der letzten gültigen Ziffernstelle des Ergebnisbezeichners um 1 erhöht, wenn die höchstwertige der abzuschneidenden Ziffern größer oder gleich 5 ist.
2. Soll nicht gerundet, sondern Stellen abgeschnitten werden, so bleibt die letzte Ziffernstelle des Ergebnisbezeichners unverändert.
3. Wenn die niederwertigsten Ziffernstellen in einem Ergebnisbezeichner durch das Zeichen P in der Maskenzeichenfolge des Ergebnisbezeichners dargestellt sind, findet die Rundung oder Verkürzung gemäß der ganz rechts stehenden Ziffernstelle, für die Internspeicherplatz zugewiesen ist, statt (siehe Beispiel).

   ROUNDED wird für die Ergebnisfelder vom Typ COMPUTATIONAL-1 bzw.COMPUTATIONAL-2 angenommen und braucht für diese nicht angegeben zu werden.

Beispiel 8-18

errechnetesErgebnis [1)]	Beschreibung desErgebnisfeldes	Ergebnis nach dem Runden	Ergebnis ohne Runden
03&2627	PIC 99 PIC 99.9 PIC 99.99 PIC 99.999	03 03.3 03.26 03.263	03 03.2 03.26 03.262
123788&6	PIC S999PPP	124000	123000

[1)] & stellt den Rechendezimalpunkt dar.

8.7.4 ON SIZE ERROR-Angabe

Eine Überlaufbedingung liegt dann vor, wenn nach der Dezimalpunktausrichtung die Anzahl der ganzzahligen Stellen im errechneten Resultat größer ist als die dafür vorgesehene Stellenanzahl.

Syntaxregeln

1. Die Verletzung der Regeln für die Berechnung von Expontialausdrücken führt immer zum Abbruch der arithmetischen Operation und zum Auftreten einer Überlaufbedingung (siehe „Arithmetische Anweisungen", Allgemeine Regel 4).
2. Die Angabe ON SIZE ERROR spezifiert eine unbedingte Anweisung, die angibt, welche Aktionen im Falle eines Überlaufs auszuführen sind.
3. Die Überlaufbedingung bezieht sich nur auf das Endresultat einer arithmetischen Operation; sie bezieht sich nicht auf Zwischenergebnisse, außer in der MULTIPLY- und DIVIDE-Anweisung.
4. Ist ROUNDED angegeben, so findet das Runden vor der Prüfung auf Überlauf statt.
5. Ist ON SIZE ERROR angegeben, und eine Überlaufbedingung tritt nach der Ausführung der arithmetischen Anweisung auf, bleiben die Werte der Ergebnisbezeichner dieselben wie vor der Ausführung der Anweisung. Die Werte der Ergebnisbezeichner, für die keine Überlaufbedingung auftritt, bleiben unberührt von einem Überlauf, der für andere Ergebnisbezeichner während einer Operation auftritt. Nach Beendigung der arithmetischen Operationen geht die Ablaufsteuerung über zur in der ON SIZE ERROR-Angabe angegebenen unbedingten Anweisung. Die Ausführung wird gemäß den Regeln für diese Anweisung fortgesetzt. Wird dabei ein Prozedursprung oder eine Bedingungsanweisung ausgeführt, die explizit den Übergang der Ablaufsteuerung bewirkt, erfolgt dieser Übergang; im anderen Fall geht die Ablaufsteuerung nach Beendigung der Ausführung von unbedingte-anweisung der ON SIZE ERROR-Angabe zum Ende der arithmetischen Anweisung über, und die ON SIZE ERROR-Angabe wird, falls angegeben, ignoriert.
6. Ist ON SIZE ERROR nicht angegeben und es tritt eine Überlaufbedingung auf, sind die Werte der betroffenen Ergebnisbezeichner unbestimmt. Die Werte der Ergebnisbezeichner, für die keine Überlaufbedingung auftritt, bleiben unberührt von einem Überlauf, der für andere Ergebnisbezeichner während einer Operation auftritt. Nach Beendigung der Operationen geht die Ablaufsteuerung zum Ende der arithmetischen Anweisung über, und die NOT ON SIZE ERROR-Angabe wird, falls vorhanden, ignoriert.
7. Tritt keine Überlaufbedingung auf, geht die Ablaufsteuerung zum Ende der arithmetischen Anweisung oder, falls angegeben, zu unbedingte-anweisung der NOT ON SIZE ERROR-Angabe über. Im letzteren Fall wird die Ausführung gemäß den Regeln für die angegebene(n) unbedingte(n) Anweisung(en) fortgesetzt. Wird dabei ein Prozedursprung oder eine Bedingungsanweisung ausgeführt, die einen expliziten Übergang der Ablaufsteuerung bewirkt, erfolgt dieser Übergang; im anderen Fall geht nach Beendigung der Ausführung von unbedingte-anweisung der NOT ON SIZE ERROR-Angabe die Ablaufsteuerung zum Ende der arithmetischen Anweisung über.
8. Wenn für eine ADD-Anweisung mit der CORRESPONDING-Angabe oder für eine SUBTRACT-Anweisung mit der CORRESPONDING-Angabe ein Überlauf in einer der Teiloperationen auftritt, wird die unbedingte Anweisung in der ON SIZE ERROR-Angabe erst ausgeführt, wenn alle einzelnen Additionen oder Subtraktionen beendet sind.
9. Division durch Null bewirkt immer eine Überlaufbedingung.

   Für Datenfelder, die mit COMPUTATIONAL-1 oder COMPUTATIONAL-2 erklärt sind, wird bei der Division durch Null die Ausführung der unbedingten Anweisung in der ON SIZE ERROR-Angabe bewirkt.

Beispiel 8-19

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
```

```
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  A PIC 99 VALUE ZERO.
77  B PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    MOVE 44 TO A.
    MOVE 72 TO B.
    ADD A TO B
    ON SIZE ERROR
      PERFORM PROC-A
    END-ADD
    STOP RUN.
PROC-A.
    DISPLAY "Laengenfehler!" UPON T.
    DISPLAY A UPON T.
    DISPLAY B UPON T.
```

Aktueller Wert von A: 44
Aktueller Wert von B: 72
Errechnetes Ergebnis: 116

Das Ergebnisfeld B ist zu klein, um das errechnete Ergebnis aufzunehmen, und die Überlaufbedingung tritt ein. Da ON SIZE ERROR angegeben wurde, wird die Anweisung PERFORM PROC-A ausgeführt. Das Ergebnisfeld B bleibt unverändert.

8.8 Überlappende Operanden

Für alle Anweisungen gilt: Belegen ein Sende- und ein Empfangsfeld ganz oder teilweise denselben Speicherplatz, ohne in derselben Datenerklärung definiert zu sein, so ist das Ergebnis der Ausführung der Anweisung undefiniert. Bei einigen Anweisungen ist das Ergebnis der Ausführung auch dann undefiniert, wenn Sende- und Empfangsfeld in derselben Datenerklärung definiert sind (Näheres dazu siehe unter den einzelnen Anweisungen).

8.9 Inkompatible Daten

Mit Ausnahme der Überprüfung der Klassenbedingung gilt für jedes Ansprechen eines Datenfeldes in der PROCEDURE DIVISION: Falls der Inhalt eines Datenfeldes unverträglich ist mit seiner in der PICTURE-Klausel vereinbarten Datenklasse, ist das Ergebnis der Operation unvorhersehbar.

Allgemeine Regel

1. Jeder Operation mit einem numerischen Datenfeld, das eventuell einen nichtnumerischen Inhalt haben könnte (z.B. durch Redefinition des Datenfeldes oder nach Ausführung einer MOVE-Anweisung mit einem Gruppenfeld als Operanden), sollte der Klassentest IF NUMERIC vorangehen. Die Operation kann nur dann erfolgreich durchgeführt werden, wenn der Klassentest den Wahrheitswert TRUE ergeben hat.

8.10 Anweisungen

In diesem Kapitel werden folgende Themen behandelt:

8.10.1 ACCEPT-Anweisung

Funktion

Die ACCEPT-Anweisung überträgt geringe Datenmengen in ein Datenfeld. Die Daten werden entweder aus einer Systemdatei gelesen oder vom Compiler bzw. vom Betriebssystem zur Verfügung gestellt.

Format 1	liest über entsprechende Merknamen Benutzereingaben oder liefert Informationen des Betriebssystems und des Compilers.
Format 2	liefert Datums- und Uhrzeitangaben des Betriebssystems.
Format 3 & 4	dienen dem Zugriff auf die Kommandozeile des POSIX-Subsystems.
Format 5	liefert den Inhalt einer BS2000- oder POSIX-Umgebungsvariablen oder den Inhalt eines bestimmten Arguments von der POSIX-Kommandozeile.

Format 1

```
ACCEPT bezeichner [ FROM merkname ]
```

Syntaxregeln

1. bezeichner kann eine alphanumerische Datengruppe, ein alphabetisches, alphanumerisches, extern dezimales Datenfeld oder externes Gleitpunkt-Datenfeld sein.
   Ist merkname mit Betriebssysteminformationen verknüpft (COMPILER-INFO, PROCESS-INFO, TERMINAL-INFO oder DATE-ISO4), dann darf bezeichner auch ein nationales Datenfeld sein.
2. merkname muss im SPECIAL-NAMES-Paragrafen angegeben werden und mit einem der folgenden Herstellernamen verknüpft sein:

   ```
   SYSIPT
   TERMINAL
   CONSOLE
   job-variable-name (BS2000 job variable)
   COMPILER-INFO
   CPU-TIME, PROCESS-INFO, TERMINAL-INFO, DATE-ISO4
   ```

3. Mit der Angabe des Merknamens für SYSIPT, TERMINAL und CONSOLE wird eine Systemdatei spezifiziert, aus der die Daten gelesen werden sollen.
   SYSIPT bezeichnet die Systemdatei gleichen Namens.
   TERMINAL bezeichnet die (im Normalfall auf die Datensichtstation zugewiesene) Systemdatei SYSDTA.
   CONSOLE bezeichnet den Bedienplatz des Systems.
4. Mit der Angabe des Merknamens für einen Jobvariablennamen wird die zugehörige Jobvariable des Betriebssystems angesprochen, die eingelesen werden soll.
5. Mit der Angabe des Merknamens für COMPILER-INFO, CPU-TIME, PROCESS-INFO oder TERMINAL-INFO wird die Information spezifiziert, die angefordert werden soll. COMPILER-INFO bezeichnet Informationen, die der Compiler liefert.
   CPU-TIME, PROCESS-INFO, TERMINAL-INFO und DATE-ISO4 bezeichnen Informationen, die das Betriebssystem liefert.
6. Falls die FROM-Angabe fehlt, werden die Daten standardmäßig von der logischen Eingabedatei SYSIPT gelesen. Mit einer entsprechenden Compiler-Steueranweisung können die Daten auch von der logischen Eingabedatei SYSDTA gelesen werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).
7. bezeichner darf nicht mit der ANY LENGTH-Klausel definiert sein.
8. Die Ausführung der ACCEPT-Anweisung und der Aufbau der gelieferten Informationen für die einzelnen Funktionen sind im Handbuch „COBOL2000 Benutzerhandbuch“ [1] beschrieben.

Allgemeine Regeln

1. Die Daten werden linksbündig in den durch bezeichner angegebenen Bereich übertragen, unabhängig von der für bezeichner angegebenen Maskenzeichenfolge. Für die eingelesenen Daten wird dabei weder Druckaufbereitung noch Fehlerkontrolle vorgenommen.
   Ausgenommen davon ist CPU-TIME. Die CPU-Zeit wird entsprechend den Regeln der MOVE-Anweisung von einem Sendefeld mit der Beschreibung PIC 9(6)V9(4) übertragen.
   Ist bezeichner ein null-längiges Datenfeld, so werden keine Daten übertragen.
2. Falls die für eine ACCEPT-Anweisung verwendete Systemdatei identisch ist mit einer in einer READ-Anweisung verwendeten Systemdatei, so ist das Ergebnis unbestimmt.
3. Eine ACCEPT-Anweisung für Jobvariable wird zur Ablaufzeit mit einer Fehlermeldung zurückgewiesen, wenn die Jobvariablen im Betriebssystem nicht vorhanden sind. Kann die Jobvariable aus bestimmten Gründen nicht eingelesen werden, gibt das Laufzeitsystem eine Fehlermeldung aus. Abhängig von einer entsprechenden Compiler-Steueranweisung (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]) wird das Programm fortgesetzt oder abgebrochen. Im Fortsetzungsfall wird entsprechend der allgemeinen Regel 1 das Literal "/*" in den bezeichner übertragen.

Beispiel 8-20

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
...
PROCEDURE DIVISION.
    ...
    ACCEPT EINGABE FROM T.
```

Der Merkname T wird im SPECIAL-NAMES-Paragrafen mit dem Herstellernamen TERMINAL verknüpft. Die folgende ACCEPT-Anweisung fordert Daten aus der TERMINAL zugewiesenen Systemdatei SYSDTA an und überträgt diese in das mit EINGABE bezeichnete Datenfeld.

Format 2

```
ACCEPT bezeichner FROM {DATE | DAY | DAY-OF-WEEK | TIME}
```

Syntaxregeln

1. bezeichner darf kein alphabetisches Datenelement sein.
2. Durch die ACCEPT-Anweisung wird die angeforderte Information in das durch bezeichner angegebene Datenfeld, entsprechend den für die MOVE-Anweisung geltenden Regeln, übertragen. DATE, DAY, DAY-OF-WEEK und TIME sind besondere Datenfelder und werden daher nicht in der Übersetzungseinheit beschrieben.

Allgemeine Regeln

1. DATE setzt sich zusammen aus den Datenelementen Jahr innerhalb eines Jahrhunderts, Monat des Jahres und Tag des Monats. Die Reihenfolge dieser gedachten
   Datenelemente ist von links nach rechts Jahr, Monat, Tag. Daher würde z.B. der 1. April 1996 dargestellt als 960401. Falls DATE in einem COBOL-Programm angesprochen wird, wird es interpretiert als wäre es als sechsstelliges, ganzzahliges, numerisches Datenelement ohne Vorzeichen erklärt worden (PIC 9(6)).
2. DAY setzt sich zusammen aus den Datenelementen Jahr des Jahrhunderts und Tag des Jahres. Die Reihenfolge dieser gedachten Datenelemente ist von links nach rechts Jahr und Tag des Jahres. Daher

würde z.B. der 1. April 1998 dargestellt als 98091. Falls DAY in einem COBOL-Programm angesprochen wird, wird es interpretiert, als wäre es als fünfstelliges, ganzzahliges, numerisches Datenelement ohne Vorzeichen erklärt worden (PIC 9(5)).

3. DAY-OF-WEEK besteht aus einem einzigen Datenelement, dessen Inhalt der Wochentag ist. Falls DAY-OF-WEEK in einem COBOL-Programm angesprochen wird, wird es interpretiert, als wäre es als vorzeichenloses, ganzzahliges, numerisches Datenelement von der Länge einer Ziffernstelle beschrieben worden (PIC 9).
   Bei DAY-OF-WEEK bedeutet der Wert 1 Montag, Wert 2 Dienstag ... Wert 7 Sonntag.
4. TIME setzt sich zusammen aus den Datenelementen Stunden, Minuten, Sekunden und Hundertstel-Sekunden. TIME basiert auf einer 24-Stunden-Zeitangabe; 2 Uhr 41 Minuten nachmittags wird also dargestellt als 14410000. Falls TIME in einem COBOL-Programm angesprochen wird, wird es interpretiert als wäre es als achtstelliges, ganzzahliges, numerisches Datenelement ohne Vorzeichen erklärt worden (PIC 9(8)). Der Minimalwert, den TIME enthalten kann, ist 00000000; der Maximalwert 23595900.
   Die beiden letzten Ziffernstellen werden vom System nicht geliefert und sind daher immer auf Null gesetzt.

Die folgenden drei Formate der ACCEPT-Anweisung sind Erweiterungen aus dem X/OPEN Portability Guide. Sie erlauben den Zugriff auf Umgebungsvariablen und Kommandozeilen. Der Zugriff auf eine Kommandozeile ist nur sinnvoll, wenn das Zielprogramm im POSIX-Subsystem abläuft, das ab BS2000/OSD 2.0 zur Verfügung steht.
Der Ablauf des COBOL2000-Compilers und der von ihm erzeugten Programme unter POSIX ist im Handbuch „COBOL2000 Benutzerhandbuch“ [1] beschrieben.

Format 3

liefert die aktuelle Anzahl der Argumente in der Kommandozeile.

```
ACCEPT bezeichner-1 [ FROM merkname-3 ]
    [END-ACCEPT]
```

Syntaxregeln

1. bezeichner-1 muss sich auf ein Datenelement beziehen, das als Ganzzahl ohne Vorzeichen beschrieben ist.
2. merkname-3 muss im SPECIAL-NAMES-Paragrafen mit dem Herstellernamen ARGUMENT-NUMBER verknüpft sein.

Format 4

liefert nacheinander die Inhalte der Argumente in der Kommandozeile.

```
ACCEPT bezeichner-2 [ FROM merkname-4 ]
    [ON EXCEPTION unbedingte-anweisung-1 [NOT ON EXCEPTION unbedingte-anweisung-2]]
    [END-ACCEPT]
```

Syntaxregeln

1. bezeichner-2 muss sich auf ein alphanumerisches Datenelement beziehen, das ohne die ANY LENGTH-Klausel definiert ist.
2. merkname-4 muss im SPECIAL-NAMES-Paragrafen mit dem Herstellernamen ARGUMENT-VALUE verknüpft sein.
3. NOT ON EXCEPTION darf nur angegeben werden, wenn auch ON EXCEPTION angegeben ist.

Format 5

liefert den Inhalt einer Umgebungsvariablen oder den Inhalt eines bestimmten Arguments von der Kommandozeile. Der Name der angesprochenen Umgebungsvariablen bzw. die Nummer des angesprochenen Arguments in der Kommandozeile muss zuvor durch entsprechende DISPLAY-Anweisungen festgelegt werden.

```
ACCEPT bezeichner-2 [ FROM merkname-6 ]
    [ON EXCEPTION unbedingte-anweisung-1 [NOT ON EXCEPTION unbedingte-anweisung-2]]
    [END-ACCEPT]
```

Syntaxregeln

1. bezeichner-2 muss sich auf ein alphanumerisches Datenelement beziehen, das ohne die ANY LENGTH-Klausel definiert ist.
2. merkname-6 muss im SPECIAL-NAMES-Paragrafen mit den Herstellernamen ARGUMENT-VALUE oder ENVIRONMENT-VALUE verknüpft sein.
3. NOT ON EXCEPTION darf nur angegeben werden, wenn auch ON EXCEPTION angegeben ist.

> Ein ausführliches Beispiel für den Zugriff auf Kommandozeile und Umgebungsvariable befindet sich im Handbuch „COBOL2000 Benutzerhandbuch“ [1].

8.10.2 ADD-Anweisung

Funktion

Die ADD-Anweisung addiert zwei oder mehr numerische Operanden und speichert das Ergebnis ab.

Format 1	die ADD-Anweisung speichert die Summe im jeweiligen Operandenfeld ab. Mehrere Additionen können mit einer ADD-Anweisung durchgeführt werden, indem mehr als ein Ergebnisfeld angegeben wird.
Format 2	die ADD-Anweisung speichert die Summe in einem getrennten Ergebnisfeld ab.
Format 3	die ADD-Anweisung addiert die Elemente einer Datengruppe zu den entsprechenden Datenelementen einer anderen Gruppe.

Format 1

```
ADD {bezeichner-1 | literal-1}... TO {bezeichner-2 [ROUNDED]}...
    [ON SIZE ERROR unbedingte-anweisung-1]
    [NOT ON SIZE ERROR unbedingte-anweisung-2]
    [END-ADD]
```

Syntaxregeln

1. Jeder Bezeichner muss sich auf ein numerisches Datenfeld beziehen.
2. Die gemeinsame Stellenzahl der Operanden, die in einer gegebenen Anweisung ermittelt wird, darf nicht mehr als 31 Ziffern betragen (siehe „Arithmetische Anweisungen").
3. Die Werte der Operanden, die dem Wort TO vorangehen, werden addiert und die Summe zu dem derzeitigen Wert von bezeichner-2... dazuaddiert. Das Ergebnis wird in bezeichner-2 ... abgespeichert.

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die ROUNDED-Angabe und (NOT) ON SIZE ERROR-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-21

Anweisung	Maskenzeichenfolge des Ergebnisfeldes	Rechnung
ADD A, B TO C, D		A + B + C abgespeichert in CA + B + D abgespeichert in D
ADD A, B, C TO D	S9999V99	A + B + C + D abgespeichert in D als SnnnnVnn
ADD A, 14 TO C ROUNDED	99999	A + 14 + C abgespeichert in C als nnnnn, wobei, wenn nötig, gerundet wird.

Format 2

```
ADD {bezeichner-1 | literal-1}... TO {bezeichner-2 | literal-2} GIVING {bezeichner-3 [ROUNDED]}...
    [ON SIZE ERROR unbedingte-anweisung-1]
    [NOT ON SIZE ERROR unbedingte-anweisung-2]
    [END-ADD]
```

Syntaxregeln

1. Jeder Bezeichner, der dem Wort GIVING vorangeht, muss sich auf ein numerisches Datenelement beziehen.
2. bezeichner-3 kann sich auf ein numerisches oder numerisch druckaufbereitetes Datenelement beziehen.
3. Die gemeinsame Stellenzahl der Operanden, die in einer gegebenen Anweisung ermittelt wird, mit Ausnahme des Datenfeldes, das dem Wort GIVING folgt, darf nicht mehr als 31 Ziffern betragen (siehe „Arithmetische Anweisungen").
4. Die Werte der Operanden, die dem Wort GIVING vorangehen, werden addiert; danach wird die Summe als neuer Wert in bezeichner-3 abgespeichert.

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die GIVING-Angabe, ROUNDED-Angabe und (NOT) ON SIZE ERROR-Angabe sind in diesem

Beispiel 8-22

Anweisung	Maskenzeichenfolge des Ergebnisfeldes	Rechnung
ADD A, B, C GIVING D.	9999.99	A + B + C abgespeichert in D als nnnn.nn
ADD A, B, 43.6 GIVING D ON SIZE ERROR GO TO UEBERLAUF END-ADD.	99V99	A + B + 43.6 abgespeichert in D. Wenn die Anzahl der ganzzahligen Stellen im Ergebnis größer als 2 ist, tritt die Überlaufbedingung ein und die in SIZE ERROR angegebene GO TO-Anweisung wird ausgeführt.

Format 3

```
ADD {CORR | CORRESPONDING} bezeichner-1 TO bezeichner-2 [ROUNDED]
    [ON SIZE ERROR unbedingte-anweisung-1]
    [NOT ON SIZE ERROR unbedingte-anweisung-2]
    [END-ADD]
```

Syntaxregeln

1. Jeder Bezeichner muss sich auf eine Datengruppe beziehen.
2. Die gemeinsame Stellenzahl der Operanden, die für jedes Paar von entsprechenden Datenfeldern getrennt bestimmt wird, darf nicht mehr als 31 Ziffern betragen (siehe „Arithmetische Anweisungen").
3. Datenelemente innerhalb des ersten Operanden (bezeichner-1) werden zu den entsprechenden Datenelementen im zweiten Operanden (bezeichner-2 ...) addiert. Die Ergebnisse werden in den Feldern des zweiten Operanden abgespeichert.

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die CORRESPONDING-Angabe, ROUNDED-Angabe und (NOT) ON SIZE ERROR-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-23

Siehe die Beschreibung der CORRESPONDING-Angabe als Beispiel für die Verwendung dieses Zusatzes ("CORRESPONDING-Angabe ").

8.10.3 ALLOCATE-Anweisung

Funktion

Die ALLOCATE-Anweisung weist dynamischen Speicher zu.

Format 1 Wird Speicher für eine Datenbeschreibung (datenname-1) angefordert, für die die BASED-Klausel spezifiziert ist, so wird dieser Datenbeschreibung die Adresse des angeforderten Speichers zugeordnet.
Ist die RETURNING-Angabe vorhanden, enthält der Datenzeiger bezeichner-2 ebenfalls die Adresse des zugewiesenen Speichers.

Format 2 Wird Speicher als eine Anzahl von Zeichen angefordert, wird die Adresse des zugewiesenen Speichers im Datenzeiger bezeichner-2 zurückgegeben.

Format 1

```
ALLOCATE datenname-1
    [INITIALIZED] [RETURNING bezeichner-2]
```

Syntaxregeln

1. datenname-1 muss ein Datenelement sein, das mit der BASED-Klausel spezifiziert ist.
2. Ist bezeichner-2 ein typbezogener Datenzeiger, so muss datenname-1 ebenfalls typisiert sein und die zugehörigen Typen müssen übereinstimmen.
3. Ist datenname-1 stark typisiert, so muss auch bezeichner-2 ein typbezogener Zeiger sein, dem der gleiche Typ zugeordnet ist wie datenname-1.
4. bezeichner-2 muss ein Datenelement der Kategorie datenzeiger referenzieren.

Format 2

```
ALLOCATE arithmetischer-ausdruck CHARACTERS
    [INITIALIZED] RETURNING bezeichner-2
```

Syntaxregeln

1. bezeichner-2 darf kein typbezogener Datenzeiger sein.
2. bezeichner-2 muss ein Datenelement der Kategorie datenzeiger referenzieren.

Allgemeine Regeln für beide Formate

1. arithmetischer-ausdruck legt die Anzahl von Bytes im Speicher fest, die angefordert werden. Das Resultat der Auswertung von arithmetischer-ausdruck wird in einem internen Datenfeld abgelegt, das mit `PIC S9(9) BINARY` definiert ist.
2. Ist das Resultat der Auswertung kleiner oder gleich NULL wird kein Speicher angefordert und bezeichner-2 enthält den vordefinierten Wert NULL.
3. Bei Format 1 richtet sich die Größe des angeforderten Speichers danach, wieviel Platz für eine Datenbeschreibung datenname-1 benötigt wird. Wenn eine Datenbeschreibung, die zu datenname-1 untergeordnet ist, eine OCCURS DEPENDING-Klausel enthält, wird die maximale Länge des Datensatzes angefordert.
4. Ist der angeforderte Speicherplatz verfügbar, so wird er zugewiesen. Bei Format 1 wird die Adresse von datenname-1 auf die Adresse des zugewiesenen Speichers gesetzt. Der Datenzeiger bezeichner-2 aus der RETURNING-Angabe wird auf die Adresse des zugewiesenen Speichers gesetzt.

5. Ist der angeforderte Speicherplatz nicht verfügbar, wird die Adresse von datenname-1 und/oder der Datenzeiger bezeichner-2 aus der RETURNING-Angabe auf NULL gesetzt. Es entsteht die Ausnahmesituation EC-STORAGE-NOT-AVAIL. Ist die Überprüfung für EC-STORAGE-NOT-AVAIL eingeschaltet, so wird der zugehörige Ausnahmezustand ausgelöst und in die entsprechende USE-Prozedur verzweigt. Nach Rücksprung aus der USE-Prozedur wird bei der auf die ALLOCATE-Anweisung folgende ausführbaren Anweisung fortgesetzt.
6. Die INITIALIZED-Angabe bei Format 1 bewirkt, dass für datenname-1 eine INITIALIZE-Anweisung INITIALIZE datenname-1 WITH FILLER ALL TO VALUE THEN TO DEFAULT ausgeführt wird.
7. Fehlt die INITIALIZED-Angabe bei Format 1, so wird eine INITIALIZE-Anweisung INITIALIZE datenname-1 WITH FILLER REPLACING DATA-POINTER BY NULL, PROGRAM-POINTER BY NULL, OBJECT-REFERENCE BY NULL ausgeführt.
8. Die INITIALIZED-Angabe bei Format 2 bewirkt, dass der zugewiesene Speicher mit binären NULLEN initialisiert wird.
9. Fehlt die INITIALIZED-Angabe bei Format 2, ist der Inhalt des zugewiesenen Speichers undefiniert.
10. Der Speicher bleibt zugewiesen, bis er entweder explizit mit einer FREE-Anweisung freigegeben wird oder der Programmlauf beendet wird.

8.10.4 ALTER-Anweisung

Funktion

Die ALTER-Anweisung modifiziert eine oder mehrere GO TO-Anweisungen und verändert so eine früher festgelegte Folge von Operationen.

Format

```
ALTER {prozedurname-1 TO [PROCEED TO] prozedurname-2}...
```

Syntaxregeln

1. prozedurname-1... müssen Namen von Paragrafen sein, die nur einen Programmsatz enthalten, der aus einer GO TO-Anweisung ohne DEPENDING-Angabe besteht.
2. prozedurname-2... müssen Paragrafen- oder Kapitelnamen in der PROCEDURE DIVISION sein.
3. Während der Ausführung des Programms modifiziert die ALTER-Anweisung die unter prozedurname-1... angegebene GO TO-Anweisung, indem sie die Sprungziele entsprechend durch prozedurname-2... ersetzt (siehe „GO TO-Anweisung").

Allgemeine Regeln

1. Eine GO TO-Anweisung in einem Kapitel, dessen Segmentnummer 50 oder größer ist, darf nicht von einer ALTER-Anweisung in einem Kapitel mit einer anderen Segmentnummer angesprochen werden.
2. Eine GO TO-Anweisung in einem Kapitel, dessen Segmentnummer kleiner als 50 ist, darf von einer ALTER-Anweisung in jedem beliebigen Kapitel angesprochen werden, selbst dann, wenn die von der ALTER-Anweisung angesprochene GO TO-Anweisung in einem Programmsegment ist, das noch nicht zum Ablauf aufgerufen worden ist.

8.10.5 CALL-Anweisung

Funktion

Die CALL-Anweisung übergibt die Steuerung an ein aufgerufenes Programm. Wahlweise können Operanden angegeben werden, um dem aufgerufenen Programm den Zugriff auf Daten des aufrufenden Programms zu ermöglichen.
Mit der CALL-Anweisung (Format 4) ist es auch möglich von einem COBOL-Programm aus BS2000-Systemkommandos auszuführen.

Format 1

```
CALL {bezeichner-1 | literal-1} [USING { [BY REFERENCE] {bezeichner-2|dateiname-1
}...
                                        | BY VALUE {bezeichner-5}...
                                        | BY CONTENT {bezeichner-2|literal-2}...
                                       }...]
  [RETURNING bezeichner-3]
  [ON OVERFLOW unbedingte-anweisung-1]
  [END-CALL]
```

Format 2

```
CALL {bezeichner-1 | literal-1} [USING { [BY REFERENCE] {bezeichner-2|dateiname-1
}...
                                        | BY VALUE {bezeichner-5}...
                                        | BY CONTENT {bezeichner-2|literal-2}...
                                       }...]
  [RETURNING bezeichner-3]
  [ON EXCEPTION unbedingte-anweisung-1]
  [NOT ON EXCEPTION unbedingte-anweisung-2]
  [END-CALL]
```

Syntaxregeln für beide Formate

1. literal-1 muss ein alphanumerisches Literal sein. Es darf jedoch keine figurative Konstante sein.

   Ist literal-1 der Programmname eines einzelnen Programms bzw. des äußersten Programms eines geschachtelten Programms, muss es mit einem alphabetischen Zeichen beginnen und darf nur Großbuchstaben und Ziffern enthalten. Die Namenslänge ist abhängig vom Modulformat (siehe COBOL-Benutzerhandbuch [1]).
   Ist literal-1 der Programmname für ein inneres Programm eines geschachtelten Programms, muss es mit einem alphabetischen Zeichen beginnen, darf Groß- und Kleinbuchstaben sowie Ziffern enthalten und darf eine Länge von maximal 30 Zeichen haben.
2. bezeichner-1 muss als Programmzeiger oder als alphanumerisches Datenfeld definiert sein. Der Inhalt des alphanumerischen Datenfeldes muss ein gültiger Programmname, wie in 1. beschrieben, sein. Der Inhalt des Programmzeigers wird als Adresse des Einsprungpunkts interpretiert.
3. Die USING-Angabe in einer CALL-Anweisung darf nur angegeben werden, wenn nach der dazugehörenden PROCEDURE DIVISION-Überschrift oder in der ENTRY-Anweisung des aufgerufenen Programms eine USING-Angabe geschrieben wurde. Die Anzahl der Operanden in jeder USING-Angabe muss gleich sein, sonst ist das Ergebnis undefiniert.
4. Jede in der USING-Angabe angegebene Operand muss in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION, LINKAGE SECTION oder SUB-SCHEMA SECTION definiert sein. Er kann die Stufennummer 01 oder 77 haben. Der Compiler erlaubt jedoch jede Stufennummer außer

88.
Für die Ausrichtung von Datenelementen mit USAGE INDEX, BINARY, COMPUTATIONAL, COMPUTATIONAL-5, COMPUTATIONAL-1, COMPUTATIONAL-2 im LINKAGE-Kapitel werden alle Elemente auf der 01-Stufe als auf Doppelwortgrenze ausgerichtet angenommen. Deshalb muss der Benutzer sicherstellen, dass diese Operanden in der USING-Angabe entsprechend ausgerichtet sind.

5. dateiname-1 ist nur sinnvoll, wenn das aufgerufene Programm in einer anderen Programmiersprache als COBOL geschrieben ist.
6. bezeichner-2 darf kein Funktionsbezeichner sein.
7. bezeichner-3 darf nicht in der REPORT SECTION definiert sein.
8. bezeichner-3 ist ein Empfangsfeld.
9. Als Parameter dürfen auch Objektreferenzen, Zeiger oder stark typisierte Datengruppen verwendet werden. Sie dürfen nur BY CONTENT als USING-Parameter übergeben werden.
10. bezeichner-5 muss als 2 oder 4 Byte langes Datenfeld mit USAGE COMP-5 oder als 1 Byte langes Datenfeld definiert sein. Andernfalls ist das Ergebnis der Parameterübergabe unbestimmt.
Diese Art der Parameterübergabe ist nur sinnvoll, wenn das aufgerufene Programm in einer anderen Sprache als COBOL geschrieben ist (siehe Beispiel 8-25).
11. literal-2 muss ein alphanumerisches oder nationales Literal sein oder eine der figurativen Konstanten SPACE, LOW-VALUE, HIGH-VALUE.
12. bezeichner-2, bezeichner-3 und bezeichner-5 dürfen nicht mit der ANY LENGTH-Klausel definiert sein.

Allgemeine Regeln für beide Formate

1. literal-1 oder bezeichner-1 enthält den Namen des aufgerufenen Programms. Das Programm, das eine CALL-Anweisung enthält, ist das aufrufende Programm. Ist das aufgerufene Programm ein COBOL-Programm, müssen literal-1 bzw. bezeichner-1 den im PROGRAM-ID-Paragrafen oder in der ENTRY-Anweisung stehenden Programmnamen enthalten. Ist bezeichner-1 als Programmzeiger definiert, dann enthält er die Adresse eines Einsprungpunktes in ein Programm.
2. Bei der Ausführung der CALL-Anweisung wird die Steuerung an das aufgerufene Programm übergeben. Nach der Rückgabe der Steuerung vom aufgerufenen Programm wird, falls vorhanden, unbedingte-anweisung-2 ausgeführt und dann zum Ende der CALL-Anweisung verzweigt. Falls NOT ON EXCEPTION fehlt, wird zum Ende der CALL-Anweisung verzweigt.
3. Ist während der Ausführung einer CALL-Anweisung das aufgerufene Programm nicht verfügbar oder bereits aktiv, ohne dass es rekursiv ist, wird eine der folgenden Aktionen durchgeführt:
   a. Ist ON OVERFLOW bzw. ON EXCEPTION angegeben, geht die Steuerung zu unbedingte-anweisung-1 über. Nach Beendigung von unbedingte-anweisung-1 geht die Steuerung zum Ende der CALL-Anweisung über.
   b. Ist ON OVERFLOW bzw. ON EXCEPTION nicht angegeben, wird der Programmablauf nach Ausgabe einer Fehlermeldung abgebrochen.
4. Haben zwei Programme in einer Ablaufeinheit denselben Namen, so muss mindestens eines davon ein inneres Programm eines geschachtelten Programms sein. Der Aufruf eines Programms, das einen mehrfach verwendeten Programmnamen besitzt, ist nur unter folgenden Bedingungen erfolgreich:
   a. Das aufgerufene Programm ist in dem aufrufenden Programm direkt enthalten.
   b. Das aufgerufene Programm besitzt das COMMON-Attribut und wird vom direkt übergeordneten Programm oder von einem seiner Geschwisterprogramme oder deren Abkömmlingen aufgerufen.
   c. Das aufrufende Programm ist ein inneres Programm eines Schachtelprogramms und ruft ein getrennt übersetztes Programm einer Ablaufeinheit auf.
5. Die vom aufrufenden Programm an das aufgerufene Programm als Parameter zu übergebenden Daten werden in der USING-Angabe der CALL-Anweisung mit bezeichner-2... angegeben. Anzahl und Reihenfolge der Parameter müssen mit denen in der USING-Angabe der PROCEDURE DIVISION-

Überschrift bzw. der ENTRY-Anweisung übereinstimmen. Eine Ausnahme bilden die Indizes, die Tabellen zugeordnet sind (INDEXED BY-Angabe): Die Indizes in einem aufrufenden und in einem aufgerufenen Programm weisen immer auf getrennte Indizes hin.

6. Wird dateiname-1 in der Liste der USING-Angabe angegeben, wird an das aufgerufene Programm die Anfangsadresse des System-Dateisteuerungsblocks der jeweiligen Datei übergeben.
7. Die Angaben BY CONTENT und BY REFERENCE können zusammen verwendet werden; die Angabe BY CONTENT oder BY REFERENCE gilt für alle folgenden Parameter, bis eine andere BY CONTENT- oder BY REFERENCE-Angabe hinzukommt. Ist weder BY CONTENT noch BY REFERENCE angegeben, wird BY REFERENCE angenommen.
8. Wird ein Parameter BY REFERENCE übergeben, so belegt er im aufrufenden und im aufgerufenen Programm denselben Speicherplatz. Die Beschreibung des Datenfeldes im aufgerufenen Programm muss dieselbe Anzahl von Zeichenpositionen aufweisen wie die des entsprechenden Datenfeldes im aufrufenden Programm.
9. Wird ein Parameter BY CONTENT übergeben, erstellt das aufrufende Programm eine Kopie des Parameters und übergibt die Kopie des Parameters BY REFERENCE.
   Die Datenbeschreibung des entsprechenden Parameters im aufgerufenen Programm muss dabei wie folgt gewählt werden:
   - bei Angabe von `bezeichner-2:`die gleiche wie die von `bezeichner-2`
   - bei Angabe von figurativen Konstanten: PIC X
   - bei Angabe eines alphanumerischen Literals: PIC X(n). Der Wiederholungsfaktor n darf dabei höchstens so groß wie die Länge des Literals sein. Im Gegensatz zum Verhalten bei sonstigen Übertragungen, kann hier das Literal am Ende nicht mit Leerzeichen auf die Länge des entsprechenden Parameters aus dem Unterprogramm aufgefüllt werden.
   - bei Angabe eines nationalen Literals: PIC N(n). Der Wiederholungsfaktor n darf dabei höchstens so groß wie die Länge des Literals sein. Im Gegensatz zum Verhalten bei sonstigen Übertragungen, kann hier das Literal am Ende nicht mit Leerzeichen auf die Länge des entsprechenden Parameters aus dem Unterprogramm aufgefüllt werden.
10. Die Angabe BY VALUE ermöglicht die C-konforme, direkte Wertübergabe an C-Programme. Die Parameterübergabe „by value“ bewirkt, dass nur der Wert des Parameters an das aufgerufene C-Programm übergeben wird. Das aufgerufene Programm kann auf diesen Wert zugreifen und ihn verändern, wobei der Wert des Parameters im COBOL-Programm unverändert bleibt.
11. Aufgerufene Programme können CALL-Anweisungen enthalten. Ein aufgerufenes Programm darf aber keine CALL-Anweisung ausführen, die direkt oder indirekt das aufrufende Programm über die Standard-Einsprungstelle oder eine mit der ENTRY-Anweisung generierte Einsprungstelle aufruft.
12. Ist die RETURNING-Angabe definiert, wird das Ergebnis des aktuellen Programms in bezeichner-5 abgelegt.

Format 3

```
CALL { bezeichner-1 | literal-1 } AS { NESTED | programmprototypname-1 }
       [USING {  [BY REFERENCE] { bezeichner-2 | OMITTED }
                | [BY  CONTENT] { bezeichner-4 | arithmetischer-ausdruck-1 |
literal-2 }
                | [BY  VALUE] { bezeichner-4 | arithmetischer-ausdruck-1 | literal-
2 }
              }...]
[RETURNING bezeichner-3]
[ON EXCEPTION unbedingte-anweisung-1]
[NOT ON EXCEPTION unbedingte-anweisung-2]
[END-CALL]
```

Syntaxregeln

1. literal-1 muss ein alphanumerisches Literal sein. Es darf jedoch keine figurative Konstante sein.

   Ist literal-1 der Programmname eines einzelnen Programms bzw. des äußersten Programms eines geschachtelten Programms, muss es mit einem alphabetischen Zeichen beginnen und darf nur Großbuchstaben und Ziffern enthalten. Die Namenslänge ist abhängig vom Modulformat (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

   Ist literal-1 der Programmname für ein inneres Programm eines geschachtelten Programms, muss es mit einem alphabetischen Zeichen beginnen, darf Groß- und Kleinbuchstaben sowie Ziffern enthalten und darf eine Länge von maximal 30 Zeichen haben.
2. bezeichner-1 muss als Programmzeiger oder als alphanumerisches Datenfeld definiert sein. Als alphanumerisches Datenfeld muss sein Wert ein gültiger Programmname, wie in 1. beschrieben, sein. Der Inhalt des Programmzeigers wird als Adresse des Einsprungpunkts interpretiert.
3. Die NESTED-Angabe darf nur in einer Programm-Definition spezifiziert werden.
4. NESTED darf nur zusammen mit literal-1 angegeben werden. literal-1 muss identisch sein mit programm-name, der im PROGRAM-ID Paragrafen eines geschachtelten Programms definiert ist.
5. Ist programmprototypname-1 angegeben, so muss im REPOSITORY-Paragrafen ein Eintrag für diesen Namen vorhanden sein.
6. bezeichner-4 und jeder Bezeichner in arithmetischer-ausdruck-1 ist ein sendender Operand.
7. Ist BY CONTENT oder BY REFERENCE für ein Argument spezifiziert, so muss die BY REFERENCE-Angabe in der PROCEDURE DIVISION-Überschrift für den entsprechenden formalen Parameter gelten.
8. BY CONTENT darf nicht ausgelassen werden, wenn bezeichner-4 als Empfangsfeld zulässig ist.
9. Ist BY VALUE für ein Argument angegeben, so muss die BY VALUE-Angabe in der PROCEDURE DIVISION-Überschrift für den entsprechenden formalen Parameter angegeben sein.
10. Ist bezeichner-4 oder der entsprechende formale Parameter in der PROCEDURE DIVISION-Überschrift mit BY VALUE angegeben, so muss bezeichner-4 von der Klasse numerisch, objekt oder zeiger sein.
11. Ist OMITTED angegeben oder ein Argument ganz weggelassen, so muss in der PROCEDURE DIVISION-Überschrift für den entsprechenden formalen Parameter die Angabe OPTIONAL gemacht sein.
12. bezeichner-2 und bezeichner-3 dürfen nicht mit der ANY LENGTH-Klausel definiert sein.
13. Wenn bezeichner-2 oder der korrespondierende formale Parameter mit einer BY VALUE-Angabe spezifiziert ist, darf er nur von der Klasse numerisch, objekt oder zeiger sein.
14. bezeichner-2 muss auf einen Adressbezeichner oder ein Datenelement verweisen, die in der File Section, Working-Storage Section, Local-Storage Section oder Linkage Section definiert sind.
15. Wenn die BY REFERENCE-Angabe für bezeichner-2, der kein Adressbezeichner ist, spezifiziert oder impliziert ist, stellt bezeichner-2 sowohl ein Sende-, als auch Empfangsfeld dar. Andernfalls spezifiziert bezeichner-2 ein Sendefeld**.**

   Anmerkung:
   Die Angabe BY REFERENCE ADDRESS OF datenname wird behandelt wie BY CONTENT ADDRESS OF datenname.
16. bezeichner-3 darf nicht in der REPORT SECTION definiert sein.
17. bezeichner-3 ist ein Empfangsfeld.
18. Konformität von Parametern und Rückgabe-Elementen (siehe "Konformität von Parametern und Rückgabe-Elementen ") muss gewährleistet sein.

Allgemeine Regeln

Grundsätzlich sind auch die Regeln für Format 1 und 2 zutreffend. Zusätzlich gelten noch folgende Regeln:

1. Die Angaben BY REFERENCE, BY CONTENT und BY VALUE beziehen sich **nur** auf das direkt nachfolgende Argument.
2. Ist ein Argument weder mit BY REFERENCE oder BY CONTENT noch mit BY VALUE angegeben, so wird es folgendermaßen behandelt:
   a. es wird BY REFERENCE angenommen, wenn die BY REFERENCE-Angabe für den entsprechenden formalen Parameter in der PROCEDURE DIVISION-Überschrift definiert und das Argument ein Bezeichner ist, der als Empfangsfeld zugelassen ist.
   b. es wird BY CONTENT angenommen, wenn die BY REFERENCE-Angabe für den entsprechenden formalen Parameter in der PROCEDURE DIVISION-Überschrift definiert ist und das Argument ein Literal, ein arithmetischer Ausdruck oder sonst ein Bezeichner ist, der nicht als Empfangsfeld zugelassen ist.
   c. es wird BY VALUE angegeben, wenn die BY VALUE-Angabe für den entsprechenden formalen Parameter in der PROCEDURE DIVISION-Überschrift angegeben ist.
3. Ein Argument am Ende der USING-Liste darf auch ganz weggelassen werden (d.h. es muss dafür nicht OMITTED angegeben werden), wenn auch alle nachfolgenden Argumente weggelassen worden sind (Syntaxregel 11 beachten!).
4. Ist OMITTED angegeben oder ist ein Argument ganz weggelassen, dann ist die Omitted-Argument Bedingung für dieses Argument in einem gerufenen Programm wahr.
5. Ist für ein Argument die Omitted-Argument Bedingung wahr und wird darauf in einem gerufenen Programm (außer in einer Omitted-Argument Bedingung) Bezug genommen, so ist das Verhalten undefiniert.
6. Bei der Ausführung der CALL-Anweisung wird die Steuerung an das aufgerufene Programm übergeben. Nach der Rückgabe der Steuerung vom aufgerufenen Programm wird, falls vorhanden, unbedingte-anweisung-2 ausgeführt und dann zum Ende der CALL-Anweisung verzweigt. Falls NOT ON EXCEPTION fehlt, wird zum Ende der CALL-Anweisung verzweigt.
7. Ist während der Ausführung einer CALL-Anweisung das aufgerufene Programm nicht verfügbar oder bereits aktiv, ohne dass es rekursiv ist, wird eine der folgenden Aktionen durchgeführt:
   a. Ist ON EXCEPTION angegeben, geht die Steuerung zu unbedingte-anweisung-1 über. Nach Beendigung von unbedingte-anweisung-1 geht die Steuerung zum Ende der CALL-Anweisung über.
   b. Ist ON EXCEPTION nicht angegeben, wird der Programmablauf nach Ausgabe einer Fehlermeldung abgebrochen.

Format 4

```
CALL UPON SYSTEM USING { bezeichner-1 | literal-1 } [bezeichner-2]

   [STATUS bezeichner-3]

   [ON EXCEPTION unbedingte-anweisung-1]
  [NOT ON EXCEPTION unbedingte-anweisung-2]

       [END-CALL]
```

Syntaxregeln

1. bezeichner-3 muss ein numerisches Datenfeld sein, das mit PIC S9(8) SYNC und USAGE COMP, USAGE COMP-5 oder USAGE BINARY beschrieben ist.
2. bezeichner-2 darf kein Funktionsbezeichner sein.
3. bezeichner-3 muss ein numerisches Datenfeld sein, das mit PIC S9(8) SYNC und USAGE COMP, USAGE COMP-5 oder USAGE BINARY beschrieben ist.
4. bezeichner-2 darf kein Funktionsbezeichner sein.

Allgemeine Regeln

1. literal-1 bzw. bezeichner-1 enthält das auszuführende BS2000-Kommando.
   Der Schrägstrich (/) vor dem Kommando kann angegeben werden. Kleinbuchstaben werden *nicht* in Großbuchstaben umgesetzt.
2. Mit bezeichner-2 wird ein Bereich zur Aufnahme von Systemausgaben spezifiziert.Dieser Antwortbereich sollte groß genug angelegt werden, um die vollständige Ausgabe des angegebenen BS2000-Kommandos aufnehmen zu können.
   Ist bezeichner-2 variabel lang und enthält das DEPENDING ON-Feld, dann wird die maximale Länge von bezeichner-2 verwendet.
3. Ist bezeichner-2 nicht angegeben oder ist seine aktuelle Länge 0, dann erfolgt die Ausgabe nach SYSOUT.
4. Tritt bei der Ausführung des Kommandos ein Fehler auf, enthält der Antwortbereich, falls er groß genug angelegt wurde, den Fehlermeldungstext des Systems.
5. Mit bezeichner-3 kann ein Statusfeld angegeben werden, in dem bestimmte Werte das Resultat der Kommandoausführung anzeigen. Folgende Werte können auftreten:

00	Erfolgreiche Ausführung des Kommandos
04	Das Kommando wurde ausgeführt, aber auf Grund des zu klein angelegten Antwortbereichs konnten ein oder mehrere Sätze nicht eingetragen werden.
30	Fehler bei der Ausführung des Kommandos; keine weitere Information verfügbar
34	Das Kommando darf nicht in einem Programm angegeben werden.
40	Die aktuelle Länge von bezeichner-1 ist ungültig (<= 0 oder > 32767 Byte).
41	Die aktuelle Länge von bezeichner-2 ist ungültig (< 0).
90	Es ist nicht genügend Arbeitsspeicher zur Ausführung des Kommandos verfügbar; Abhilfe: kleinere Bereiche (bezeichner-1 oder bezeichner-2) angeben

6. Ist während der Ausführung der CALL-Anweisung ein Fehler aufgetreten, wird eine der folgenden Aktionen durchgeführt:
   a. Ist ON EXCEPTION angegeben, wird unbedingte-anweisung-1 ausgeführt. Nach Beendigung von unbedingte-anweisung-1 geht die Steuerung zum Ende der CALL-Anweisung über.
   b. Ist ON EXCEPTION nicht angegeben, wird der Programmablauf ohne Fehlermeldung fortgesetzt.
7. Wurde die CALL-Anweisung erfolgreich ausgeführt, wird eine der folgenden Aktionen durchgeführt:
   a. Ist NOT ON EXCEPTION angegeben, wird unbedingte-anweisung-2 ausgeführt.
   b. Ist NOT ON EXCEPTION nicht angegeben, wird zum Ende der CALL-Anweisung verzweigt.

> **i** Die Ausgabe des Systems besteht aus variablen Sätzen mit Längenfeldern und Sonderzeichen zur Vorschubsteuerung o.ä.; ein Satz entspricht i.d.R. mehreren Zeilen am Bildschirm. In den Antwortbereich werden immer nur vollständige Sätze eingetragen; der nicht belegte Rest des Antwortbereichs wird gelöscht. Zu Strukturen und Weiterverarbeitung des Antwortbereichs siehe Benutzerhandbuch „Makroaufrufe an den Ablaufteil - CMD-Makro“ [12].

Beispiel 8-24

für die Anwendung der CALL-Anweisung im Format CALL bezeichner-1

Hauptprogramm:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01   CALL-OPERAND PIC X(8) VALUE SPACE.
01   TABLE-FUNCTION.
     02  FUNCTION-1 PIC X(8) VALUE "ADDREC".
     02  FILLER     PIC X(72) VALUE SPACE.
     02  FUNCTION-2 PIC X(8) VALUE "DELREC".
     02  FILLER     PIC X(72) VALUE SPACE.
     02  FUNCTION-3 PIC X(8) VALUE "CHGREC".
     02  FILLER     PIC X(72) VALUE SPACE.
01   RECORD-NUMBER PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
HAUPT.
     PERFORM UNTIL CALL-OPERAND = FUNCTION-1 OR FUNCTION-2
                                             OR FUNCTION-3
       DISPLAY "Bitte gewaehlte Funktion eingeben" UPON T
       DISPLAY TABLE-FUNCTION UPON T
       ACCEPT CALL-OPERAND FROM T
     END-PERFORM
     PERFORM UNTIL RECORD-NUMBER NUMERIC
       DISPLAY "Bitte Satznummer eingeben"
                "(numerisch, achtstellig)" UPON T
       ACCEPT RECORD-NUMBER FROM T
     END-PERFORM
     CALL CALL-OPERAND USING RECORD-NUMBER
     END-CALL
     STOP RUN.
```

Unterprogramm ADDREC:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDREC.
ENVIRONMENT DIVISION.
DATA DIVISION.
...
LINKAGE SECTION.
01 RECORD-NUMBER PIC 9(8).
PROCEDURE DIVISION USING RECORD-NUMBER.
    ...
    EXIT PROGRAM.
```

Unterprogramm DELREC:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DELREC.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
...
LINKAGE SECTION.
01  RECORD-NUMBER PIC 9(8).
PROCEDURE DIVISION USING RECORD-NUMBER.
    ...
    EXIT PROGRAM.
```

Unterprogramm CHGREC:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CHGREC.
ENVIRONMENT DIVISION.
DATA DIVISION.
...
LINKAGE SECTION.
01  RECORD-NUMBER PIC 9(8).
PROCEDURE DIVISION USING RECORD-NUMBER.
    ...
    EXIT PROGRAM.
```

Beispiel 8-25

für die Verwendung von CALL ... USING BY VALUE

Hauptprogramm:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SPECIAL-NAMES.
         TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  C  PIC  9(4) USAGE COMP-5 VALUE 1.
01  D  PIC  9(9) USAGE COMP-5 VALUE 1.
01  E  PIC  S9(4) USAGE COMP-5 VALUE -1.
01  F  PIC  S9(9) USAGE COMP-5 VALUE -1.
01  RTC  PIC S9(10) SIGN IS LEADING SEPARATE.
PROCEDURE DIVISION.
1ST SECTION.
1.
CALL "C1" USING BY VALUE C, D.
    MOVE RETURN-CODE TO RTC.
    DISPLAY "RETURN-CODE = " RTC UPON T.
    CALL "D1" USING BY VALUE E, F.
    MOVE RETURN-CODE TO RTC.
    DISPLAY "RETURN-CODE = " RTC UPON T.
    MOVE 0 TO RETURN-CODE.
```

Unterprogramm C1:

Unterprogramm D1:

```
long C1(unsigned short
c,
          unsigned
long d)
{
long ret_val;
if (c && d)
   ret_val = 1;
else
   ret_val = -1;
return ret_val;
}
```

```
long D1(signed short c, signed
long d)
{
long ret_val;
if (c && d)
   ret_val = 1;
else
   ret_val = -1;
return ret_val;
}
```

Beispiel 8-26

Dateinamen-Zuweisung mit dem SET-FILE-LINK-Kommando im Dialog und Status-Abfrage:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CMD.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LINKFILE.                                                    (1)
  02  FILLER  PIC X(30) VALUE "ADD-FILE-LINK LINK=XXX,F-NAME=".
  02  FILNAM  PIC X(54).
01  RTC       PIC S9(8) SYNC BINARY.                             (2)
01 RTC-ED     PIC Z(6)99.
01 TFT-CMD    PIC X(40) VALUE "SHOW-FILE-LINK".
01 RESP       PIC X(2000) VALUE ALL SPACE.                       (3)
PROCEDURE DIVISION.
MAIN SECTION.
PARA.
    DISPLAY "BITTE DATEINAMEN EINGEBEN X(54)" UPON T.
    ACCEPT FILNAM FROM T.
    CALL UPON SYSTEM USING LINKFILE
                           STATUS RTC
      ON EXCEPTION

         MOVE RTC TO RTC-ED
         DISPLAY "FEHLER BEIM KOMMANDOAUFRUF, STATUS= "
                    RTC-ED UPON T
    END-CALL
    CALL UPON SYSTEM USING TFT-CMD
                           RESP
    END-CALL.
    DISPLAY "ANTWORTBEREICH RESP" RESP UPON T.
FIN.
    STOP RUN.
```

(1) Beschreibung von bezeichner-1 mit Unterstrukturen für BS2000-Systemkommando und Dateiname.

8.10.6 CANCEL-Anweisung

Funktion

Die CANCEL-Anweisung bewirkt, dass sich das angegebene Programm beim nächsten Aufruf im Initialzustand befindet.

Format

```
CANCEL {bezeichner-1 | literal-1} ...
```

Syntaxregeln

1. literal-1 muss ein alphanumerisches Literal und ein gültiger Programmname sein. Es darf jedoch keine figurative Konstante sein.

   Ist literal-1 der Name eines getrennt übersetzten Programms bzw. des äußersten Programms eines geschachtelten Programms, muss es mit einem alphabetischen Zeichen beginnen und darf nur Großbuchstaben und Ziffern enthalten. Die Namenslänge ist abhängig vom Modulformat.
   Ist literal-1 der Programmname für ein inneres Programm eines geschachtelten Programms, muss es mit einem alphabetischen Zeichen beginnen, darf Groß- und Kleinbuchstaben sowie Ziffern enthalten und darf eine Länge von maximal 30 Zeichen haben.
2. bezeichner-1 muss als alphanumerisches Datenfeld definiert sein, so dass sein Wert ein gültiger Programmname, wie in 1. beschrieben, sein kann.

Allgemeine Regeln

1. literal-1 bzw. der Inhalt von bezeichner-1 geben das Programm an, das den Initialzustand erhält.
2. Eine erfolgreich ausgeführte CANCEL-Anweisung bewirkt, dass die Dateien im angesprochenen Programm geschlossen werden. Wird das Programm im Anschluss an eine erfolgreich ausgeführte CANCEL-Anweisung in derselben Ablaufeinheit bzw. in einem geschachtelten Programm aufgerufen, befindet es sich im Initialzustand.

   Der durch literal-1 oder den Inhalt von bezeichner-1 benannte Programmname muss in der Ablaufeinheit bzw. im geschachtelten Programm eindeutig sein, ausgenommen es handelt sich um einen Programmnamen, der unter bestimmten Bedingungen (siehe „CALL-Anweisung", allgemeine Regel 4) mehrmals verwendet werden darf.Der Programmname darf nicht mit den ersten 7 Zeichen des Programmnamens im PROGRAM-ID-Paragrafen des Programms, das diese CANCEL-Anweisung enthält, übereinstimmen.
3. Aufgerufene Programme dürfen CANCEL-Anweisungen enthalten. Ein aufgerufenes Programm darf jedoch keine CANCEL-Anweisung ausführen, die direkt oder indirekt das aufrufende Programm referenziert.
4. Die logische Verbindung zu einem Programm, das mittels einer CANCEL-Anweisung in den Initialzustand gesetzt wurde, wird nur wiederhergestellt, wenn es nachfolgend mit CALL aufgerufen wird.
5. Eine CANCEL-Anweisung bleibt ohne Wirkung, wenn einer der folgenden Fälle vorliegt:
   - Das betreffende Programm befindet sich noch im Initialzustand, weil es in der aktiven Ablaufeinheit bzw. im geschachtelten Programm noch nicht aufgerufen wurde.
   - Das betreffende Programm wurde schon mittels einer CANCEL-Anweisung in den Initialzustand versetzt.
   - Das betreffende Programm bzw. (bei geschachtelten Programmen) ein übergeordnetes Programm besitzt das Attribut INITIAL.

   In diesen Fällen geht die Steuerung über zur nächsten ausführbaren Anweisung nach der CANCEL-Anweisung.
6. Während der Ausführung einer CANCEL-Anweisung wird für jede zugeordnete, geöffnete interne Datei eine implizite CLOSE-Anweisung (ohne jedwede optionale Angabe) durchgeführt. Für diese Dateien angegebene USE-Prozeduren werden nicht ausgeführt.

7. Eine CANCEL-Anweisung darf nur solche Programme ansprechen, deren Aufruf innerhalb der Aufrufhierarchie zulässig ist.
8. Wird eine explizite oder implizite CANCEL-Anweisung ausgeführt, so werden auch alle inneren Programme des von der CANCEL-Anweisung angesprochenen Programms storniert.

Beispiel 8-27

Hauptprogramm:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FEHLER-CODE PIC 9.
    88  O-K VALUE 0.
...
PROCEDURE DIVISION.
P1 SECTION.
HAUPT.
    PERFORM WITH TEST AFTER UNTIL O-K
      CALL "UPROG1" USING FEHLER-CODE
      IF NOT O-K
      THEN
        CANCEL "UPROG1"
      END-IF
    END-PERFORM
    STOP RUN.
```

Unterprogramm:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPROG1.
ENVIRONMENT DIVISION.
DATA DIVISION.
   ...
LINKAGE SECTION.
01  FEHLER-CODE  PIC 9.
    88  O-K  VALUE 0.
PROCEDURE DIVISION USING FEHLER-CODE.
    ...
    IF INTERN-ERROR
    THEN
      CONTINUE
    ELSE
      SET O-K TO TRUE
    END-IF
    EXIT PROGRAM.
```

UPROG1 wird solange aufgerufen, bis der Wert 0 zurückgeliefert wird. Solange ein Wert ungleich 0 zurückgeliefert wird, wird UPROG1 in den Initialzustand zurückgesetzt und die Schleife fortgesetzt.

8.10.7 CLOSE-Anweisung

Funktion

Für **sequenzielle Dateiorganisation:** Die CLOSE-Anweisung beendet die Verarbeitung von Ein-/Ausgabe-Spulen, Einheiten (UNIT) und Dateien, wobei wahlweise Rückspul- oder Sperrfunktionen durchgeführt werden.

Für **relative und indizierte Dateiorganisation:** Die CLOSE-Anweisung beendet die Verarbeitung von Dateien, wobei wahlweise Sperrfunktionen durchgeführt werden.

Format 1 für sequenzielle Dateiorganisation

```
CLOSE {dateiname [ [{ REEL | UNIT }] [FOR REMOVAL]
                  | WITH { NO REWIND | LOCK }
                 ]
      }...
```

Format 2 für relative und indizierte Dateiorganisation

```
CLOSE {dateiname-1 [WITH LOCK]}...
```

Syntaxregeln

1. Die in der CLOSE-Anweisung angesprochenen Dateien können verschiedene Organisation oder Zugriffsart haben.
2. Die Ausdrücke REEL und UNIT sind gleichbedeutend und innerhalb einer CLOSE-Anweisung voll austauschbar. Die Behandlung von sequenziellen Plattenspeicherdateien ist logisch gleichzusetzen mit der Behandlung von Dateien auf Band oder ähnlichen sequenziellen Medien.
3. Bei zeilensequenziellen Dateien ist von den zusätzlichen Angaben nur WITH LOCK zulässig.

Allgemeine Regeln

1. Eine CLOSE-Anweisung darf nur für eine eröffnete Datei ausgeführt werden.
2. Die Angaben in der CLOSE-Anweisung bedeuten:

REEL	Die Verarbeitung der aktuellen Magnetbandspule soll abgeschlossen werden.
UNIT	Die Verarbeitung der aktuellen Plattenspeichereinheit soll abgeschlossen werden.
NO REWIND	Nach Abschluss der Verarbeitung einer Banddatei soll diese nicht auf den Spulenanfang zurückpositioniert werden.
LOCK	Die Datei kann im selben Programmlauf nicht wieder eröffnet werden.
REMOVAL	Die aktuelle Spule einer Magnetbanddatei soll entladen werden.

3. Um die Wirkung von verschiedenen CLOSE-Anweisungen, angewandt auf verschie
   dene Speichermedien, zu zeigen, werden alle Ein-/Ausgabe-Dateien in die folgenden Kategorien eingeteilt:
   a. **UNIT-RECORD-Datenträgerdatei (Einheitsdatensatzdatei):**
      Eine einem Ein- oder Ausgabe-Gerät zugewiesene Datei, für die die Begriffe Rückspulen, UNIT und Spule keine Bedeutung haben.
   b. **Sequenzielle Eindatenträgerdatei:**
      Eine sequenzielle Datei, die vollkommen auf einem Datenträger enthalten ist. Auf diesem Datenträger können sich mehrere Dateien befinden.
   c. **Sequenzielle Mehrdatenträgerdatei:**
      Eine sequenzielle Datei, die sich über mehr als einen Datenträger erstreckt.

4. Die Ergebnisse der Ausführung der verschiedenen CLOSE-Anweisungen sind - bezogen auf die jeweilige Dateikategorie - in der folgenden Tabelle zusammengefasst:

CLOSE-Anweisung	**Kategorie**			
	Dateien-sequenziell organisiert			relativ/indiziert
	Einheitsdaten-satz-Datei	Eindatenträ-ger-Datei	Mehrdatenträ-ger-Datei	Ein-/Mehrdaten-träger-Datei
CLOSE	C	C, G	C, G, A	C
CLOSE WITH LOCK	C, E	C,G,E	C, G, E, A	C,E
CLOSE WITH NO REWIND	C, H	C, B	C, B, A	–
CLOSE FOR REMOVAL	C, H	G, D	F, G, D	–
CLOSE REEL/UNIT	J	F, G	F, G	–
CLOSE REEL /UNIT FOR REMOVAL	J	F, G, D	F, G, D	–

Tabelle 26: Zusammenhang zwischen Dateien und Angaben der CLOSE-Anweisung

Die in der Tabelle verwendeten Symbole werden nachfolgend definiert. Falls die Defi-nition eines Symbols für eine Ein- oder Ausgabe-Datei unterschiedlich ist, werden verschiedene Symbole verwendet; andernfalls gelten die Angaben für Dateien mit Eröffnungsart INPUT, OUTPUT und I-O.

A **Vorhergehende Datenträger nicht berührt**

Alle Datenträger der Datei, die vor der aktuellen Datenträgereinheit verarbeitet wur-den, wurden entsprechend den Standard-Datenträger-Wechselroutinen verarbeitet; ausgenommen sind die durch eine vorhergehende CLOSE-Anweisung mit Angabe REEL/UNIT angesprochenen Datenträger.

B **Kein Rückspulen der aktuellen Spule**

Der aktuelle Datenträger wird auf das logische Dateiende auf dem Datenträger positioniert.

C **Standard-Dateiabschluss**

Für Dateien, die mit INPUT- oder I-O-Angabe eröffnet wurden:

Falls die Datei auf Dateiende positioniert ist und eine LABEL RECORDS-Klausel angegeben wurde, wird (falls eine USE-Prozedur vorhanden ist) die Standard- und Benutzer-Dateiende-Routine für Kennsatzbehandlung durchlaufen. Die Reihenfolge der Ausführung dieser beiden Routinen wird durch die USE-Prozedur festgelegt. Danach werden die Standard-Abschlussroutinen des Systems durchlaufen.

Falls die Datei auf Dateiende positioniert ist, jedoch keine LABEL RECORDS-Klausel angegeben war, werden nur die Standard-Abschlussroutinen des Systems durchlaufen.

Falls die Datei nicht auf Dateiende positioniert ist, werden lediglich die Standard--Abschlussroutinen des Systems durchlaufen. Selbst bei Vorhandensein von USE-Prozeduren wird in diesem Fall keine Kennsatzbehandlung durchgeführt. (Das Ende einer Eingabe- oder Ein-/Ausgabe-Datei ist erreicht, wenn die unbedingte Anweisung der AT END-Angabe der READ-Anweisung, falls angegeben, durchlaufen wurde und keine CLOSE-Anweisung durchgeführt wurde.)

Für Dateien, die mit OUTPUT-Angabe eröffnet wurden:

Falls Kennsatzbehandlung für diese Datei vereinbart wurde, werden die Standard-Endekennsatz-Routinen und die Benutzer-Endekennsatz-Routinen (vorausgesetzt, solche wurden durch eine USE-Prozedur angegeben) durchlaufen. Die Reihenfolge der Ausführung dieser Vereinbarungen wird durch die USE-Anweisung festgelegt. Danach werden die Standard-Abschlussprozeduren des Systems durchlaufen.

Falls keine Kennsätze für die Datei vereinbart wurden, werden nur die Standard-Abschlussprozeduren des Systems durchgeführt.

D **Entladen der aktuellen Spule**

Bei Verwendung der REMOVAL-Angabe wird die aktuelle Spule einer Magnetband-datei entladen. Die weitere Verarbeitung der Datei kann nur mit der richtigen Folgespule fortgesetzt werden. Nach Ausführung einer CLOSE-Anweisung ohne die Angabe REEL/UNIT und einer OPEN-Anweisung für diese Datei ist eine erneute Verarbeitung dieser Spule möglich.

E **Standard-Datei-Sperrung**

Bei Verwendung der LOCK-Angabe kann die Datei im selben Programmlauf nicht wieder eröffnet werden.

F **Standard-Datenträgerabschluss**

Für Dateien, die mit INPUT- oder I-O-Angabe eröffnet wurden, werden die folgen-den Operationen durchgeführt:

- Der aktuelle Datenträger ist der einzige oder der letzte Datenträger für die Datei:

  Plattenspeicherdateien

  Die Bearbeitung des aktuellen Datenblocks wird beendet. Die nächste READ-Anweisung stellt den ersten Datensatz des nachfolgenden Datenblocks zur Verfügung.

  Banddateien

  Die Bearbeitung der Spule wird beendet. Die nächste READ-Anweisung führt zu einer Ende-Bedingung.
- Es existieren weitere Datenträger für die Datei:
  a. Datenträgerwechsel.
  b. USE-Prozeduren zur Anfangskennsatz-Verarbeitung (Standard- und Benutzer-Kennsätze), falls durch eine USE-Anweisung vereinbart. Die Reihenfolge der Ausführung dieser Prozeduren wird durch die USE-Anweisung festgelegt.
  c. Der nächste Datensatz auf dem neuen Datenträger wird für den folgenden Lesevorgang zur Verfügung gestellt.

Für Dateien, die im Ausgabemodus eröffnet wurden, werden die folgenden Operationen durchgeführt:

- *Plattenspeicherdateien*

  Die Bearbeitung des aktuellen Datenblocks wird beendet. Die nächste WRITE-Anweisung erstellt einen neuen Datenblock.
- *Banddateien*
  a. Prozedurvereinbarungs-Prozeduren zur Endekennsatz-Verarbeitung (Standard- und Benutzerkennsätze), falls durch eine USE-Anweisung vereinbart. Dabei wird die Reihenfolge der Ausführung dieser Prozeduren durch die USE-Anweisung festgelegt.
  b. Datenträgerwechsel.
  c. Prozedurvereinbarungs-Prozeduren zur Anfangskennsatz-Verarbeitung (Standard- und Benutzer-Kennsätze), falls durch eine USE-Anweisung vereinbart. Dabei wird die Reihenfolge der Ausführung dieser Prozeduren durch die USE-Anweisung festgelegt.

G **Zurückspulen**

Der aktuelle Datenträger wird auf Anfang positioniert (bei Magnetbanddateien ist dies der Spulenanfang; bei Plattenspeicherdateien ist dies der Anfang der betreffenden Datei).

H Der Ein-/Ausgabe-Zustand 07 wird gesetzt.

Die optionalen Angaben werden ignoriert.

J Der Ein-/Ausgabe-Zustand 07 wird gesetzt.

Die CLOSE-Anweisung wird ignoriert.

5. Nach Ausführung einer CLOSE-Anweisung wird der Inhalt des in der FILE STATUS-Klausel angegebenen Datenfeldes (falls eine solche Klausel vorhanden ist) aktualisiert (siehe auch „FILE STATUS-Klausel").
6. Nach erfolgreichem Abschluss einer CLOSE-Anweisung steht der der Datei zugeordnete Satzbereich nicht mehr zur Verfügung. Die Verfügbarkeit ist nach erfolglosem Ablauf einer CLOSE-Anweisung nicht vorhersagbar.
7. Ist eine optionale Eingabedatei nicht vorhanden, wird keine Datei-Endeverarbeitung und keine Spulen-/Bandverarbeitung durchgeführt.
8. Sind mehrere Dateinamen angegeben, so ist die Wirkung die gleiche, wie wenn für jeden Dateinamen eine eigene CLOSE-Anweisung gegeben worden wäre.
9. Alle Dateien, die nach Abschluss eines Prozesses noch geöffnet sind, werden geschlossen.

8.10.8 COMPUTE-Anweisung

Funktion

Mit der COMPUTE-Anweisung wird einem Datenfeld der Wert eines Datenfeldes, Literals oder eines arithmetischen Ausdruckes zugewiesen.

Format

```
COMPUTE {bezeichner-1 [ROUNDED]}... = { bezeichner-2
                                      | literal-1
                                      | arithmetischer-ausdruck }

       [ON SIZE ERROR unbedingte-anweisung-1]

       [NOT ON SIZE ERROR unbedingte-anweisung-2]

       [END-COMPUTE]
```

Syntaxregeln

1. bezeichner-1... muss sich auf ein numerisches Datenelement oder numerisch-druckaufbereitetes Datenelement beziehen.
2. bezeichner-2 muss sich auf ein numerisches Datenelement beziehen.
3. Die Angabe für arithmetischer-ausdruck in der COMPUTE-Anweisung lässt jede sinnvolle Kombination von Bezeichnern (die die allgemeinen Regeln für Datennamen in den Grundrechnungsarten erfüllen müssen), Literalen und arithmetischen Operanden zu, die, falls notwendig, auch geklammert sein können (siehe auch unter „Arithmetische Ausdrücke").
4. Bei Angabe von bezeichner-2 oder literal-1 wird der Wert von bezeichner-1 gleich dem Wert von bezeichner-2 oder literal-1 gesetzt.
5. Wird ein arithmetischer Ausdruck verwendet, so wird der Wert des arithmetischen Ausdrucks errechnet und dann dieser Wert als neuer Wert von bezeichner-1... abgespeichert.
6. Die COMPUTE-Anweisung erlaubt dem Benutzer, arithmetische Operationen zu kombinieren, ohne die Einschränkungen bezüglich der gemeinsamen Stellenanzahl der Operanden oder Datenempfangsfelder, die für die Anweisungen ADD und SUBTRACT gelten, zu beachten.
7. In einer COMPUTE-Anweisung können bis zu 50 Datennamen angegeben werden.

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die ROUNDED-Angabe und (NOT) ON SIZE ERROR-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-28

Anweisung	Rechnung
COMPUTE A = (B + C) / D - E.	Der Wert des Ausdrucks (B + C) / D - E wird A zugewiesen. Bei der Berechnung des Wertes gelten die Präzedenzregeln zur Auflösung von Ausdrücken.
COMPUTE A = 2.	Der Wert 2 wird A zugewiesen

8.10.9 CONTINUE-Anweisung

Die CONTINUE-Anweisung ist eine nicht ausführbare Anweisung. Die Verarbeitung wird bei der nächsten ausführbaren Anweisung fortgesetzt.

Format

```
CONTINUE
```

Syntaxregel

1. Die CONTINUE-Anweisung kann überall da verwendet werden, wo eine bedingte oder unbedingte Anweisung verwendet werden kann.

Allgemeine Regel

1. Die CONTINUE-Anweisung hat keine Auswirkung auf die Programmausführung.

Beispiel 8-29

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CONT1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 N PIC 9.
77 K PIC 9(3).
77 Z PIC 9(6) VALUE ALL ZERO.
77 E PIC 9(3).
PROCEDURE DIVISION.
PROC SECTION.
EINGABE.
    DISPLAY "Obere Grenze N eingeben" UPON T.
    ACCEPT N FROM T.
    IF N NUMERIC
    THEN
      CONTINUE
    ELSE
      DISPLAY "Falsche Eingabe" UPON T
      PERFORM EINGABE
    END-IF.
BERECHNUNG.
    PERFORM WITH TEST BEFORE VARYING K FROM 1 BY 1 UNTIL K > N
      COMPUTE E = K ** 3
      ADD E TO Z
    END-PERFORM
    DISPLAY "Ergebnis = " Z UPON T.
ENDE.
    STOP RUN.
```

CONTINUE bewirkt, dass die IF-Anweisung syntaktisch richtig ist, obwohl der THEN-Zweig keine ausführbare Anweisung enthält.

Beispiel 8-30

```
READ INPUT-DATEI AT END CONTINUE.
```

Um einem Programmabruch bei Dateiende vorzubeugen, ist AT END angegeben; mit CONTINUE wird die syntaktisch erforderliche unbedingte Anweisung gegeben, auch wenn an dieser Stelle nichts ausgeführt werden soll.

8.10.10 DELETE-Anweisung

Gilt für relative **und indizierte Dateiorganisation**

Funktion

Durch eine DELETE-Anweisung wird ein Datensatz einer Plattenspeicherdatei logisch gelöscht.

Format

```
DELETE dateiname RECORD
  [INVALID KEY unbedingte-anweisung-1]
  [NOT INVALID KEY unbedingte-anweisung-2]
  [END-DELETE]
```

Syntaxregeln

1. Für eine Datei, die sich im sequenziellen Zugriffsmodus befindet, darf der Zusatz INVALID KEY in der DELETE-Anweisung nicht angegeben werden.
2. Für eine Datei, die sich nicht im sequenziellen Zugriffsmodus befindet, ist die Angabe des Zusatzes INVALID KEY obligatorisch, falls keine zutreffende USE-Prozedur vorhanden ist.

Allgemeine Regeln

1. Die in der DELETE-Anweisung angesprochene Datei muss sich während der Ausführung dieser Anweisung im Eröffnungsmodus I-O befinden (siehe auch „OPEN-Anweisung").
2. Nach erfolgreicher Ausführung der DELETE-Anweisung ist der in Frage kommende Datensatz in der Datei gelöscht.
3. Die Ausführung einer DELETE-Anweisung beeinflusst nicht den Inhalt des zu der Datei gehörigen Datensatzbereichs oder den Inhalt des Datenelements, das in der DEPENDING ON-Angabe der RECORD-Klausel mit datenname angegeben wurde.
4. Bei einer Datei, die sich im sequenziellen Zugriffsmodus befindet, muss die der DELETE-Anweisung vorausgegangene Ein-/Ausgabe-Anweisung eine erfolgreich abgeschlossene READ-Anweisung gewesen sein. Zwischen dem Lesen und Löschen darf der RELATIVE KEY (bei **relativer Dateiorganisation**) bzw. der RECORD KEY(bei **indizierter Dateiorganisation**) nicht verändert werden. Die DELETE-Anweisung löscht in diesem Fall logisch den durch die READ-Anweisung gelesenen Datensatz aus der Datei.
5. Nach Ausführung der DELETE-Anweisung wird der Inhalt des für die Datei in der FILE STATUS-Klausel angegebenen Datenfeldes aktualisiert (siehe „FILE STATUS-Klausel").
6. Die Fortsetzung des Ablaufs der DELETE-Anweisung hängt davon ab, ob INVALIDKEY oder NOT INVALID KEY angegeben ist (siehe „Schlüsselfehler-Bedingung").

8.10.11 DISPLAY-Anweisung

Funktion

Format 1	wird zur Ausgabe von kleineren Datenmengen verwendet.
Format 2	dient dem Zugriff auf eine POSIX-Kommandozeile
Format 3 & 4	dienen dem Zugriff auf eine BS2000- oder POSIX-Umgebungsvariable

Format 1

```
DISPLAY {literal-1 | bezeichner-1}... [UPON merkname] [WITH NO ADVANCING]
```

Syntaxregeln

1. Falls literal-1 numerisch ist, muss es eine vorzeichenlose Ganzzahl sein.

   bezeichner-1 darf nicht von der Klasse national, zeiger oder objekt sein.
2. merkname muss im SPECIAL-NAMES-Paragrafen angegeben werden und mit einem der folgenden Herstellernamen verknüpft sein:
   CONSOLE, PRINTER, PRINTER01 - PRINTER99, SYSOPT, TERMINAL, jobvariablenname (Jobvariable BS2000).
3. Mit der Angabe des Merknamens für SYSOPT, TERMINAL, CONSOLE, PRINTER und PRINTER01-PRINTER99 wird eine Systemdatei spezifiziert, in die die Daten geschrieben werden sollen.

   SYSOPT bezeichnet die Systemdatei gleichen Namens.

   TERMINAL bezeichnet die Systemdatei SYSOUT.

   CONSOLE bezeichnet den Bedienplatz des Systems.

   PRINTER bezeichnet die Systemdatei SYSLST, PRINTER01-PRINTER99 bezeichnen die Systemdateien SYSLST01-SYSLST99.
4. Falls die UPON-Angabe fehlt, werden die Daten standardmäßig in die logische Ausgabedatei SYSLST ausgegeben. Mit einer entsprechenden Compiler-Steueranweisung können die Daten auch in die logische Ausgabedatei SYSOUT ausgegeben werden (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).
5. bezeichner-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.

Allgemeine Regeln

1. Mit literal-1 oder bezeichner-1 werden die Operanden angegeben, die in der angeführten Reihenfolge ausgegeben werden sollen. Der Inhalt des durch bezeichner angegebenen Datenfeldes wird, falls notwendig, nach den folgenden Regeln in externe Formate umgewandelt:
   - Intern dezimale und binäre Felder werden in extern dezimale Datenfelder umgewandelt.
   - Interne Gleitpunktdatenfelder werden in externe Gleitpunktdatenfelder umgewandelt.

   Alle anderen Datenfelder erfordern keine Umwandlung.

   Falls einer der Operanden eine figurative Konstante ist (außer ALL literal), wird diese mit der Länge 1 ausgegeben.

   Falls einer der Operanden die figurative Konstante ALL literal ist, wird das Literal einmal ausgegeben.
2. Für jedes Gerät wird eine maximale logische Satzgröße angenommen. Diese Größen sind in Tabelle 27 aufgeführt.

Gerät	Maximale Datensatzgröße
CONSOLE	180 Zeichen

PRINTERPRINTER01-PRINTER99	132 Zeichen + 1 Byte Steuerinformation
SYSOPT	80 Zeichen: 72 Datenbytes; die Bytes 73-80 enthalten die ersten 8 Bytes des PROGRAM-ID-Namens.
TERMINAL	8192 Zeichen

Tabelle 27: Maximale logische Datensatzgröße für die DISPLAY-Anweisung

3. Wenn eine DISPLAY-Anweisung mehr als einen Operanden enthält, werden die Inhalte der angegebenen Datenbereiche und Literale nebeneinander von links nach rechts ausgegeben.
   Null-längige Datenfelder werden nicht ausgegeben. Sind alle Operanden null-längige Datenfelder, so hat die DISPLAY-Anweisung keine Auswirkung.
4. Die NO ADVANCING-Angabe wird vom Compiler als Kommentar behandelt.
5. Bei Druckerausgabe wird ein Zeilenvorschub durch die folgenden Angaben erzeugt: DISPLAY, WRITE und WRITE AFTER ADVANCING. Durch eine WRITE-Anweisung ohne ADVANCING-Angabe oder mit BEFORE ADVANCING-Angabe wird ein Zeilenvorschub nach dem Drucken bewirkt. Daher kann sich durch gemischte Anwendungen von DISPLAY- und WRITE-Anweisungen auf das gleiche Gerät innerhalb ein und desselben Programmes ein Überdrucken von Zeilen ergeben. Auf Laserdrucker ist ein Überdrucken nicht möglich.
6. Die maximale Datensatzgröße bei Jobvariablen beträgt 256 Zeichen.
   Falls die Gesamtzeichenanzahl der Operanden die maximale Datensatzgröße überschreitet, wird der Datensatz auf die maximale Länge abgeschnitten.
7. Wenn eine Jobvariable als Monitor-Jobvariable (überwachende Jobvariable) MONJV verwendet wird, dann schützt das System die ersten 128 Bytes (Systemteil) dieser Jobvariablen gegen Schreibzugriffe. Es wird daher nur der Teil eines Datensatzes, der mit der Position 129 beginnt, ab Position 129 der Monitor-Jobvariablen geschrieben.
   Ansonsten gilt für überwachende Jobvariablen die Allgemeine Regel 4.

Weitere Angaben siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1].

Beispiel 8-31

```
SPECIAL-NAMES.
    TERMINAL IS SONDER-AUSGABE.
     ...
PROCEDURE DIVISION.
     ...
    DISPLAY AUSGABE-TEXT UPON SONDER-AUSGABE.
```

Hierbei ist dem Merknamen SONDER-AUSGABE der Herstellername TERMINAL im SPECIAL-NAMES-Paragrafen zugewiesen. Durch die DISPLAY-Anweisung wird der Inhalt von AUSGABE-TEXT auf SYSOUT ausgegeben.

Beispiel 8-32

```
DISPLAY "Hello universe!".
```

Da die UPON-Angabe fehlt, wird das Literal "Hello universe!" standardmäßig in die Ausgabedatei SYSLST ausgegeben. Wurde die Compiler-Steueranweisung
COMOPT REDIRECT-ACCEPT-DISPLAY=YES
(in SDF: ACCEPT-DISPLAY-ASSGN=*TERMINAL) angegeben, wird das Literal in die Ausgabedatei SYSOUT ausgegeben (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

Die folgenden drei Formate der DISPLAY-Anweisung sind Erweiterungen aus dem X/OPEN Portability Guide. Sie erlauben den Zugriff auf Umgebungsvariablen und Kommandozeilen. Der Zugriff auf eine Kommandozeile ist nur sinnvoll, wenn das Zielprogramm im POSIX-Subsystem abläuft, das ab BS2000/OSD 2.0 zur Verfügung steht. Der Ablauf des COBOL2000-Compilers und der von ihm erzeugten Programme unter POSIX ist im Handbuch „COBOL2000 Benutzerhandbuch“ [1] beschrieben.

Format 2

setzt die Nummer des Arguments in der Kommandozeile, auf das anschließend mit einer ACCEPT-Anweisung zugegriffen wird.

```
DISPLAY {bezeichner-3 | ganzzahl-1} UPON merkname-3 [END-DISPLAY]
```

Syntaxregeln

1. bezeichner-3 muss sich auf ein Datenelement beziehen, das als Ganzzahl ohne Vorzeichen beschrieben ist.
2. ganzzahl-1 darf kein Vorzeichen besitzen.
3. merkname-3 muss im SPECIAL-NAMES-Paragrafen mit dem Herstellernamen ARGUMENT-NUMBER verknüpft sein.

Format 3

setzt den Namen der Umgebungsvariablen, auf die anschließend mit einer ACCEPT- oder DISPLAY-Anweisung zugegriffen wird.

```
DISPLAY {bezeichner-4 | literal-1} UPON merkname-5 [END-DISPLAY]
```

Syntaxregeln

1. bezeichner-4 muss sich auf ein alphanumerisches Datenelement beziehen, das ohne ANY LENGTH-Klausel definiert ist.
2. literal-1 muss ein alphanumerisches Literal sein.
3. merkname-5 muss im SPECIAL-NAMES-Paragrafen mit dem Herstellernamen ENVIRONMENT-NAME verknüpft sein.

Format 4

schreibt in die Umgebungsvariable, die zuvor mit einer DISPLAY-Anweisung vom Format 3 benannt wurde.

```
DISPLAY {bezeichner-2 | literal-2} UPON merkname-6

    [ON EXCEPTION unbedingte-anweisung-1]
    [NOT ON EXCEPTION unbedingte-anweisung-2]
    [END-DISPLAY]
```

Syntaxregeln

1. bezeichner-2 muss sich auf ein alphanumerisches Datenelement beziehen, das ohne ANY LENGTH-Klausel definiert ist.
2. literal-2 muss ein alphanumerisches Literal sein.
3. merkname-6 muss im SPECIAL-NAMES-Paragrafen mit dem Herstellernamen ENVIRONMENT-VALUE verknüpft sein.
4. NOT ON EXCEPTION darf nur angegeben werden, wenn auch ON EXCEPTION angegeben ist.

Ein ausführliches Beispiel für den Zugriff auf Kommandozeile und Umgebungsvariable befindet sich im Handbuch „COBOL2000 Benutzerhandbuch“ [1].

8.10.12 DIVIDE-Anweisung

Funktion

Die DIVIDE-Anweisung führt die Division zweier numerischer Operanden durch und speichert das Ergebnis ab.

Format 1	der DIVIDE-Anweisung speichert die Quotienten im jeweiligen Dividendenfeld ab.
Format 2	der DIVIDE-Anweisung speichert den Quotienten in mehreren gesonderten Ergebnisfeldern ab.
Format 3	der DIVIDE-Anweisung verwendet die GIVING-Angabe zum Abspeichern des Quotienten und erzeugt den Divisionsrest unter Verwendung der REMAINDER-Angabe.

Format 1

```
DIVIDE {bezeichner-1 | literal-1} INTO {bezeichner-2 [ROUNDED]}...
    [ON SIZE ERROR unbedingte-anweisung-1]
    [NOT ON SIZE ERROR unbedingte-anweisung-2]
    [END-DIVIDE]
```

Syntaxregeln

1. Jeder Bezeichner muss sich auf ein numerisches Datenelement beziehen.
2. Der Wert von bezeichner-2 wird durch den Wert von bezeichner-1 oder literal-1 dividiert. Der Quotient ersetzt dann den Wert von bezeichner-2 etc.
3. Die Maximalgröße des Quotienten beträgt 31 Dezimalziffern.
4. Division durch Null ergibt immer eine Überlaufbedingung (SIZE ERROR).
5. Bei Division mit ON SIZE ERROR kann trotzdem ein Divisionsfehler auftreten, da nicht auf Überlauf des Quotienten geprüft wird (nur auf Division durch 0).

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die ROUNDED-Angabe, (NOT) ON SIZE ERROR-Angabe und GIVING-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-33

Anweisung	Maskenzeichenfolge des Ergebnisfeldes	Rechnung
DIVIDE A INTO B	9(4)V9(2)	B / A abgespeichert als nnnnVnn in B.

Format 2

```
DIVIDE {bezeichner-1 | literal-1} {INTO | BY} {bezeichner-2 | literal-2}
        GIVING {bezeichner-3 [ROUNDED]}...
       [ON SIZE ERROR unbedingte-anweisung-1]
       [NOT ON SIZE ERROR unbedingte-anweisung-2]
       [END-DIVIDE]
```

Syntaxregeln

1. bezeichner-1 oder bezeichner-2 müssen sich auf ein numerisches Datenelement beziehen.
2. bezeichner-3... kann sich auf ein numerisches Datenelement oder auf ein numerisch druckaufbereitetes Datenelement beziehen.

3. Bei Verwendung der INTO-Angabe wird der Wert von bezeichner-2 oder literal-2 durch den Wert von bezeichner-1 oder literal-1 dividiert; bei Verwendung der BY-Angabe wird der Wert von bezeichner-1 oder literal-1 durch den Wert von bezeichner-2 oder literal-2 dividiert. Der Quotient wird in bezeichner-3... abgespeichert.
4. Die Maximalgröße des Quotienten beträgt 31 Dezimalziffern.
5. Division durch Null ergibt immer eine Überlaufbedingung (SIZE ERROR).
6. Bei Division mit ON SIZE ERROR kann trotzdem ein Divisionsfehler auftreten, da nicht auf Überlauf des Quotienten geprüft wird (nur auf Division durch 0).

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die ROUNDED-Angabe, (NOT) ON SIZE ERROR-Angabe und GIVING-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-34

Anweisung	Maskenzeichenfolge desErgebnisfeldes	Rechnung
DIVIDE A INTO BGIVING C ROUNDED	S999V99 für C	B / A abgespeichert in C als nnnVnn, nachdem, wie verlangt, die Stelle ganz rechts gerundet wurde.
DIVIDE A BY B, GIVING C, DROUNDED	9(5) für C9(4) für D	A / B abgespeichert in C als nnnnn, in D als nnnn, nachdem, wie verlangt, die Stelle ganz rechts gerundet wurde.

Format 3

```
DIVIDE {bezeichner-1 | literal-1} {INTO | BY} {bezeichner-2 | literal-2} GIVING
bezeichner-3 [ROUNDED]

        REMAINDER bezeichner-4

       [ON SIZE ERROR unbedingte-anweisung-1]

       [NOT ON SIZE ERROR unbedingte-anweisung-2]

       [END-DIVIDE]
```

Syntaxregeln

1. bezeichner-1 oder bezeichner-2 müssen sich auf ein numerisches Datenelement beziehen.
2. bezeichner-3 oder bezeichner-4 können sich auf ein numerisches Datenelement oder auf ein numerisch druckaufbereitetes Datenelement beziehen.
3. Bei Verwendung der INTO-Angabe wird der Wert von bezeichner-2 oder literal-2 durch den Wert von bezeichner-1 oder literal-1 dividiert; bei Verwendung der BY-Angabe wird der Wert von bezeichner-1 oder literal-1 durch den Wert von bezeichner-2 oder literal-2 dividiert. Der Quotient wird in bezeichner-3 abgespeichert.
4. Bei Verwendung der REMAINDER-Angabe wird der Divisionsrest in bezeichner-4 abgespeichert.
   Der Divisionsrest wird errechnet, indem vom Dividenden das Produkt aus Quotient und Divisor subtrahiert wird.
   Ist bezeichner-3 als numerisch druckaufbereitetes Datenelement definiert, wird zur Berechnung des Divisionsrestes für den Quotienten ein Zwischenfeld verwendet, das den Wert in nicht druckaufbereiteter Form enthält.

Ist die ROUNDED-Angabe und REMAINDER-Angabe vorhanden, wird als Quotient bei der Berechnung des Divisionsrestes ein Zwischenfeld verwendet, das den Quotienten der DIVIDE-Anweisung in abgeschnittener statt abgerundeter Form enthält.

5. Ist ON SIZE ERROR angegeben und tritt beim Quotienten ein Überlauf auf, entfällt die Berechnung des Divisionsrestes. Daher bleibt in diesem Fall der Inhalt der Datenfelder, auf die sich bezeichner-3 und bezeichner-4 beziehen, unverändert.

   Wenn beim Divisionsrest ein Überlauf auftritt, bleibt der Wert des Datenfeldes, auf das sich bezeichner-4 bezieht, unverändert.
6. Die Genauigkeit des für die REMAINDER-Angabe notwendigen Datenfeldes (bezeichner-4) wird durch die oben beschriebenen Berechnungen festgelegt. Entsprechende Dezimalpunktausrichtung und Abschneidung (nicht Rundung) wird, wenn nötig, für den Inhalt des Datenfeldes, auf das sich bezeichner-4 bezieht, durchgeführt.
7. Die Maximalgröße des Quotienten beträgt 31 Dezimalziffern.
8. Division durch Null ergibt immer eine Überlaufbedingung.

Für weitere Regeln siehe unter „Angaben in Anweisungen"; die ROUNDED-Angabe, (NOT) ON SIZE ERROR-Angabe und GIVING-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-35

Anweisung	**Maskenzeichenfolgedes Ergebnisfeldes**	**Rechnung**
DIVIDE A BY B, GIVING C REMAINDER D	9(5) für C9(2) für D	A / B abgespeichert in C als nnnnn, der Rest (A - C * B) abgespeichert in D als nn.

8.10.13 ENTRY-Anweisung

Funktion

Die ENTRY-Anweisung wird in einem COBOL-Unterprogramm einer Ablaufeinheit verwendet, um den Namen der Einsprungstelle, mit dem das Unterprogramm aufgerufen wird, festzulegen (im Gegensatz zur PROCEDURE DIVISION-Überschrift als Standard-Einsprungstelle). Wahlweise können Datennamen angegeben werden, wenn vom aufrufenden Programm Daten als Parameter übernommen werden sollen.

Format

```
ENTRY literal [USING {datenname-1}... ].
```

Syntaxregeln

1. literal muss ein alphanumerisches Literal sein.

   literal muss ein gültiger Programmname sein, d.h. er muss mit einem alphabetischen Zeichen beginnen, darf nur Buchstaben und Ziffern enthalten und eine Länge von maximal 7 Zeichen haben.
2. Der durch das Literal benannte Programmname muss in den Programmen, die zu einer Ablaufeinheit zusammengebunden werden, eindeutig sein. Er darf auch nicht mit den ersten 7 Zeichen des Programmnamens im PROGRAM-ID-Paragrafen des Programms, das diese ENTRY-Anweisung enthält, übereinstimmen.
3. Die USING-Angabe darf nur geschrieben werden, wenn die dazugehörende CALL-Anweisung im aufrufenden Programm auch eine USING-Angabe enthält. Die Anzahl der Operanden in den sich entsprechenden USING-Angaben muss gleich sein, sonst ist das Ergebnis undefiniert.
4. Jeder in der USING-Angabe der ENTRY-Anweisung angegebene datenname-1... muss in der LINKAGE SECTION des Programms, das diese ENTRY-Anweisung enthält, als Datenfeld definiert sein und eine Stufennummer 01 oder 77 haben.
5. Die ENTRY-Anweisung darf nicht in den Programmen eines geschachtelten Programms oder in einer Methode verwendet werden.

Allgemeine Regeln

1. Die ENTRY-Anweisung legt die Einsprungstelle im aufgerufenen Programm fest. Der Name der Einsprungstelle wird durch das Literal benannt. Die Verknüpfung zu dieser Einsprungstelle wird durch eine CALL-Anweisung in einem anderen Programm, die sich auf diese Einsprungstelle bezieht, hergestellt.
2. Die USING-Angabe bewirkt, dass sich während des Ablaufs datenname-1 der ENTRY-Anweisung im aufgerufenen Programm und bezeichner-1 in der USING-Angabe der CALL-Anweisung im aufrufenden Programm auf dieselben Daten beziehen, die in gleicher Weise für das aufgerufene als auch für das aufrufende Programm verfügbar sind. Die Gleichheit der Namen ist nicht erforderlich.

   In der USING-Angabe der ENTRY-Anweisung im aufgerufenen Programm darf ein Datenname nur ein einziges Mal vorkommen, in der USING-Angabe der CALL-Anweisung darf derselbe bezeichner-1 jedoch mehrmals angegeben werden.
3. Im aufgerufenen Programm werden die Operanden der USING-Angabe entsprechend der im LINKAGE-Kapitel angegebenen Datenbeschreibungen behandelt.
4. Eine ENTRY-Anweisung darf nur einmal (beim Programmeinsprung) durchlaufen werden. Es dürfen keine weiteren ENTRY-Anweisungen folgen.
5. Wird ein Datenfeld in der CALL-Anweisung als Parameter BY CONTENT übergeben, wird der Wert des Datenfeldes vor Ausführung der CALL-Anweisung in einen Speicherbereich übertragen, der die in der CALL-Anweisung für bezeichner-2 angegebenen Eigenschaften besitzt. Jeder Parameter in der BY CONTENT-Angabe der CALL-Anweisung muss in Datentyp und Länge gleich dem entsprechenden Parameter in der USING-Angabe der Prozedurteilüberschrift sein.

8.10.14 EVALUATE-Anweisung

Funktion

Die EVALUATE-Anweisung beschreibt eine Struktur von Mehrfachverzweigungen und Mehrfachverbindungen. Sie kann die Auswertung mehrfacher Bedingungen bewirken. Die nachfolgenden Schritte des Programms hängen von den Ergebnissen dieser Auswertung ab.

Format

```
EVALUATE {bezeichner-1  | literal-1  | ausdruck-1  | TRUE | FALSE }
 [ALSO {bezeichner-2 | literal-2  | ausdruck-2 | TRUE | FALSE} ]...
{  {  WHEN     {  ANY | bedingung-1 | teilbedingung-1 | TRUE | FALSE
                  | [NOT] {  {bezeichner-3 | literal-3 | arithm-ausdruck-1}
                             [   {THROUGH | THRU}
                                 {bezeichner-4 | literal-4 | arithm-ausdruck-2}
                             ]
                          }
               }
      [  ALSO  {  ANY | bedingung-2 | teilbedingung-2 | TRUE | FALSE
                  | [NOT] {  {bezeichner-5 | literal-5 | arithm-ausdruck-3}
                             [   {THROUGH | THRU}
                                 {bezeichner-6 | literal-6 | arithm-ausdruck-4}
                             ]
                          }
               }
      ]...
   }...
   unbedingte-anweisung-1
} ...

[WHEN OTHER unbedingte-anweisung-2]

[END-EVALUATE]
```

Syntaxregeln

1. Die Operanden oder die Wörter TRUE und FALSE vor der ersten WHEN-Angabe der EVALUATE-Anweisung sind einzeln betrachtet Auswahlsubjekte (Subjekte) und insgesamt gesehen eine Folge von Auswahlsubjekten (Subjektfolge oder Subjektvektor).
2. Die Operanden oder die Wörter TRUE, FALSE und ANY in einer WHEN-Angabe sind einzeln betrachtet Auswahlobjekte (Objekte) und insgesamt gesehen eine Folge von Auswahlobjekten (Objektfolge oder Objektvektor).
3. Die Wörter THROUGH und THRU bedeuten das gleiche.
4. Zwei durch THROUGH verbundene Operanden müssen der gleichen Datenklasse angehören. Sie bilden ein einzelnes Objekt. Die verbundenen Operanden dürfen weder von der Klasse objekt noch von der Klasse zeiger sein.
5. Die Anzahl der Objekte innerhalb einer Objektfolge muss der Anzahl der Subjekte entsprechen.

6. Jedes Objekt einer Objektfolge muss mit demjenigen Subjekt einer Subjektfolge übereinstimmen, das die gleiche Position in der Reihenfolge besitzt, und zwar nach folgenden Regeln:
   a. Bezeichner, Literale oder arithmetische Ausdrücke eines Objekts müssen für den Vergleich mit dem entsprechenden Subjekt gültige Operanden sein - gemäß den Regeln für Vergleichsbedingungen.
   b. bedingung-1, bedingung-2, TRUE oder FALSE in einer Objektfolge müssen einem Bedingungsausdruck oder den Wörtern TRUE oder FALSE in der Subjektfolge entsprechen.
   c. Das Wort ANY kann jeder Kategorie von Subjekten entsprechen.
   d. teilbedingung-1 oder teilbedingung-2 darf nicht mit einem arithmetischen Operator beginnen.
   e. teilbedingung-1 oder teilbedingung-2 in einer Objektfolge müssen einem Bezeichner, einem Literal oder einem arithmetischen Ausdruck in der Subjektfolge entsprechen. Sie müssen so angegeben werden, dass durch das Einfügen des entsprechenden Subjekts vor der Teilbedingung eine gültige einfache Bedingung entsteht.

Allgemeine Regeln

1. Die Ausführung der EVALUATE-Anweisung wirkt so, als ob jedes Subjekt und jedes Objekt ausgewertet und ihnen ein numerischer oder nichtnumerischer Wert, ein Bereich mit numerischen oder nichtnumerischen Werten oder ein Wahrheitswert zugewiesen würde. Diese Werte sind wie folgt festgelegt:
   a. Jedes durch bezeichner-1, bezeichner-2 angegebene Subjekt und jedes durch bezeichner-3, bezeichner-5 ohne NOT oder THROUGH angegebene Objekt erhält Wert und Datenklasse des durch Bezeichner repräsentierten Datenfeldes.
   b. Jedes durch literal-1, literal-2 angegebene Subjekt und jedes durch literal-3, literal-5 ohne NOT oder THROUGH angegebene Objekt erhält Wert und Datenklasse des durch Literal repräsentierten Datenfeldes. Ist literal-3, literal-5 die figurative Konstante ZERO, wird die Datenklasse des entsprechenden Subjektes zugewiesen.
   c. Jedem Subjekt, in dem ausdruck-1, ausdruck-2 ein arithmetischer Ausdruck ist, und jedem Objekt ohne NOT oder THROUGH, in dem arithm-ausdruck-1, arithm-ausdruck-3 angegeben ist, wird ein numerischer Wert - gemäß den Regeln der Auswertung arithmetischer Ausdrücke - zugewiesen.
   d. Jedem Subjekt, in dem ausdruck-1, ausdruck-2 ein Bedingungsausdruck ist, und jedem Objekt mit bedingung-1, bedingung-2 werden Wahrheitswerte - gemäß den Regeln der Auswertung von Bedingungsausdrücken - zugewiesen.
   e. Jedem durch TRUE oder FALSE angegebenen Subjekt oder Objekt wird ein Wahrheitswert zugewiesen. Der Wahrheitswert „wahr“ bezieht sich auf die Datenfelder, die mit TRUE, der Wahrheitswert „falsch“ auf die Datenfelder, die mit FALSE angegeben sind.
   f. Jedes Objekt, das durch das Wort ANY bezeichnet ist, wird nicht weiter ausgewertet.
   g. Ist für ein Objekt THROUGH ohne NOT angegeben, umfasst der Wertebereich alle zulässigen Werte des Subjekts, die größer oder gleich dem ersten Operanden und kleiner oder gleich dem zweiten Operanden sind.
   h. Ist für ein Objekt NOT angegeben, umfasst der Wertebereich alle zulässigen Werte des Subjekts, die nicht gleich dem Wert sind bzw. nicht in dem Wertebereich liegen, der ohne NOT-Angabe dem Datenfeld zugewiesen worden wäre.
   i. Ist für ein Objekt eine Teilbedingung angegeben, so wird ihm der Wahrheitswert der Bedingung zugewiesen, die sich durch das Anfügen des Objekts an das Ergebnis der Auswertung des zugehörigen Subjekts ergibt.
2. Die EVALUATE-Anweisung wird so ausgeführt, als ob die den Subjekten und Objekten zugewiesenen Werte miteinander verglichen werden, um festzustellen, ob eine WHEN-Angabe die Bedingungen der Subjektfolge erfüllt. Dieser Vergleich läuft wie folgt ab:

a. Jedes einzelne Objekt innerhalb der Objektfolge der ersten WHEN-Angabe wird mit dem Subjekt verglichen, das innerhalb der Subjektfolge dieselbe Position in der Reihenfolge besitzt. Dabei muss eine der folgenden Bedingungen erfüllt sein:
   - Sind den zu vergleichenden Datenfeldern numerische oder nichtnumerische Werte oder ein Bereich von numerischen oder nichtnumerischen Werten zugewiesen, so ist der Vergleich dann erfüllt, wenn der dem Objekt zugewiesene Wert oder Wert des Bereichs gleich dem Wert ist, der dem Subjekt zugewiesen wurde (siehe „Vergleichsbedingung").
   - Ist das Objekt eine Teilbedingung, so ist der Vergleich dann erfüllt, wenn dem Objekt der Wahrheitswert „wahr“ zugewiesen wurde.
   - Wurden den zu vergleichenden Datenfeldern Wahrheitswerte zugewiesen, so ist der Vergleich dann erfüllt, wenn den Datenfeldern der gleiche Wahrheitswert zugewiesen wird.
   - Ist das zu vergleichende Objekt mit ANY angegeben, so ist der Vergleich immer erfüllt, egal, welchen Wert das Subjekt hat.

b. Ist der oben genannte Vergleich für jedes Objekt der zu vergleichenden Objektfolge erfüllt, so wird die WHEN-Angabe, die diese Objektfolge enthält, als diejenige ausgewählt, die die Bedingungen der Subjektfolge erfüllt.

c. Ist der oben genannte Vergleich für ein oder mehrere Objekte der Objektfolge nicht erfüllt, so erfüllt die Objektfolge nicht die Bedingungen der Subjektfolge.

d. Dieser Vorgang wiederholt sich für alle weiteren Objektfolgen in der Reihenfolge ihres Auftretens in der Übersetzungseinheit solange, bis entweder eine WHEN-Angabe ausgewählt ist, die die Bedingungen der Subjektfolge erfüllt, oder bis alle Objektfolgen verglichen sind.

3. Ist die Vergleichsoperation beendet, wird die Ausführung der EVALUATE-Anweisung folgendermaßen fortgesetzt:

   a. Wurde eine WHEN-Angabe ausgewählt, wird die Ausführung mit der ersten unbedingten Anweisung, die dieser WHEN-Angabe folgt, fortgesetzt. Nach Ausführung der letzten unbedingten Anweisung wird zu END-EVALUATE verzweigt.

   b. Wurde keine WHEN-Angabe ausgewählt und eine WHEN OTHER-Angabe gemacht, wird die Ausführung mit der unbedingten Anweisung der WHEN OTHER-Angabe fortgesetzt.

   c. Eine EVALUATE-Anweisung ist beendet, wenn eine der folgenden Bedingungen erfüllt ist:
      - unbedingte-anweisung-1 der ausgewählten WHEN-Angabe ist ausgeführt,
      - unbedingte-anweisung-2 ist ausgeführt,
      - es wurde keine WHEN-Angabe ausgewählt und keine WHEN OTHER-Angabe gemacht.

Beispiel 8-36

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EVAL1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 A PIC 9.
77 B PIC 9.
77 C PIC 9.
77 D PIC 9.
PROCEDURE DIVISION.
PROC SECTION.
DIALOG.
```

```
    DISPLAY "Wert fuer A eingeben" UPON T.
    ACCEPT A FROM T.
    DISPLAY "Wert fuer B eingeben" UPON T.
    ACCEPT B FROM T.
    DISPLAY "Wert fuer C eingeben" UPON T.
    ACCEPT C FROM T.
    DISPLAY "Wert fuer D eingeben" UPON T.
    ACCEPT D FROM T.
TEST1.
    EVALUATE A + B ALSO C + D
    WHEN 5 ALSO 5
      DISPLAY "Werte richtig" UPON T
    WHEN OTHER
      DISPLAY "Werte falsch" UPON T
    END-EVALUATE.
ENDE.
    STOP RUN.
```

Beispiel 8-37

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BSP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  BESTELLART PIC 9.
    88 VORORT        VALUE 1.
    88 SCHRIFTLICH   VALUE 2 THRU 4.
01  KUNDENART PIC X.
    88 PRIVAT         VALUE "1".
    88 GEWERBLICH     VALUE "2".
01  GEWICHT       PIC 9999.
01 VERSANDART     PIC 9.
   88 ABHOLUNG    VALUE 1.
   88 POST        VALUE 2.
   88 BAHN        VALUE 3.
   88 UPS         VALUE 4.
PROCEDURE DIVISION.
PROC SECTION.
DIALOG.
    DISPLAY "Bestellart eingeben" UPON T.
    DISPLAY "  Vorort = 1, Schriftlich = 2-4 " UPON T.
    ACCEPT BESTELLART FROM T.
    DISPLAY "Kundenart eingeben" UPON T.
    DISPLAY "Gewerblich = 2 , Privat = 1 " UPON T.
    ACCEPT KUNDENART FROM T.
    DISPLAY "Gewicht eingeben" UPON T.
    ACCEPT GEWICHT FROM T.
BESTIMMUNG-VERSANDART.
    EVALUATE TRUE ALSO TRUE ALSO TRUE
    WHEN PRIVAT ALSO VOR-ORT ALSO ANY
    WHEN GEWERBLICH ALSO VOR-ORT ALSO ANY
```

```
        SET ABHOLUNG TO TRUE
      WHEN PRIVAT ALSO SCHRIFTLICH ALSO GEWICHT < 5
        SET POST TO TRUE
      WHEN GEWERBLICH ALSO SCHRIFTLICH ALSO GEWICHT < 10
        SET UPS TO TRUE
      WHEN OTHER SET BAHN TO TRUE
      END-EVALUATE.
AUSGABE.
      DISPLAY "Versandart = " VERSANDART UPON T.
      STOP RUN.
```

Erläuterungen

Die Objekte VORORT, SCHRIFTLICH, PRIVAT, GEWERBLICH (= Bedingungsnamen) werden auf ihre Wahrheitswerte hin abgeprüft. Die Subjekte werden durch die drei TRUE dargestellt; alle drei Subjekte (= Subjektfolge) besitzen den Wahrheitswert „Wahr“. Eine Objektfolge erfüllt dann die Bedingung der Subjektfolge, wenn alle Objekte innerhalb der Folge, außer die durch ANY bezeichneten, den Wahrheitswert „Wahr“ zugewiesen bekommen (eine WHEN-Angabe entspricht einer Objektfolge). Die dieser WHEN-Angabe folgende Anweisung wird dann ausgeführt. Erfüllt keine der angegebenen Objektfolgen den Vergleich, so wird die Anweisung der WHEN OTHER-Angabe ausgeführt. Die Angabe ANY muss in den ersten beiden WHEN-Angaben verwendet werden, da in ihnen nur zwei und in den anderen WHEN-Angaben drei Bedingungsnamen auf ihren Wahrheitswert hin abgeprüft werden und die Anzahl der Objekte innerhalb der Objektfolge der Anzahl der Subjekte entsprechen muss.

8.10.15 EXIT-Anweisung

Funktion

Die EXIT-Anweisung stellt einen allgemeinen Ausgang am Ende einer Reihe von Prozeduren bereit.

Format

```
EXIT.
```

Syntaxregeln

1. Der EXIT-Anweisung muss ein Paragrafenname vorangehen. Sie muss die einzige Anweisung in einem Paragrafen sein.
2. Die EXIT-Anweisung dient nur dazu, einer bestimmten Stelle im Programm einen Prozedurnamen zuzuordnen, ohne dass dies Auswirkung auf den Ablauf des Programms hat.

Allgemeine Regeln

1. Durch die EXIT-Anweisung am Ende einer Reihe von Prozeduren ist es möglich, den normalen Ablauf dieser Prozedur zu unterbrechen und das Programm direkt am Ende der Prozedurfolge fortzusetzen.
2. Wenn beim Ablauf ein EXIT-Paragraf erreicht wird und keine zugehörige PERFORM- oder USE-Anweisung aktiv ist, wird der Ablauf beim ersten Programmsatz des nächsten Paragrafen fortgesetzt.

Beispiel 8-38

```
PROCEDURE DIVISION.
    ...
    PERFORM X-PAR THRU Y-PAR.
    ...
X-PAR.
    ...
    IF A IS ZERO, GO TO Y-PAR.
    ...
Y-PAR.
    EXIT.
Z-PAR.
    ...
```

In diesem Fall ist der EXIT-Paragraf die letzte Prozedur, die von der PERFORM-Anweisung abgedeckt wird. Ist der Wert von A gleich Null, wird mit der GO TO-Anweisung der normale Ablauf der im Bereich von X-PAR bis Y-PAR liegenden Anweisungen unterbrochen und direkt zum Ende des durch die PERFORM-Anweisung angegebenen Prozedurbereiches verzweigt. Danach wird der Programmablauf hinter der PERFORM-Anweisung fortgesetzt.

8.10.16 EXIT METHOD-Anweisung

Die Anweisung EXIT METHOD kennzeichnet das logische Ende einer aufgerufenen Methode.

Format

```
EXIT METHOD
```

Syntaxregel

1. Die EXIT METHOD-Anweisung darf nur in der PROCEDURE DIVISION einer Methode angegeben werden.

Allgemeine Regel

1. Bei der Ausführung der EXIT METHOD-Anweisung wird die Methode beendet und die Kontrolle an das aufrufende Programm zurückgegeben, das die Bearbeitung nach der INVOKE-Anweisung fortsetzt.

8.10.17 EXIT PARAGRAPH-Anweisung

Die EXIT PARAGRAPH-Anweisung bewirkt eine Verzweigung auf eine angenommene CONTINUE-Anweisung, die sich am Ende des aktuellen Paragrafen befindet.

Format

```
EXIT PARAGRAPH
```

Syntaxregel

1. Die EXIT PARAGRAPH-Anweisung darf nur innerhalb eines Paragrafen angegeben werden.

8.10.18 EXIT PERFORM-Anweisung

Funktion

Die EXIT PERFORM-Anweisung bietet die Möglichkeit, aus einer „in-line“- PERFORM-Anweisung zum Ende der PERFORM-Anweisung oder zur Wiederholung der Schleife zu verzweigen.

Format

```
EXIT {[TO TEST OF] PERFORM | PERFORM CYCLE}
```

Syntaxregeln

1. Eine EXIT [TO TEST OF] PERFORM-Anweisung kann nur innerhalb eines „in-line“-Perform angegeben werden.
2. Eine EXIT TO TEST OF PERFORM-Anweisung kann sich nur auf PERFORM-Anweisungen vom Format 2, 3 oder 4 beziehen (siehe „PERFORM-Anweisung").
3. Die Anweisung EXIT PERFORM CYCLE ist äquivalent zur Anweisung EXIT TO TEST OF PERFORM.

Allgemeine Regeln

1. Bei EXIT PERFORM wird die zugehörige „in-line“-PERFORM-Anweisung verlassen.
2. Abhängig vom Format der PERFORM-Anweisung bewirkt
   EXIT TO TEST OF PERFORM unterschiedliche Verzweigungen:
   a. Bei Format 2 wird zum Test von „Ende der Schleife“ verzweigt.
   b. Bei Format 3 wird zum Test von „UNTIL bedingung“ verzweigt.
   c. Bei Angabe von TEST AFTER in Format 4 wird zum Test von „UNTIL bedingung“ verzweigt. Ist die Bedingung erfüllt, wird die PERFORM-Anweisung beendet, andernfalls wird die Inkrementierung durchgeführt.
   d. Bei Angabe von TEST BEFORE in Format 4 wird zum Inkrementieren verzweigt. Anschließend erfolgt der Test von „UNTIL bedingung“.

Beispiel 8-39

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXITP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  EINGABE        PIC 99.
01  EINGABE-STATUS PIC X VALUE LOW-VALUE.
   88  EINGABE-ENDE   VALUE HIGH-VALUE.
PIC 9(3).PIC 99.PIC Z9.9(2).
PROCEDURE DIVISION.
PROC SECTION.
BERECHNUNG.
    MOVE 0 TO ZAEHLER SUMME
    DISPLAY "Berechnung des Durchschnitts von Zahlen" UPON T
    PERFORM WITH TEST AFTER UNTIL EINGABE-ENDE
       DISPLAY "ZAHLEN EINGEBEN (2 Stellen, Ende bei 00)" UPON T
       ACCEPT EINGABE FROM T
       IF EINGABE IS NOT NUMERIC
```

```
          THEN
            DISPLAY "Eingabe nicht numerisch/nicht zweistellig!" UPON T
            EXIT TO TEST OF PERFORM
          END-IF
          IF EINGABE = 0
          THEN
            SET EINGABE-ENDE TO TRUE
            EXIT TO TEST OF PERFORM
          END-IF
          ADD 1 TO ZAEHLER
          ADD EINGABE TO SUMME
       END-PERFORM
       IF ZAEHLER > 0
       THEN
         COMPUTE DURCHSCHNITT ROUNDED = SUMME / ZAEHLER
         DISPLAY "Durchschnitt = " DURCHSCHNITT UPON T
       ELSE
         DISPLAY "Kein Durchschnitt berechnet" UPON T
       END-IF
       STOP RUN.
```

Das Programm berechnet den Durchschnitt von Zahlen, die am Terminal eingegeben werden. Endekriterium ist die Eingabe von 00, Fehleingaben werden gemeldet.

8.10.19 EXIT PROGRAM-Anweisung

Funktion

Die EXIT PROGRAM-Anweisung kennzeichnet das dynamische Ende eines aufgerufenen Programms.

Format

```
EXIT PROGRAM
```

Syntaxregeln

1. Falls eine EXIT PROGRAM-Anweisung eine von mehreren unbedingten Anweisungen eines Programmsatzes ist, muss sie die letzte Anweisung dieses Programmsatzes sein.
2. Eine EXIT PROGRAM-Anweisung darf nicht in einer globalen USE-Prozedur benutzt werden.

Allgemeine Regeln

1. Bei der Ausführung der EXIT PROGRAM-Anweisung in einem aufgerufenen Programm wird die Steuerung an die rufende Anweisung zurückgegeben. Eine EXIT PROGRAM-Anweisung in einem Programm, das nicht als Unterprogramm aufgerufen wurde, bleibt ohne Wirkung.
2. Der Programmzustand des aufrufenden Programms bleibt unverändert und ist identisch mit dem Zustand, der zum Zeitpunkt der Ausführung der CALL-Anweisung im aufrufenden Programm existierte. Dies gilt nicht für Daten, die bei der CALL-Anweisung an das aufgerufene Programm übergeben wurden und von diesem geändert wurden.
3. Die EXIT PROGRAM-Anweisung in einem Programm mit dem Attribut INITIAL hat zusätzlich dieselbe Wirkung wie eine CANCEL-Anweisung für dieses Programm.
4. Im aufgerufenen Programm werden die Steuerungsmechanismen für alle PERFORM-Anweisungen auf ihren Anfangswert gesetzt.

8.10.20 EXIT SECTION-Anweisung

Die EXIT SECTION-Anweisung bewirkt eine Verzweigung auf eine angenommene CONTINUE-Anweisung, die sich am Ende des letzten Paragrafen des aktuellen Kapitels (SECTION) befindet.

Format

```
EXIT SECTION
```

Syntaxregel

1. Die EXIT SECTION-Anweisung darf nur innerhalb eines Kapitels (SECTION) angegeben werden.

8.10.21 FREE-Anweisung

Funktion

Die FREE-Anweisung gibt Speicher frei, der vorher mit einer ALLOCATE-Anweisung (siehe "ALLOCATE-Anweisung") angefordert wurde.

Format

```
FREE {datenname-1}...
```

Syntaxregel

1. datenname-1 muss von der Kategorie datenzeiger sein.

Allgemeine Regeln

Die FREE-Anweisung wird wie folgt ausgeführt:

1. Wenn der Datenzeiger datenname-1 auf den Anfang vom Speicher zeigt, der vorher mit der ALLOCATE-Anweisung angefordert wurde, wird dieser Speicher freigegeben und der Datenzeiger datenname-1 wird auf NULL gesetzt. Die Länge des freigegebenen Speichers ist gleich der Länge des Speichers, die mit ALLOCATE angefordert wurde. Der Inhalt irgendeines Datenfeldes innerhalb des freigegebenen Speicherbereichs wird undefiniert.
2. Wenn der Datenzeiger datenname-1 die vordefinierte Adresse NULL enthält, wird kein Speicher freigegeben.
3. In allen anderen Fällen entsteht die Ausnahmesituation EC-STORAGE-NOT-ALLOC. datenname-1 wird nicht verändert. Ist die Überprüfung für EC-STORAGE-NOT--ALLOC eingeschaltet, so wird der zugehörige Ausnahmezustand ausgelöst und in die entsprechende USE-Prozedur, falls vorhanden, verzweigt. Bei nicht eingeschalteter Überprüfung der Ausnahmesituation oder nach Ausführung einer USE-Prozedur, die mit RESUME AT NEXT STATEMENT verlassen wird, wird der Programmlauf entsprechend Regel 4 fortgesetzt.

   > ¡ Die FREE-Anweisung kann bei ausgeschalteter Überprüfung der Ausnahmesituation EC-STORAGE-NOT-ALLOC zu einem Absturz führen.

4. Ist mehr als ein datenname-1 angegeben, dann wirkt dies, als wäre für jeden datenname-1 eine separate FREE-Anweisung in der vorgegebenen Reihenfolge angegeben worden. Entsteht durch eine dieser impliziten FREE-Anweisungen die Ausnahmesituation EC-STORAGE-NOT-ALLOC, so wird der Programmlauf (ggf. nach Rückkehr aus der entsprechenden USE-Prozedur) bei der nächsten impliziten FREE-Anweisung, falls vorhanden, fortgesetzt.

8.10.22 GOBACK-Anweisung

Funktion

Die GOBACK-Anweisung kennzeichnet das logische Ende eines Programms.

Format

```
GOBACK
```

Syntaxregel

1. Die GOBACK - Anweisung darf nicht in einer globalen USE-Prozedur und nicht in einer Methode verwendet werden.

Allgemeine Regeln

1. Die GOBACK-Anweisung wirkt wie die Folge:
   EXIT PROGRAM
   STOP RUN
2. Für GOBACK gelten daher die gleichen Regeln wie für EXIT PROGRAM und STOP RUN.

8.10.23 GO TO-Anweisung

Funktion

Mit Hilfe der GO TO-Anweisung wird der Programmablauf am angegebenen Sprungziel fortgesetzt.

Format 1 der GO TO-Anweisung verzweigt zu einer bestimmten Prozedur.

Format 2 der GO TO-Anweisung verzweigt zu einer von mehreren bestimmten Prozeduren, abhängig vom Wert eines Datenfeldes.

Format 1

```
GO TO [ prozedurname ]
```

Syntaxregeln

1. Falls die GO TO-Anweisung in einer durchgehenden Folge von unbedingten Anweisungen innerhalb eines Programmsatzes vorkommt, muss sie als letzte Anweisung in diesem Programmsatz erscheinen.
2. Einer GO TO-Anweisung, auf die sich eine ALTER-Anweisung bezieht, muss ein Paragrafenname vorangehen, und sie muss die einzige Anweisung in diesem Paragrafen sein (siehe „ALTER-Anweisung").
3. Einer GO TO-Anweisung ohne Angabe von prozedurname muss ein Paragrafenname vorausgehen, und sie muss die einzige Anweisung in diesem Paragrafen sein.

Allgemeine Regeln

1. Wird eine GO TO-Anweisung ausgeführt, so wird der Ablauf bei prozedurname fortgesetzt.
2. Wenn prozedurname nicht angegeben ist, muss eine ALTER-Anweisung, die sich auf die GO TO-Anweisung bezieht, vor der GO TO-Anweisung ausgeführt sein.
3. Die GO TO-Anweisung ohne Angabe von prozedurname muss durch eine ALTER-Anweisung mit einem Sprungziel versehen werden. Ist dies nicht der Fall, wird das Programm mit der Fehlermeldung 9142 abgebrochen.

Beispiel 8-40

```
     GO TO ENDE-ROUTINE.
     ...
ENDE-ROUTINE.
     CLOSE KARTEN-DATEI, DRUCK-DATEI.
     STOP RUN.
```

In diesem Beispiel verzweigt die GO TO-Anweisung zu der Prozedur ENDE-ROUTINE; die CLOSE-Anweisung wird direkt im Anschluss an die GO TO-Anweisung ausgeführt.

Format 2

```
GO TO {prozedurname-1}...

      DEPENDING ON bezeichner
```

Syntaxregel

1. bezeichner ist der Name eines als ganzzahlig beschriebenen numerischen Datenelements. Das Datenformat muss entweder DISPLAY, COMPUTATIONAL, COMPUTATIONAL-5, BINARY, COMPUTATIONAL-3 oder PACKED-DECIMAL sein.

Allgemeine Regel

1. Wenn eine GO TO-Anweisung ausgeführt wird, wird der Ablauf bei prozedurname-1... fortgesetzt, entsprechend dem Wert des bezeichners, der 1, 2, ... n betragen kann.

   Falls der Bezeichner einen anderen Wert als 1, 2, ... n hat, findet keine Verzweigung statt und der Ablauf wird bei der nächsten Anweisung in der normalen Ablauffolge fortgesetzt (n stellt die angegebene Anzahl von Prozedurnamen dar).

Beispiel 8-41

```
77  A PICTURE 9.
...
    MOVE 3 TO A.
    ...
    GO TO X-PAR Y-PAR Z-PAR DEPENDING ON A.
```

Da zum Zeitpunkt der Ausführung der GO TO-Anweisung der Wert von A 3 beträgt, wird der Ablauf bei Z-PAR, dem dritten Prozedurnamen der Reihe, fortgesetzt.

8.10.24 IF-Anweisung

Funktion

Die IF-Anweisung bewirkt, dass eine Bedingung (siehe unter „Bedingungen") ausgewertet wird. Die nachfolgende Aktion des Programmes hängt davon ab, ob die Bedingung wahr oder falsch ist.

Format 1

```
IF bedingung THEN anweisung-1 [ELSE anweisung-2]

   END-IF
```

Format 2

```
IF bedingung THEN {anweisung-1 | NEXT SENTENCE} [ELSE {anweisung-2 | NEXT SENTENCE
}]
```

Syntaxregeln

1. anweisung-1 und anweisung-2 können aus einer oder mehreren unbedingten oder bedingten Anweisung (en) bestehen.
2. Eine IF-Anweisung mit NEXT SENTENCE-Angabe darf nicht in anweisung-1 bzw. anweisung-2 einer IF-Anweisung vom Format 1 stehen, die unmittelbar mit END-IF abgeschlossen wird.

Allgemeine Regeln

1. anweisung-1 und/oder anweisung-2 können eine IF-Anweisung enthalten. In diesem Fall spricht man von „geschachtelter IF-Anweisung“.
2. Der Gültigkeitsbereich der IF-Anweisung kann wie folgt begrenzt werden:
   a. durch END-IF auf derselben Schachtelungsebene
   b. mit einem Punkt
   c. bei Schachtelung mit einer ELSE-Angabe, die zu einer IF-Anweisung auf höherer Schachtelungsebene gehört.
3. IF-Anweisungen innerhalb von IF-Anweisungen werden von links nach rechts als Kombinationen von IF, ELSE und END-IF betrachtet. Daher wird bei jedem auftretenden ELSE angenommen, dass es sich auf das unmittelbar vorausgegangene IF bezieht, dem noch kein ELSE oder END-IF zugeordnet ist.
4. Eine IF-Anweisung wird folgendermaßen ausgeführt:
   a. Ist die Bedingung wahr und anweisung-1 angegeben, wird mit der ersten Anweisung von anweisung-1 fortgesetzt. Wird dabei ein Prozedursprung oder eine Bedingungsanweisung ausgeführt, die einen expliziten Übergang der Steuerung bewirkt, erfolgt dieser Übergang gemäß den Regeln für diese Anweisung. Nach Beendigung der Ausführung von anweisung-1 wird ELSE, falls angegeben, ignoriert, und die Steuerung geht zum Ende der IF-Anweisung über.
   b. Ist die Bedingung wahr und statt anweisung-1 NEXT SENTENCE angegeben, wird ELSE, falls angegeben, ignoriert und der Ablauf bei der nächsten ausführbaren Anweisung fortgesetzt.
   c. Ist die Bedingung falsch und anweisung-2 angegeben, wird anweisung-1 bzw. NEXT SENTENCE ignoriert. Es wird mit der ersten Anweisung von anweisung-2 fortgesetzt. Wird dabei ein Prozedursprung oder eine Bedingungsanweisung ausgeführt, die einen expliziten Übergang der Steuerung bewirkt, erfolgt dieser Übergang gemäß den Regeln für diese Anweisung. Nach Beendigung der Ausführung von anweisung-2 geht die Steuerung zum Ende der IF-Anweisung über.
   d. Ist die Bedingung falsch und ELSE nicht angegeben, wird anweisung-1 ignoriert, und die Steuerung geht zum Ende der IF-Anweisung über.

e. Ist die Bedingung falsch und ELSE NEXT SENTENCE angegeben, wird anweisung-1 ignoriert und der Ablauf mit dem nächsten ausführbaren Programmsatz fortgesetzt.

Beispiel 8-42

```
IF A = B
THEN
  anweisung-1
END-IF
anweisung-2.
```

Wenn A = B wahr ist, werden anweisung-1 und anweisung-2 ausgeführt.

Wenn A = B falsch ist, wird nur anweisung-2 ausgeführt.

Beispiel 8-43

```
IF A = B
THEN
  anweisung-1
ELSE
  anweisung-2
END-IF
anweisung-3.
```

Wenn A = B wahr ist, werden anweisung-1 und anweisung-3 ausgeführt.

Wenn A = B falsch ist, werden anweisung-2 und anweisung-3 ausgeführt.

Beispiel 8-44

```
IF A = B
THEN
  CONTINUE
ELSE
  anweisung-1
END-IF
anweisung-2.
```

Wenn A = B wahr ist, wird anweisung-2 ausgeführt.

Wenn A = B falsch ist, werden anweisung-1 und anweisung-2 ausgeführt.

Beispiel 8-45

```
|-> IF MAENNLICH
|   |-> IF VERHEIRATET
|   |      ADD 1 TO MAENNL-VERHEIRATET
|   |-> ELSE
|   |   |-> IF GESCHIEDEN
|   |   |      ADD 1 TO MAENNL-GESCH
|   |   |-> ELSE
|   |   |      ADD 1 TO MAENNL-LEDIG
|   |   |-> END-IF
|   |-> END-IF
```

```
|-> ELSE
|       ADD 1 TO WEIBLICH
|-> END-IF
    nächste Anweisung
```

Dies ist ein Strukturbeispiel für eine geschachtelte IF-Anweisung. Die Pfeile bezeichnen die IF-, ELSE- und END-IF-Zuordnungen.

8.10.25 INITIALIZE-Anweisung

Funktion

Die INITIALIZE-Anweisung ermöglicht es, ausgewählte Datenfelder mit spezifischen Werten zu versorgen.

Format

```
INITIALIZE {bezeichner-1} ... [WITH FILLER]

    [{ ALL | kategoriename...} TO VALUE  ]

    [ THEN REPLACING {kategoriename DATA BY {bezeichner-2 | literal-1}}... ]

[THEN TO DEFAULT ]
```

dabei steht `kategoriename` für:

```
{  ALPHABETIC
 | ALPHANUMERIC
 | ALPHANUMERIC-EDITED
 | DATA-POINTER
 | NATIONAL
 | NUMERIC
 |  NUMERIC-EDITED
 | OBJECT-REFERENCE
 | PROGRAM-POINTER
}
```

Syntaxregeln

1. bezeichner-1 ist das Empfangsfeld der INITIALIZE-Anweisung. literal-1 und bezeichner-2 sind Sendefelder.
2. Für die Sendefelder gilt:
   - Wenn DATA-POINTER in der REPLACING-Angabe genannt ist, muss bezeichner-2 gültiges Sendefeld einer SET-Anweisung sein, deren Empfangsfeld der Kategorie datenzeiger angehört.
   - Wenn PROGRAM-POINTER in der REPLACING-Angabe genannt ist, muss bezeichner-2 ein gültiges Sendefeld einer SET-Anweisung sein, deren Empfangsfeld der Kategorie programmzeiger angehört.
   - Wenn OBJECT-REFERENCE in der REPLACING-Angabe genannt ist, muss bezeichner-2 gültiges Sendefeld einer SET-Anweisung sein, deren Empfangsfeld der Kategorie objektreferenz angehört.
   - Jede andere in der REPLACING-Angabe genannte Datenkategorie muss als Datenkategorie eines Empfangsfeldes in einer MOVE-Anweisung zulässig sein, in der bezeichner-2 oder literal-1 das Sendefeld darstellen.
3. Jede Datenkategorie darf nur einmal in der VALUE-Angabe und in der REPLACING-Angabe genannt werden.
4. Die Datenerklärung von bezeichner-1 oder eines Datenfeldes, das bezeichner-1 untergeordnet ist, darf nicht die DEPENDING-Angabe der OCCURS-Klausel enthalten.
5. Ein Indexdatenfeld darf nicht als Operand einer INITIALIZE-Anweisung auftreten.
6. Die Datenerklärung von bezeichner-1 darf nicht die RENAMES-Klausel enthalten.
7. Ist bezeichner-1 ein Tabellenelement oder enthält bezeichner-1 Tabellen, werden die VALUE-Angaben für Tabellenelemente ignoriert. Enthält die INITIALZE-Anweisung jedoch REPLACING- oder DEFAULT-Angaben, kommen statt dessen diese zur Wirkung.

Allgemeine Regeln

1. Ist bezeichner-1 eine nationale Gruppe, so wird er wie eine Gruppe verarbeitet. Ist bezeichner-2 eine nationale Gruppe, so wird er wie ein Datenelement verarbeitet.
2. Die Schlüsselwörter in kategoriename entsprechen den Kategorien von Daten (siehe unter "3. Klassen und Kategorien von Daten und Literalen"). Das Wort ALL in der VALUE-Angabe bedeutet dasselbe, als wäre jede Kategorie aus kategoriename angegeben worden.
3. Ist mehr als ein bezeichner-1 angegeben, dann wirkt dies, als wurde für jeden bezeichner-1 eine separate INITIALIZE-Anweisung in der vorgegebenen Reihenfolge angegeben. Entsteht durch eine dieser impliziten INITIALIZE-Anweisungen die Ausnahmesituation EC-DATA-CONVERSION, so wird der Programmlauf nach Ausführung einer USE-Prozedur, die mit RESUME AT NEXT STATEMENT verlassen wird, bei der nächsten impliziten INITIALIZE-Anweisung, falls vorhanden, fortgesetzt.
4. Unabhängig davon, ob bezeichner-1 ein Datenelement oder eine Gruppe darstellt, werden alle Übertragungen so ausgeführt, als sei eine Folge von MOVE- oder SET-Anweisungen angegeben, von denen jede ein Datenelement als Empfangsfeld enthält. Die Empfangsfelder dieser impliziten Anweisungen werden in Regel 3, die Sendefelder in Regel 4 bestimmt.
   Die Initialisierung wird wie folgt durchgeführt:
   - Falls die Kategorie datenzeiger, programmzeiger oder objektreferenz ist, wirkt bezeichner-2 als Sendefeld einer impliziten SET-Anweisung, deren Empfangsfeld jeweils ein dem Bezeichner bezeichner-1 untergeordnetes elementares Empfangsfeld ist.
   - Falls die Kategorie nicht datenzeiger, programmzeiger oder objektreferenz ist, wirkt bezeichner-2 oder literal-1 als Sendefeld einer impliziten MOVE-Anweisung, deren Empfangsfeld jeweils ein dem Bezeichner bezeichner untergeordnetes elementares Empfangsfeld ist.
5. Das Empfangsfeld in jeder impliziten MOVE- oder SET-Anweisung wird bestimmt durch die Anwendung folgender Schritte in der festgelegten Reihenfolge:
   a. Für die Ermittlung der elementaren Empfangsfelder werden zunächst folgende Datenelemente ausgeschlossen:
      i. Indexdatenfelder.
      ii. Datenfelder, die einem bezeichner-1 untergeordnet sind und deren Datenerklärung die REDEFINES-Klausel enthält und Datenfelder, die einem solchen Feld untergeordnet sind.
      iii. Datenfelder, die einem bezeichner-1 untergeordnet sind und deren Datenerklärung die RENAMES-Klausel enthält.
      iv. FILLER-Datenelemente, wenn die Angabe WITH FILLER fehlt.

      Anmerkung:
      bezeichner-1 selbst darf jedoch in seiner Datenerklärung die REDEFINES-Klausel enthalten oder einem Datenfeld mit einer REDEFINES-Klausel untergeordnet sein.
   b. Ein Datenelement ist ein potentielles Empfangsfeld, wenn:
      i. bezeichner-1 ein Datenelement ist,
      ii. oder wenn es bezeichner-1 untergeordnet ist. Wenn das Datenelement ein Tabellenelement ist, so ist jedes Vorkommen ein potentielles Empfangsfeld.
   c. Schließlich ist jedes potentielle Empfangsfeld wirklich ein Empfangsfeld, wenn mindestens eine der folgenden Bedingungen in dieser Reihenfolge zutrifft:
      i. Die INITIALIZE-Anweisung enthält die VALUE-Angabe, die Kategorie des Datenelements ist in der VALUE-Angabe explizit oder implizit enthalten und eine der folgenden Bedingungen ist erfüllt:
         a. Die Kategorie des Datenelements ist entweder datenzeiger, objektreferenz oder programmzeiger, oder
         b. eine VALUE-Klausel (beschrieben durch Format 1) ist in der Datenbeschreibung des Datenelements enthalten.

ii. Die INITIALIZE-Anweisung enthält die REPLACING-Angabe und die Kategorie des Datenelements ist eine der Kategorien aus der REPLACING-Angabe, oder

iii. die INITIALIZE-Anweisung enthält die DEFAULT-Angabe, oder

iv. die INITIALIZE-Anweisung enthält weder die VALUE-Angabe noch die REPLACING-Angabe.

Die jeweils erste Bedingung, die für ein Datenelement zutrifft, kommt zum Zug - die nachfolgenden werden dann für dieses Datenelement nicht mehr ausgewertet.

6. Für die Ermittlung der elementaren Sendefelder gelten die folgenden Regeln:
   a. Qualifiziert sich das Empfangsfeld aufgrund der VALUE-Angabe, so gilt:
      i. Ist das Empfangsfeld von der Kategorie datenzeiger oder programmzeiger, so ist das Sendefeld die vordefinierte Adresse NULL.
      ii. Ist das Empfangsfeld von der Kategorie objektreferenz, so ist das Sendefeld die vordefinierte Objektreferenz NULL.
      iii. In allen übrigen Fällen ist das Sendefeld das Literal, das in der VALUE-Klausel in der Definition des Empfangsfeld angegeben ist (siehe Abschnitt "VALUE-Klausel" Format 1 bzw. Format 3).
   b. Qualifiziert sich das Empfangsfeld aufgrund der REPLACING-Angabe, so ist das Sendefeld literal-1 oder bezeichner-2, die der Kategorie des Empfangsfelds in der REPLACING-Angabe zugeordnet sind.
   c. Wenn für das Empfangsfeld weder Regel a) noch b) zutreffen, hängt der Wert des Sendefelds wie folgt von der Kategorie des Empfangsfelds ab:.

Kategorie des Empfangsfelds	**Sendeoperand**
alphabetisch	Figurative Konstante alphanumerische SPACES
alphanumerisch	Figurative Konstante alphanumerische SPACES
alphanumerisch druckaufbereitet	Figurative Konstante alphanumerische SPACES
national	Figurative Konstante nationale SPACES
numerisch	Figurative Konstante ZEROES
numerisch druckaufbereitet	Figurative Konstante ZEROES
datenzeiger	Vordefinierte Adresse NULL
programmzeiger	Vordefinierte Adresse NULL
objektreferenz	Vordefinierte Objektreferenz NULL

7. Die durch bezeichner-1 dargestellten Datenelemente werden in der Reihenfolge ihres Auftretens (von links nach rechts) in der INITIALIZE-Anweisung mit den angegebenen Werten vorbelegt. Wenn bezeichner-1 eine Gruppe repräsentiert, werden innerhalb dieser Gruppe die betroffenen Datenelemente in der Reihenfolge ihrer Definition in der Gruppe initialisiert.
8. Wenn bezeichner-1 und bezeichner-2 denselben Speicherplatz belegen, ist das Ergebnis der Ausführung dieser Anweisung undefiniert (siehe „Überlappende Operanden").

Beispiel 8-46

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INIT1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
```

```
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LOHNSATZ.
    02  NAME       PIC X(30).
    02  VORNAME    PIC X(30).
    02  ADRESSE    PIC X(30).
02  GEBURTSDATUM.
    03
        TAG    PIC 99.
    03
        MONAT  PIC 99.
    03
        JAHR   PIC 99.
02  EINSTELLDATUM.
    03
        TAG     PIC 99.
    03
        MONAT   PIC 99.
    03
        JAHR    PIC 99.
02  STUNDENZAHL   PIC 9(3).
02  STUNDEN-SATZ  PIC 9(2)V99.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
INITIALIZE LOHNSATZ.
DISPLAY LOHNSATZ UPON T.
STOP RUN.
```

Die Anweisung INITIALIZE LOHNSATZ bedeutet:

```
MOVE SPACE TO NAME VORNAME ADDRESSE
MOVE ZERO TO TAG OF GEBURTSDATUM
             MONAT OF GEBURTSDATUM
             JAHR OF GEBURTSDATUM
             TAG OF EINSTELLDATUM
             MONAT OF EINSTELLDATUM
             JAHR OF EINSTELLDATUM
             STUNDENZAHL, STUNDEN-SATZ
```

Beispiel 8-47

```
01 Datenstruktur.
   02 alfaFeld        PIC X(20).
   02 numFeld         PIC 9(15).
   02 zeiger          POINTER.
   02 FILLER          PIC 9(5).99.
INITIALIZE Datenstruktur FILLER
      REPLACING ALPHANUMERIC BY HIGH VALUE
                NUMERIC BY 5
      DEFAULT
```

Diese INITIALIZE-Anweisung bedeutet:

```
MOVE HIGH-VALUE TO alfaFeld
MOVE 5 TO numFeld
SET zeiger TO NULL                  (1)
MOVE ZERO TO <FILLER-datenfeld>     (2)
```

(1) auf Grund der Default-Angabe
(2) auf Grund der FILLER-Angabe und der Default-Angabe

Beispiel 8-48

```
01 Datenstruktur.
   02 alfaFeld        PIC X(20).
   02 numFeld         PIC 9(15) value 1860.
   02 zeiger          POINTER.
   02 FILLER          PIC 9(5).99.
   02 objref          OBJECT REFERENCE.
INITIALIZE Datenstruktur FILLER
      ALL TO VALUE
      REPLACING ALPHANUMERIC BY LOW-VALUE
             NUMERIC BY 5
DEFAULT.
```

Diese INITIALIZE-Anweisung bedeutet:

```
MOVE LOW-VALUE TO alfaFeld
MOVE 1860 TO numFeld               (1)
SET zeiger TO NULL                 (2)
MOVE ZERO TO <FILLER-datenfeld>    (3)
SET objref TO NULL                 (2)
```

(1) auf Grund der VALUE-Angabe;
die REPLACING-Angabe für NUMERIC kommt in diesem Fall nicht mehr zum Zug
(2) auf Grund der VALUE-Angabe
(3) auf Grund der FILLER-Angabe und der Default-Angabe

Beispiel 8-49

```
01 Datenstruktur.
   02 alfaFeld        PIC X(20).
   02 numFeld         PIC 9(15).
   02 zeiger          POINTER.
   02 FILLER          PIC 9(5).99.
   02 objref          OBJECT REFERENCE.
INITIALIZE Datenstruktur
      DATA-POINTER TO VALUE
      REPLACING ALPHANUMERIC BY HIGH-VALUE
             NUMERIC BY 5
DEFAULT.
```

Diese INITIALIZE-Anweisung bedeutet:

```
MOVE HIGH-VALUE TO alfaFeld
MOVE 5 TO numFeld
SET zeiger TO NULL              (1)
SET objref TO NULL              (2)
```

(1) auf Grund der VALUE-Angabe;
(2) auf Grund der DEFAULT-Angabe

Beispiel 8-50

```
01 PTR USAGE POINTER.
...
INITIALIZE PTR DATA-POINTER TO VALUE
```

ist eine umständlichere Schreibweise für:

```
SET PTR TO NULL.
```

Beispiel 8-51

```
01 Datastructure VALUE "XXXX1234".
   02 alfaFeld          PIC X(4).
   02 numFeld           PIC 9(4).

INITIALIZE Data-structure ALL TO VALUE DEFAULT
```

Diese INITIALIZE-Anweisung bedeutet:

```
MOVE SPACE TO alfaFeld
MOVE ZERO TO numFeld
```

Die VALUE-Angabe in der INITIALIZE-Anweisung wird nicht wirksam, da die VALUE-Klausel auf der Ebene der Datengruppe und nicht auf der Ebene der Datenelemente angegeben ist.

8.10.26 INSPECT-Anweisung

Funktion

Mit der INSPECT-Anweisung lassen sich einzelne Zeichen oder Zeichengruppen in Datenfeldern zählen, ersetzen oder zählen und ersetzen.

Format 1 (zählen)

```
INSPECT bezeichner-1 TALLYING

{bezeichner-2 FOR { { {ALL | LEADING} {bezeichner-3 | literal-1} | CHARACTERS}
  [{BEFORE | AFTER} INITIAL {bezeichner-4 | literal-2}]... }... }...
```

Format 2 (ersetzen)

```
INSPECT bezeichner-1 REPLACING

{ CHARACTERS BY {bezeichner-5 | literal-3} [{BEFORE | AFTER} INITIAL {bezeichner-4
| literal-2}]...

 |{ALL | LEADING | FIRST} { {bezeichner-3 | literal-1} BY {bezeichner-5 | literal-
3} [{BEFORE | AFTER} INITIAL {bezeichner-4 | literal-2} ]... }...
}...
```

Format 3 (zählen und ersetzen)

```
INSPECT bezeichner-1 TALLYING

{bezeichner-2 FOR {{ {ALL | LEADING} {bezeichner-3 | literal-1} | CHARACTERS} [{
BEFORE | AFTER} INITIAL {bezeichner-4 | literal-2}]... }... }...

REPLACING

{ CHARACTERS BY {bezeichner-5 | literal-3} [{BEFORE | AFTER} INITIAL {bezeichner-4
| literal-2 }]...

 |{ALL | LEADING | FIRST} { {bezeichner-3 | literal-1} BY {identifier-5 | literal-
3} [{BEFORE | AFTER} INITIAL {bezeichner-4 | literal-2}]... }... }...
```

Format 4 (umwandeln)

```
INSPECT bezeichner-1 CONVERTING {bezeichner-6 | literal-4} TO {bezeichner-7 |
literal-5} [{BEFORE | AFTER} INITIAL {bezeichner-4 | literal-2}]...
```

Syntaxregeln für alle Formate

1. bezeichner-1 kann sowohl eine alphanumerische oder nationale Datengruppe als auch ein Datenelement bezeichnen, dass implizit oder explizit mit USAGE DISPLAY oder USAGE NATIONAL beschrieben ist.
2. bezeichner-3, ..., bezeichner-n können sowohl eine alphanumerische oder nationale Datengruppe als auch ein Datenelement bezeichnen, dass implizit oder explizit mit USAGE DISPLAY oder USAGE NATIONAL beschrieben ist.
3. literal-1, ..., literal-5 müssen nichtnumerische Literale sein. Sie dürfen keine figurativen Konstanten sein, die mit ALL beginnen.
   Sind literal-1, literal-2 oder literal-4 figurative Konstanten, dann stehen sie für ein implizites 1 Zeichen langes Datenfeld. Ist bezeichner-1 von der Klasse national, dann ist auch die Klasse der figurativen Konstanten national. Andernfalls ist die Klasse der figurativen Konstanten alphanumerisch.

4. Ist irgendein literal-1, ..., literal-5, bezeichner-1, ..., bezeichner-7 von der Klasse national, dann müssen alle von der Klasse national sein.
5. Jeder einzelnen ALL-, LEADING-, CHARACTERS-, FIRST- oder CONVERTING-Angabe darf nur eine BEFORE- und/oder AFTER-Angabe zugeordnet werden.

Syntaxregel für Format 1 und 3

6. bezeichner-2 muss ein numerisches Datenelement sein.

Syntaxregeln für Format 2 und 3

7. literal-3 bzw. bezeichner-5 müssen die gleiche Größe haben wie literal-1 bzw. bezeichner-3. Ist literal-3 eine figurative Konstante, hat es implizit die gleiche Größe wie literal-1 bzw. bezeichner-3.
8. Ist CHARACTERS angegeben, dürfen literal-3 und bezeichner-5 nur 1 Zeichen lang sein.

Syntaxregeln für Format 4

9. literal-5 bzw. bezeichner-7 müssen die gleiche Größe haben wie literal-4 bzw. bezeichner-6. Ist literal-5 eine figurative Konstante, hat es implizit die gleiche Größe wie literal-4 bzw. bezeichner-6.
10. Dasselbe Zeichen darf nur einmal in literal-4 bzw. bezeichner-6 vorkommen.

Allgemeine Regeln für alle Formate

1. Die Länge von bezeichner-1 wird wie für Sendefelder berechnet (siehe Abschnitt „OCCURS-Klausel"). Ist bezeichner-1 ein null-längiges Datenfeld:
   a. bezeichner-1 und bezeichner-2 bleiben unverändert.
   b. Die Ablaufsteuerung geht zum Ende der INSPECT-Anweisung über.
2. bezeichner-1 wird - ohne Rücksicht auf seine Datenklasse - von links nach rechts abgearbeitet.
3. Der Inhalt der Datenfelder von bezeichner-1, bezeichner-3, bezeichner-4, bezeichner-5, bezeichner-6, bezeichner-7 wird wie folgt behandelt:
   a. Ist einer dieser Bezeichner alphabetisch, alphanumerisch oder national definiert, wird sein Inhalt wie eine Zeichenkette der entsprechenden Kategorie behandelt.
   b. Ist einer dieser Bezeichner alphanumerisch druckaufbereitet, numerisch druckaufbereitet oder numerisch ohne Vorzeichen definiert, wird er so behandelt, als wäre er alphanumerisch redefiniert.

      Ist einer dieser Bezeichner numerisch mit Vorzeichen definiert, wird er so behandelt, als ob er in ein numerisches Datenfeld ohne Vorzeichen, das die gleiche Größe hat (die Vorzeichenposition nicht mitgezählt), übertragen worden wäre und dieses Feld alphanumerisch redefiniert worden wäre (siehe „MOVE-Anweisung").

      Ist bezeichner-1 ein numerisch definiertes Datenfeld mit Vorzeichen, bleibt der ursprüngliche Wert dieses Vorzeichens bis nach der Ausführung der INSPECT-Anweisung erhalten.
4. Ist einer dieser Bezeichner indiziert, wird der Indexwert für diese Felder nur einmal berechnet, und zwar unmittelbar vor der Ausführung der INSPECT-Anweisung.
5. In den Allgemeinen Regeln 5. bis 15. gilt alles, was für literal-1, literal-2, literal-3,literal-4 oder literal-5 gesagt wird, entsprechend für bezeichner-3, bezeichner-4, bezeichner-5, bezeichner-6 oder bezeichner-7.

Allgemeine Regeln für Format 1, 2 und 3

6. Während der Prüfung des Inhalts von bezeichner-1 wird jedes Vorkommen von literal-1 gezählt (Format 1) bzw. werden alle mit literal-1 übereinstimmenden Zeichen durch literal-3 ersetzt (Format 2). Ist CHARACTERS angegeben, wird das Zeichen in bezeichner-1, auf das beim jeweils aktuellen Vergleich positioniert ist, gezählt bzw. durch literal-3 ersetzt.
7. Die Operanden von TALLYING bzw. REPLACING werden von links nach rechts in der Reihenfolge abgearbeitet, in der sie in der INSPECT-Anweisung angegeben sind (Vergleichszyklus). Positioniert wird am Anfang des ersten Vergleichszyklus auf das am weitesten links stehende Zeichen von bezeichner-1.

8. Ist BEFORE bzw. AFTER nicht angegeben, läuft der Vergleich zur Ermittlung des Vorkommens von literal-1 in bezeichner-1 wie folgt ab:
   a. Das erste literal-1 wird mit einer seiner Größe entsprechenden Anzahl von aufeinanderfolgenden Zeichen von bezeichner-1 verglichen, beginnend mit dem am weitesten links stehenden Zeichen. Übereinstimmung besteht nur dann, wenn literal-1 und der entsprechende Teil von bezeichner-1 Zeichen für Zeichen übereinstimmen.
   b. Liegt keine Übereinstimmung von literal-1 mit bezeichner-1 vor, wird der Vergleich mit jedem folgenden literal-1 fortgesetzt, bis entweder Übereinstimmung festgestellt wird oder kein nächstes literal-1 mehr folgt.
   c. Wird in keinem Fall Übereinstimmung von literal-1 mit bezeichner-1 festgestellt, wird die Positionierung in bezeichner-1 um 1 Zeichen nach rechts verschoben und anschließend der Vergleichszyklus von neuem mit dem ersten literal-1 begonnen.
   d. Liegt Übereinstimmung vor, wird der Vergleichszyklus abgebrochen, bezeichner-2 wird um 1 erhöht und /oder die mit literal-1 übereinstimmenden Zeichen in bezeichner-1 werden durch literal-3 ersetzt. Anschließend wird die Positionierung in bezeichner-1 um die Anzahl der Zeichen von literal-1 nach rechts verschoben und der Vergleichszyklus von neuem mit dem ersten literal-1 begonnen.
   e. Der Vergleich wird so lange fortgesetzt, bis das am weitesten rechts stehende Zeichen von bezeichner-1 entweder das Zeichen ist, bei dem ein Vergleichszyklus anfängt, oder erfolgreich an einem Vergleich mit literal-1 teilgenommen hat.
   f. Ist ALL angegeben, gelten a) bis e) uneingeschränkt. Ist LEADING angegeben, wird das korrespondierende literal-1 auf jeden Fall in das erste Durchlaufen des Vergleichszyklus einbezogen. An den folgenden Vergleichszyklen nimmt es nur dann teil, wenn beim jeweils vorhergehenden Vergleich mit diesem literal-1 Übereinstimmung festgestellt wurde.
   Ist CHARACTERS angegeben, nimmt implizit ein 1 Zeichen langer Operand am Vergleichszyklus teil, so, als ob er als literal-1 angegeben wäre; nur findet kein Vergleich mit dem Inhalt von bezeichner-1 statt, sondern dieser Operand wird immer als übereinstimmend mit dem Zeichen von bezeichner-1 betrachtet, auf das während eines laufenden Vergleichszyklus positioniert ist.
9. Ist die BEFORE- bzw. AFTER-Angabe gemacht, wird das unter 7. Gesagte wie folgt eingeschränkt:
   a. Ist BEFORE angegeben, wird wie folgt verfahren: literal-1 bzw. (falls CHARACTERS angegeben ist) der implizite Operand nimmt nur an den Vergleichszyklen teil, die auch durchlaufen worden wären, wenn bezeichner-1 mit dem Zeichen geendet hätte, das unmittelbar vor dem ersten Vorkommen von literal-2 in bezeichner-1 steht.

      Kommt literal-2 in bezeichner-1 überhaupt nicht vor oder ist bezeichner-4 ein nulllängiges Datenfeld, läuft der Vergleich so ab, als ob BEFORE nicht angegeben worden wäre.
   b. Ist AFTER angegeben, gilt alles unter a) Gesagte sinngemäß, d.h. es wird nach literal-2 in bezeichner-1 gesucht. Ist es gefunden, wird auf das unmittelbar rechts daneben stehende Zeichen in bezeichner-1 positioniert. Von hier an nimmt literal-1 bzw. (falls CHARACTERS angegeben ist) der implizite Operand an den folgenden Vergleichzyklen teil.

      Kommt literal-2 in bezeichner-1 überhaupt nicht vor oder ist bezeichner-4 ein nulllängiges Datenfeld, nimmt literal-1 bzw. (falls CHARACTERS angegeben ist) der implizite Operand an keinem Vergleichszyklus teil.

Allgemeine Regeln für Format 1 und 3

10. Der Inhalt von bezeichner-2 wird bei der Ausführung der INSPECT-Anweisung nicht initialisiert.
11. Belegen bezeichner-1, bezeichner-3 oder bezeichner-4 denselben Speicherplatz wie bezeichner-2, dann ist das Ergebnis der Ausführung der INSPECT-Anweisung undefiniert, auch dann, wenn die betreffenden Bezeichner in derselben Datenerklärung definiert sind (siehe „Überlappende Operanden").

Allgemeine Regeln für Format 2 und 3

12. Die Wörter ALL, LEADING und FIRST, die geschrieben werden müssen, sind Adjektive, die für jede folgende BY-Angabe so lange gelten, bis das nächste Adjektiv angegeben wird.
13. Ist FIRST angegeben, wird literal-1 in bezeichner-1 nur an der Stelle durch literal-3 ersetzt, an der es zum erstenmal vorkommt. Diese Regel gilt für alle aufeinanderfolgenden FIRST-Angaben, unabhängig vom Wert von literal-1.
14. Belegen bezeichner-3, bezeichner-4 oder bezeichner-5 denselben Speicherplatz wie bezeichner-1, so ist das Resultat der Ausführung der INSPECT-Anweisung undefiniert, auch dann, wenn die betreffenden Bezeichner in derselben Datenerklärung definiert sind (siehe „Überlappende Operanden").

Allgemeine Regel für Format 3

15. Eine INSPECT-Anweisung des Formats 3 wird so ausgeführt, als handele es sich um 2 aufeinanderfolgende INSPECT-Anweisungen, die sich auf denselben bezeichner-1 beziehen, und zwar um eine Anweisung des Formats 1 (mit der TALLYING-Angabe) und eine Anweisung des Formats 2 (mit der REPLACING-Angabe).
    Entsprechend gelten die für Format 1 und 2 angegebenen Regeln. Die Subskripte eines Bezeichners in der Anweisung vom Format 2 werden nur einmal ausgewertet, bevor die Anweisung vom Format 1 ausgeführt wird.

Allgemeine Regeln für Format 4

16. Eine INSPECT-Anweisung des Formats 4 wird so interpretiert und ausgeführt, als han-dele es sich um eine INSPECT-Anweisung des Formats 2, in der eine Reihe von ALL-Angaben (für jedes einzelne Zeichen von literal-4 bzw. bezeichner-6 eine ALL-Angabe) gemacht sind, die sich auf denselben bezeichner-1 beziehen.

    Jedes Zeichen von literal-4 bzw. bezeichner-6 und das jeweils zugehörige Zeichen von literal-5 bzw. bezeichner-7 wird also so interpretiert, als wäre es ein eigenes literal-1 (ein eigener bezeichner-3) bzw. literal-3 (bezeichner-5) in Format 2.

    Die eindeutige Zuordnung der Zeichen von literal-4 (bezeichner-6) und literal-5 (bezeichner-7) ergibt sich aus ihrer jeweiligen Position innerhalb des Datenfeldes.
17. Belegen bezeichner-4, bezeichner-6 oder bezeichner-7 denselben Speicherplatz wie bezeichner-1, dann ist das Ergebnis der Ausführung der INSPECT-Anweisung undefiniert, auch dann, wenn die betreffenden Bezeichner in derselben Datenerklärung definiert sind (siehe „Überlappende Operanden").

Beispiel 8-52

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  INSP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ZAEHLER1 PIC 99 VALUE ZEROES.
01 ZAEHLER2 PIC 99 VALUE 0.
01 ZAEHLER3 PIC 99 VALUE 0.
01 FELD PIC X(20) VALUE SPACES.
   ...

PROCEDURE DIVISION.
PROC SECTION.
ZAEHLEN.
    MOVE "BBYZYZBBYZAXBXBBX" TO FELD.
    INSPECT FELD TALLYING
       ZAEHLER1 FOR ALL "X" AFTER INITIAL "A"
```

```
       ZAEHLER2 FOR LEADING "YZ" AFTER INITIAL "BB"           (1)
       ZAEHLER3 FOR CHARACTERS BEFORE INITIAL "A".
    DISPLAY "Nach INSPECT" UPON T.
    DISPLAY "ZAEHLER1 = *" ZAEHLER1 "*" UPON T.
    DISPLAY "ZAEHLER2 = *" ZAEHLER2 "*" UPON T.
    DISPLAY "ZAEHLER3 = *" ZAEHLER3 "*" UPON T.
ERSETZEN-1.
    MOVE "MR. COBOLUSER" TO FIELD.
    DISPLAY "Vor INSPECT" UPON T
    DISPLAY "FELD = *" FELD "*" UPON T.
    INSPECT FELD REPLACING
       CHARACTERS BY "X" AFTER INITIAL "MR. "
                         BEFORE INITIAL "U".
    DISPLAY "Nach INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*" UPON T.
ERSETZEN-2.
    MOVE "ALGOL-PROGRAM" TO FIELD.
    DISPLAY "Vor INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*" UPON T.
    INSPECT FELD REPLACING
       ALL "A" BY "C" BEFORE INITIAL "P"
       ALL "L" BY "O" BEFORE INITIAL "G"
       ALL "G" BY "B" BEFORE INITIAL "P".
    DISPLAY "Nach INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*" UPON T.
ERSETZEN-3.
    MOVE "XXYZYZXXYZ-XYZXYZ" TO FELD.
    DISPLAY "Vor INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*".
    INSPECT FELD REPLACING
       LEADING "YZ" BY "AB" BEFORE INITIAL "-"
                            AFTER INITIAL "XX"               (2)
       FIRST "YZ" BY "CD" AFTER INITIAL "-".
    DISPLAY "Nach INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*" UPON T.
UMWANDELN.
    MOVE "CE#CGDHDEF-CD#F" TO FELD.
    DISPLAY "Vor INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*" UPON T.
    INSPECT FELD CONVERTING                                 (3)
       "CDEF" TO "UVWU" AFTER "#"
                        BEFORE "-".
    DISPLAY "Nach INSPECT" UPON T.
    DISPLAY "FELD = *" FELD "*" UPON T.
ENDE.
    STOP RUN.
```

Ergebnis:

Nach INSPECT (1)

```
ZAEHLER1 = *03*
ZAEHLER2 = *02*
ZAEHLER3 = *06*
```

Vor INSPECT	Nach INSPECT
FELD = *MR. COBOLUSER * FELD = *ALGOL-PROGRAM * FELD = *XXYZYZXXYZ-XYZXYZ *	FELD = *MR. XXXXXUSER * FELD = *COBOL-PROGRAM * FELD = *XXABABXXYZ-XCDXYZ *
FELD = *CE#CGDHDEF-CD#F *	FELD = *CE#UGVHVWU-CD#F *

Erläuterung

(1) Bei ZAEHLER2 wurden die im Folgenden unterstrichenen „YZ“ von FELD gezählt:

BBYZYZBBYZAXBXBBX

Es lag also bei jedem der unterstrichenen „YZ“ eine Übereinstimmung im Sinne von Allgemeine Regel 7f) vor. Deswegen begann der Vergleichszyklus beide Male wieder mit dem ersten literal-1, d.h. mit ZAEHLER1.

Daraus ergibt sich Folgendes:

ZAEHLER3 wird im Fall der unterstrichenen „YZ“ nicht erhöht. Aus diesem Grund werden nur die folgenden unterstrichenen Zeichen gezählt:

BBYZYZBBYZAXBXBBX

ZAEHLER3 ist also = 6.

(2) Das Ersetzen von führenden „YZ“ durch „AB“ wird veranlasst durch die Angabe AFTER INITIAL "XX". BEFORE INITIAL "-" hat keine Wirkung, da aufgrund der LEADING-Angabe jede Zeichenfolge ungleich „YZ“ weitere Ersetzungen verhindert.

(3) Diese INSPECT-Anweisung hat die gleiche Wirkung wie die folgende Anweisung:

```
INSPECT FELD REPLACING
    ALL "C" BY "U" AFTER "#" BEFORE "-"
    ALL "D" BY "V" AFTER "#" BEFORE "-"
    ALL "E" BY "W" AFTER "#" BEFORE "-"
    ALL "F" BY "U" AFTER "#" BEFORE "-".
```

8.10.27 INVOKE-Anweisung

Funktion

Mit der INVOKE-Anweisung wird eine Methode aufgerufen.

Format

```
INVOKE bezeichner-1 {bezeichner-2 | literal-1}
       [USING {  [BY REFERENCE] {bezeichner-3 | OMITTED}
               | [BY CONTENT] {bezeichner-5 | arithmetischer-ausdruck-1 | literal-2}
               | [BY VALUE] {bezeichner-5 | arithmetischer-ausdruck-1 | literal-2}
              }... ]
[RETURNING bezeichner-4]
[ON EXCEPTION unbedingte-anweisung-1]
[NOT ON EXCEPTION unbedingte-anweisung-2]
[END-INVOKE]
```

Syntaxregeln

1. bezeichner-1 muss eine Objektreferenz oder ein Klassenname sein.
2. literal-1 muss ein alphanumerisches Literal sein. Es darf aber keine figurative Konstante sein.
3. Falls bezeichner-1 eine universelle Objektreferenz ist, darf weder die BY CONTENT noch die BY VALUE-Angabe spezifiziert sein. Die Angabe BY REFERENCE wird, sofern sie nicht angegeben ist, implizit angenommen.
4. bezeichner-2 muss ein alphanumerisches Datenelement sein.
5. bezeichner-3 muss ein Datenelement sein, das in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION definiert ist.
6. Wird bezeichner-5 oder sein entsprechender formaler Parameter (in der PROCEDURE DIVISION-Überschrift) mit der BY VALUE-Angabe spezifiziert, so darf er nur von der Klasse numerisch, objekt oder zeiger sein.
7. bezeichner-4 muss ein Datenelement sein, das in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION definiert ist.
8. Falls bezeichner-2 angegeben ist, muss bezeichner-1 eine universelle Objektreferenz sein.
9. Ist bezeichner-1 keine universelle Objektreferenz ist, gelten die Konformitätsregeln für Parameter und Rückgabeelemente.
10. Ist bezeichner-1 keine universelle Objektreferenz und ist die Angabe BY CONTENT oder BY REFERENCE für ein Argument spezifiziert, dann muss BY REFERENCE auch für den entsprechenden formalen Parameter in der PROCEDURE DIVISION-Überschrift angegeben werden.
11. BY CONTENT darf nicht weggelassen werden, wenn bezeichner-5 als Empfangsfeld zugelassen ist.
12. Ist die Angabe BY VALUE für ein Argument spezifiziert, so muss sie auch für den entsprechenden formalen Parameter in der PROCEDURE DIVISION-Überschrift angegeben sein.
13. Ist OMITTED angegeben, so muss die Angabe OPTIONAL für den entsprechenden formalen Parameter in der PROCEDURE DIVISION-Überschrift spezifiziert sein.
14. Wenn bezeichner-1 eine universelle Objektreferenz ist, dürfen die formalen Parameter sowie das formale Rückgabe-Element nicht mit der ANY LENGTH-Klausel definiert sein.

15. Wenn bezeichner-3 oder bezeichner-4 mit der ANY LENGTH-Klausel definiert ist, muss der entsprechende formale Parameter ebenfalls mit der ANY LENGTH-Klausel definiert sein. Wenn bezeichner-5 mit der ANY LENGTH-Klausel definiert ist, darf der entsprechende formale Parameter **nicht** mit der ANY LENGTH-Klausel definiert sein.
16. Wenn der mit bezeichner-5 korrespondierende formale Parameter mit der ANY LENGTH-Klausel definiert ist, muss bezeichner-5 von der Klasse alphabetisch oder alphanumerisch sein.
17. bezeichner-5 und jeder in arithmetischer-ausdruck-1 spezifizierte Bezeichner ist ein sendender Operand.
18. bezeichner-4 ist ein Empfangsfeld.
19. bezeichner-3 muss auf einen Adressbezeichner oder ein Datenelement verweisen, der in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE-SECTION definiert ist.
20. Wenn die BY REFERENCE-Angabe für bezeichner-3, der kein Adressbezeichner ist, spezifiziert oder impliziert ist, stellt bezeichner-3 sowohl ein Sende-, als auch Empfangsfeld dar. Andernfalls spezifiziert bezeichner-3 ein Sendefeld.

    Anmerkung:
    Die Angabe BY REFERENCE ADDRESS OF datenname wird behandelt wie BY CONTENT ADDRESS OF datenname.

Allgemeine Regeln

1. bezeichner-1 kennzeichnet eine Objektinstanz. Falls bezeichner-1 ein Klassenname ist, steht er für das Fabrik-Objekt dieser Klasse. Literal-1 oder der Inhalt von bezeichner-2 kennzeichnet eine Methode, von der diese Objektinstanz bearbeiten wird.
   literal-1 oder der Inhalt von bezeichner-2 ist der Name der gerufenen Methode, wie er im zugehörigen METHOD-ID Paragraf angegeben ist (als COBOL-Wort).
2. Die Reihenfolge der Argumente in der USING-Angabe der INVOKE-Anweisung und die entsprechenden formalen Parameter in der USING-Angabe der PROCEDURE DIVISION-Überschrift der aufgerufenen Methode bestimmen die Zuordnung von Argumenten zu formalen Parametern. Diese Zuordnung ist positionsabhängig und unabhängig von Namen. Das erste Argument entspricht also dem ersten formalen Parameter, das zweite dem zweiten und das n-te dem n-ten Parameter.
3. Ein Argument, das lediglich aus einem einzelnen Bezeichner oder Literal besteht, wird als Bezeichner oder Literal betrachtet und nicht als arithmetischer Ausdruck.
4. Falls bezeichner-1 Null ist, entsteht die Ausnahmesituation EC-OO-NULL. Es wird keine Methode aktiviert und die Ausführung erfolgt wie in Absatz 6f.
5. Falls bezeichner-1 keine universelle Objektreferenz ist und ein Argument ohne eine der Angaben BY REFERENCE, BY CONTENT oder BY VALUE spezifiziert ist, wird dieses Argument wie folgt behandelt:
    a. Es wird BY REFERENCE angenommen, wenn die BY REFERENCE-Angabe für den entsprechenden formalen Parameter spezifiziert oder implizit gegeben ist und das Argument ein Bezeichner ist, der als Empfangsfeld zulässig ist.
    b. Es wird BY CONTENT angenommen, wenn die BY REFERENCE-Angabe für den entsprechenden formalen Parameter spezifiziert oder implizit gegeben ist und das Argument ein Literal , ein arithmetischer Ausdruck oder irgendein anderer Bezeichner ist, der nicht als Empfangsfeld zulässig ist.
    c. Es wird BY VALUE angenommen, wenn die BY VALUE-Angabe für den entsprechenden formalen Parameter spezifiziert ist.
6. Die Ausführung der INVOKE-Anweisung läuft wie folgt ab:
    a. Zuerst werden arithmetischer-ausdruck-1, bezeichner-1, bezeichner-2, bezeichner-3 und bezeichner-5 berechnet. Weiter wird am Anfang der Ausführung der INVOKE-Anweisung die Element-Identifizierung für bezeichner-4 vorgenommen. Falls dabei eine Ausnahmebedingung entsteht, wird keine Methode aufgerufen und die Ausführung verläuft wie in Absatz 6f beschrieben.

b. Das Laufzeitsystem versucht die aufgerufene Methode zu finden. Kann die Methode nicht gefunden werden oder sind die Betriebsmittel, die zur Ausführung der Methode erforderlich sind, nicht verfügbar, entsteht die Ausnahmesituation EC-OO-METHOD. Die Methode wird nicht aktiviert und die Ausführung erfolgt wie in Abschnitt 6f angegeben.

c. Ist bezeichner-1 eine universelle Objektreferenz müssen die Konformitätsregeln für Parameter und Rückgabeelemente erfüllt sein. Liegt eine Verletzung dieser Regeln vor, entsteht die Ausnahmesituation EC-OO-UNIVERSAL, die Methode wird nicht aktiviert und die Ausführung erfolgt wie in Abschnitt 6f beschrieben.

d. Die in der INVOKE-Anweisung angegebene Methode wird zur Ausführung bereitgestellt und die Steuerung wird an diese aufgerufene Methode übergeben (konform den Aufrufkonventionen).

e. Nachdem die aufgerufene Methode die Steuerung zurückgegeben hat, wird, falls vorhanden, unbedingte-anweisung-2 ausgeführt und an das Ende der INVOKE-Anweisung verzweigt.

f. Ist eine der Ausnahmesituationen EC-OO-NULL, EC-OO-METHOD oder EC-OO-UNIVERSAL entstanden, so ergibt sich folgender Ablauf:

- Ist ON EXCEPTION angegeben, so wird zu unbedingte-anweisung-1 und anschließend an das Ende der INVOKE-Anweisung verzweigt. Der zugehörige Ausnahmezustand wird nicht ausgelöst.
- Ist ON EXCEPTION nicht angegeben und die Überprüfung der Ausnahmesituation eingeschaltet, so wird der zugehörige Ausnahmezustand ausgelöst und in die entsprechende USE-Prozedur verzweigt. Nach Ausführung von RESUME NEXT StATEMENT in der USE-Prozedur wird unbedingte-anweisung-2 ignoriert und bei der nächsten ausführbaren Anweisung nach Ende der INVOKE-Anweisung fortgesetzt.
- Ist ON EXCEPTION nicht angegeben, die Überprüfung der Ausnahmesituation nicht eingeschaltet, aber NOT ON EXCEPTION angegeben, so wird unbedingte-anweisung-2 ignoriert und bei der nächsten ausführbaren Anweisung nach Ende der INVOKE-Anweisung fortgesetzt.
- Ist weder ON EXCEPTION noch NOT ON EXCEPTION angegeben und die Überprüfung der Ausnahmesituation nicht eingeschaltet, wird der Programmlauf abgebrochen.

Ist eine andere Ausnahmesituation als EC-OO-NULL, EC-OO-METHOD oder EC-OO-UNIVERSAL entstanden (z.B. EC-OO-CONFORMANCE), und ist die Überprüfung der Ausnahmesituation eingeschaltet, so wird der zugehörige Ausnahmezustand ausgelöst und in die entsprechende USE-Prozedur verzweigt. Nach Rücksprung aus der USE-Prozedur wird unbedingte-anweisung-1 und unbedingte-anweisung-2 ignoriert und bei der nächsten ausführbaren Anweisung nach Ende der INVOKE-Anweisung fortgesetzt. Ist die Überprüfung der Ausnahmesituation nicht eingeschaltet, so wird der Programmlauf abgebrochen.

7. Ist eine RETURNING-Angabe spezifiziert, wird das Ergebnis der aufgerufenen Methode in bezeichner-4 abgelegt.
8. Ist eine OMITTED-Angabe spezifiziert oder ein Argument ganz weggelassen, dann ist die Omitted-Argument-Bedingung für diesen Parameter in der aufgerufenen Methode wahr.
9. Ist für einen Parameter die Omitted-Argument-Bedingung wahr und wird darauf in einer aufgerufenen Methode (außer in einer Omitted-Argument Bedingung) Bezug genommen, so ist das Verhalten undefiniert.

8.10.28 MERGE-Anweisung

Funktion

Die MERGE-Anweisung generiert eine Sortierdatei, in die sie Datensätze aus zwei oder mehreren gleichartig sortierten Eingabedateien übernimmt. Sie mischt die Datensätze in der Sortierdatei anhand einer Anzahl von angegebenen Datenfeldern (Sortierschlüssel) und stellt nach Ende des Mischvorgangs jeden Datensatz aus der Sortierdatei einer Ausgabeprozedur oder Ausgabedateien zur Verfügung.

Format

```
MERGE sortierdateiname

    {ON {DESCENDING | ASCENDING} {KEY | KEY-YY } {datenname-1}... }...

    [COLLATING SEQUENCE IS alphabetname]

    USING {dateiname-1}...

    { OUTPUT PROCEDURE IS kapitelname-1 [{THRU | THROUGH} kapitelname-2]

     |GIVING {dateiname-2}...

    }
```

Syntaxregeln

1. sortierdateiname muss in einer Sortierdateierklärung (SD) in der DATA DIVISION beschrieben sein.
2. sortierdateiname muss mit dem Sortierdateinamen übereinstimmen, der in der SELECT-Klausel (Format 2) angegeben ist.
3. In der USING-Angabe angegebene Dateinamen dürfen nicht in der GIVING-Angabe angegeben werden.
4. datenname-1... sind Sortierschlüssel. Ein Sortierschlüssel ist ein Teil des Datensatzes, der als Sortiergrundlage verwendet wird. Die Sortierschlüssel müssen in einer Datensatzbeschreibung, die zu einer SD-Erklärung gehört, definiert sein und unterliegen folgenden Regeln:
   a. Die Länge der Datenfelder darf nicht variabel sein.
   b. Datennamen, die die Sortierschlüssel beschreiben, dürfen gekennzeichnet sein (siehe „Kennzeichnung").
   c. Enthält die Sortierdateierklärung von sortierdateiname mehrere Datensatzbeschreibungen, brauchen die Schlüssel nur in einer Datensatzbeschreibung definiert sein. Ist ein Sortierschlüssel in mehr als einer Datensatzbeschreibung definiert, müssen alle Beschreibungen des Schlüssels gleich sein und gewährleisten, dass der Schlüssel innerhalb eines jeden Datensatzes an der gleichen Stelle auftritt.
   d. Ein Sortierschlüssel darf nicht mit einer OCCURS-Klausel beschrieben und nicht einem Datenfeld untergeordnet sein, das mit einer OCCURS-Klausel beschrieben ist.
   e. Enthält die Sortierdatei Sätze variabler Länge, müssen alle Sortierschlüssel in den ersten n Stellen des Satzes enthalten sein, wobei n die minimale Satzlänge ist, die für die Sortierdatei angegeben wurde.
   f. Für eine Datei können maximal 64 Schlüssel angegeben werden.
   g. Jeder Sortierschlüssel muss innerhalb der ersten 4096 Bytes liegen.
   h. Sortierschlüssel werden in der Reihenfolge ihrer Wichtigkeit für die Sortierung von links nach rechts angegeben, unabhängig davon, ob sie auf- oder absteigend sind. Die erste Angabe von datenname-1 wäre somit der Hauptsortierbegriff, die zweite Angabe von datenname-1 der nachgeschaltete Sortierbegriff.
   i. Ein dezimal gepackter Sortierschlüssel darf maximal 16 Stellen lang sein.
   j. Die Sortierschlüssel, die der Angabe KEY-YY folgen, müssen entweder mit PIC 99 USAGE DISPLAY oder USAGE PACKED-DECIMAL definiert sein.

5. kapitelname-1 ist der Name des ersten oder einzigen Kapitels der verwendeten Ausgabeprozedur. kapitelname-2 ist der Name des letzten Kapitels der Ausgabeprozedur. Er muss nur dann angegeben werden, wenn die Ausgabeprozedur aus mehreren Kapiteln besteht.
6. dateiname-1..., dateiname-2... müssen in einer Dateierklärung (FD) der DATA DIVISION beschrieben sein.
7. Die Größe der Datensätze, die an eine MERGE-Operation übergeben bzw. in eine Ausgabedatei geschrieben werden, ist abhängig vom Satzformat der Sortierdatei (variabel oder fest) und wird in den Allgemeinen Regeln unter USING-/GIVING-Angabe näher beschrieben.
8. Mindestens eine ASCENDING-/DESCENDING-Angabe muss in einer MERGE-Anweisung angegeben sein.
9. Die MERGE-Anweisung darf überall im Programm geschrieben werden außer
   a. im Bereich der Prozedurvereinbarungen,
   b. in einer zu einer SORT-/MERGE-Anweisung gehörenden Ein-/Ausgabe-Prozedur.
10. Die in der MERGE-Anweisung genannten Sortier- und Eingabedateien dürfen nicht zusammen in derselben SAME AREA, SAME SORT AREA oder SAME SORT-MERGE AREA angegeben sein (siehe „SAME AREA-Klausel").
11. Falls dateiname-2 eine indizierte Datei beschreibt, muss die erste Angabe von datenname-1 mit der ASCENDING-Angabe erfolgen. Das durch datenname-1 referenzierte Datenfeld muss in seinem Datensatz dieselben Zeichenstellen belegen wie der Satzschlüssel innerhalb der durch dateiname-2 beschriebenen Datei.

Allgemeine Regeln

1. Die Sortierreihenfolge wird durch die ASCENDING-/DESCENDING-Angabe festgelegt:
   a. Bei Angabe von ASCENDING wird vom niedrigsten zum höchsten Wert des Sortierschlüssels sortiert (aufsteigende Sortierreihenfolge).
   b. Bei Angabe von DESCENDING wird vom höchsten zum niedrigsten Wert des Sortierschlüssels sortiert (absteigende Sortierreihenfolge).

      Für die Sortierreihenfolge gelten die Regeln für den Vergleich von Operanden in „Vergleichsbedingungen“ (siehe Abschnitt „Vergleichsbedingung").
2. Sind - in Übereinstimmung mit den Regeln für Vergleichsbedingungen - die Schlüssel-Datenfelder von Sätzen innerhalb einer oder mehrerer Dateien inhaltsgleich, werden diese Sätze folgendermaßen verarbeitet:
   a. Die Eingabedateien werden in der Reihenfolge verarbeitet, die in der MERGE-Anweisung festgelegt wurde.
   b. In der Ausgabedatei werden zuerst alle Sätze der ersten Eingabedatei aufgeführt, dann die der zweiten Eingabedatei usw.
3. Beim Programmablauf ist die Sortierfolge für die Vergleiche von alphanumerischen Sortierfeldern wie folgt festgelegt:
   a. ist COLLATING SEQUENCE in der MERGE-Anweisung angegeben, ist diese Angabe Kriterium für die Sortierfolge.
   b. ist COLLATING SEQUENCE in der MERGE-Anweisung nicht angegeben, wird die programmspezifische Sortierfolge verwendet (siehe "OBJECT-COMPUTER-Paragraf").

   Für Vergleiche von nationalen Sortierfeldern wird die nationale (maschineneigene) Sortierfolge verwendet.
4. Alle Datensätze der Eingabedateien (dateiname-1...) werden in die mit sortierdatei-name bezeichnete Sortierdatei übertragen. Bei Beginn der Ausführung der MERGE-Anweisung dürfen die Eingabedateien nicht geöffnet sein. Die MERGE-Anweisung wird für jede genannte Datei in folgenden Schritten ausgeführt:
   a. Die Verarbeitung der Datei wird eingeleitet. Dies geschieht so, als ob eine OPENINPUT-Anweisung ausgeführt würde.

b. Jeder logische Datensatz wird für den Mischvorgang zur Verfügung gestellt und anschließend freigegeben, als ob eine READ-Anweisung mit NEXT RECORD- und AT END-Angabe ausgeführt würde. Enthält eine Eingabedatei die RECORD-Klausel mit DEPENDING-Angabe, wird das zugehörige DEPENDING ON-Datenfeld bei dieser READ-Operation nicht versorgt. Relative Dateien müssen im FILE-CONTROL-Paragrafen mit ACCESS MODE IS SEQUENTIAL beschrieben sein.
Hat die Sortierdatei Sätze variabler Länge, wird als Satzlänge eines Datensatzes, der an die MERGE-Operation übergeben wurde, die Größe verwendet, die der Satz beim Einlesen hatte. Diese Länge muss in dem Bereich liegen, der in der RECORD-Klausel für die Sortierdatei festgelegt ist (siehe „RECORD-Klausel").
Hat die Sortierdatei festes Satzformat, werden bei kürzeren Sätzen die fehlenden Stellen mit Blanks aufgefüllt, längere Sätze dürfen nicht vorkommen.

c. Die Verarbeitung der Datei wird beendet. Dies geschieht so, als ob eine CLOSE-Anweisung ohne Wahlangaben ausgeführt würde.

Diese impliziten Funktionen werden so durchgeführt, dass alle vereinbarten USE AFTER EXCEPTION / ERROR-Prozeduren vollzogen sind.

5. OUTPUT PROCEDURE bedeutet, dass der Prozedurteil eine Ausgabeprozedur enthält, in der Datensätze nach dem Mischen bearbeitet werden. Ist OUTPUT
PROCEDURE angegeben, wird die Steuerung dann an die Ausgabeprozedur übergeben, nachdem die Sortierdatei durch die MERGE-Anweisung bearbeitet worden ist. Beim Durchlaufen einer RETURN-Anweisung übernimmt die Ausgabeprozedur die Datensätze aus der Sortierdatei. Der Compiler fügt einen Rückkehrmechanismus am Ende des letzten Kapitels der Ausgabeprozedur ein, d.h., ist die letzte Anweisung der Ausgabeprozedur durchlaufen, wird die Ausgabeprozedur abgeschlossen und die Steuerung geht an die der MERGE-Anweisung folgende Anweisung über.

   Für die Ausgabeprozedur, die ein abgeschlossener Teil innerhalb des Prozedurteils ist, gelten folgende Regeln:

   a. Sie muss aus einem oder mehreren zusammenhängenden Kapiteln bestehen.
   b. Sie muss mindestens eine RETURN-Anweisung enthalten, damit die gemischten Datensätze für die Verarbeitung verfügbar sind.
   c. Sie darf nicht zur Ausführung einer MERGE-, RELEASE- oder SORT-Anweisung führen.
   d. Sie kann jede Prozedur enthalten, die erforderlich ist, um Datensätze auszuwählen, zu verändern oder zu kopieren.
   e. Die Ausgabeprozedur darf verlassen werden, falls der Benutzer dafür sorgt, dass einem Sprung aus der Ausgabeprozedur ein Rücksprung folgt, um ein ordnungsgemäßes Verlassen dieser Prozedur zu gewährleisten (Durchlaufen der letzten Anweisung der Ausgabeprozedur).
   f. Von außerhalb darf auf Prozedurnamen in der Ausgabeprozedur gesprungen werden, falls dabei keine RETURN-Anweisung oder das Ende der Ausgabeprozedur durchlaufen wird.

6. Wird die Angabe OUTPUT PROCEDURE verwendet, wird eine Programmverzweigung entsprechend einer PERFORM-Anweisung, Format 1, ausgeführt. Das bedeutet, dass alle Kapitel, die die Prozedur bilden, einmal durchlaufen werden und die Ausführung der Prozedur nach Verarbeitung der letzten Anweisung beendet wird. Deshalb darf jede Prozedur durch eine EXIT-Anweisung abgeschlossen werden.
7. Kommen MERGE-Anweisungen in segmentierten Programmen vor, gelten folgende Einschränkungen:
   a. Falls eine MERGE-Anweisung in einem Kapitel auftritt, das nicht in einem unabhän-gigen Segment liegt, so müssen alle Ausgabeprozeduren, auf die sich die MERGE-Anweisung bezieht entweder vollständig innerhalb eines festen Segments oder vollständig in einem einzelnen unabhängigen Segment enthalten sein.
   b. Wenn eine MERGE-Anweisung in einem unabhängigen Segment vorkommt, so müssen alle Ausgabeprozeduren, auf die sich die MERGE-Anweisung bezieht entweder vollständig innerhalb eines festen Segments oder vollständig innerhalb des gleichen unabhängigen Segments wie die MERGE-Anweisung enthalten sein.

Diese Einschränkungen gelten nicht für den in diesem Handbuch beschriebenen Compiler.

8. Ist die GIVING-Angabe vereinbart, werden alle gemischten Sätze in die Ausgabedatei (dateiname-3...) geschrieben. Bei Beginn der Ausführung der MERGE-Anweisung darf die Ausgabedatei nicht geöffnet sein. Die MERGE-Anweisung wird für jede genannte Datei in folgenden Schritten ausgeführt:
   a. Die Verarbeitung der Datei wird eingeleitet. Dies geschieht so, als ob eine OPEN OUTPUT-Angabe ausgeführt würde.
   b. Die gemischten logischen Datensätze werden verarbeitet und in die Ausgabedatei geschrieben, als ob eine WRITE-Anweisung ohne jede Wahlangabe ausgeführt würde. Die Länge dieser Datensätze muss innerhalb der Grenzen liegen, die für die Ausgabedatei festgelegt sind (siehe „RECORD-Klausel").
   Bei einer relativen Datei enthält das Satzschlüsselfeld des ersten verarbeiteten Satzes den Wert 1, das des zweiten verarbeiteten Satzes den Wert 2 usw. Nach der Ausführung der MERGE-Anweisung zeigt der Inhalt des Satzschlüsselfeldes auf den letzten verarbeiteten Satz der Datei. Die Datei muss im FILE-CONTROL-Paragrafen mit ACCESS MODE IS SEQUENTIAL beschrieben sein.
   c. Die Verarbeitung der Datei wird beendet. Dies geschieht so, als ob eine CLOSE-Anweisung ohne Wahlangaben ausgeführt würde. Enthält die Ausgabedatei die RECORD-Klausel mit DEPENDING ON-Angabe, wird beim Schreiben eines Satzes das zugehörige DEPENDING ON-Datenfeld nicht ausgewertet, sondern die aktuelle Satzlänge verwendet.

Beispiel 8-53

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MISCH1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT EINGABE1 ASSIGN TO "EINGABE1".
   SELECT EINGABE2 ASSIGN TO "EINGABE2".
   SELECT AUSGABE1 ASSIGN TO "AUSGABE1".
   SELECT AUSGABE2 ASSIGN TO "AUSGABE2".
   SELECT SORTIER ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD EINGABE1 LABEL RECORD STANDARD.
01 ESATZ1.
   02  E10          PIC X.
   02  E11          PIC 9(4).
   02  E12          PIC 9(4).
   02  E13          PIC 9(4).
FD EINGABE2 LABEL RECORD STANDARD.
01 ESATZ2.
   02  E20          PIC X.
   02  E21          PIC 9(4).
   02  E22          PIC 9(4).
   02  E23          PIC 9(4).
FD AUSGABE1 LABEL RECORD STANDARD.
01 A1SATZ PIC X(13).
FD AUSGABE2 LABEL RECORD STANDARD.
01 A2SATZ PIC X(13).
SD SORTIER LABEL RECORD STANDARD.
01 SSATZ.
   02  S0           PIC X.
   02  S1           PIC 9(4).
   02  S2           PIC 9(4).
   02  S3           PIC 9(4).
```

```
PROCEDURE DIVISION.
HAUPT SECTION.
H01.
   MERGE SORTIER ON ASCENDING S1 S2 S3    (1)
   USING EINGABE1 EINGABE2                 |
   GIVING AUSGABE1 AUSGABE2.              (1)
H02.
   STOP RUN.
```

(1) Die Sätze von zwei gleichartig sortierten Dateien werden in zwei identische Dateien in sortierter Reihenfolge ausgegeben.

Alle Dateien sind aufsteigend nach den Sortierbegriffen S1 S2 S3 sortiert.

Beispiel 8-54

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MISCH2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EINGABE1 ASSIGN TO "EINGABE1".
    SELECT EINGABE2 ASSIGN TO "EINGABE2".
    SELECT AUSGABE1 ASSIGN TO "AUSGABE1".
    SELECT AUSGABE2 ASSIGN TO "AUSGABE2".
    SELECT SORTIER ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD EINGABE1 LABEL RECORD STANDARD.
01 ESATZ1.
   02  E10          PIC X.
   02  E11          PIC 9(4).
   02  E12          PIC 9(4).
   02  E13          PIC 9(4).
FD EINGABE2 LABEL RECORD STANDARD.
01 ESATZ2.
   02  E20          PIC X.
   02  E21          PIC 9(4).
   02  E22          PIC 9(4).
   02  E23          PIC 9(4).
FD AUSGABE1 LABEL RECORD STANDARD.
01 A1SATZ PIC X(13).
FD AUSGABE2 LABEL RECORD STANDARD.
01 A2SATZ PIC X(13).
SD SORTIER LABEL RECORD STANDARD.
01 SSATZ.
   02  S0           PIC X.
   02  S1           PIC 9(4).
   02  S2           PIC 9(4).
   02  S3           PIC 9(4).
WORKING-STORAGE SECTION.
01 MISCH-STATUS PIC X VALUE LOW-VALUE.
    88  MISCH-ENDE VALUE HIGH-VALUE.
PROCEDURE DIVISION.
HAUPT SECTION.
H01.
```

```
    OPEN OUTPUT AUSGABE1 AUSGABE2.
H02.
    MERGE SORTIER ON ASCENDING S1 S2 S3    (1)
    USING EINGABE1 EINGABE2                 |
    OUTPUT PROCEDURE IS AUS1.              (1)
H03.
    CLOSE AUSGABE1 AUSGABE2.
    STOP RUN.
           .
           .
           .
AUS1 SECTION.
A01.
    PERFORM UNTIL MISCH-ENDE               (2)
      RETURN SORTIER                        |
      AT END                                |
        SET MISCH-END TO TRUE               |
      NOT AT END                            |
        WRITE A1SATZ FROM SSATZ             |
        WRITE A2SATZ FROM SSATZ            (2)
      END-RETURN
    END-PERFORM.
```

(1) Die Sätze aus den gleichartig sortierten Dateien werden in sortierter Reihenfolge von der Sortierdatei übernommen. Dann geht die Steuerung an die Ausgabeproze dur (AUS1 SECTION) über.

(2) Die Sortierdatei wird satzweise übernommen und in die Ausgabedateien geschrieben.

8.10.29 MOVE-Anweisung

Funktion

Die MOVE-Anweisung überträgt Daten von einem Datenfeld in ein oder mehrere andere Datenfelder.

Format 1	der MOVE-Anweisung überträgt ein Datenfeld in ein oder mehrere andere Datenfelder.
Format 2	der MOVE-Anweisung überträgt einander entsprechende Datenfelder von einer Gruppe zu einer anderen.

Format 1

```
MOVE {bezeichner-1 | literal} TO {bezeichner-2}...
```

Syntaxregeln

1. Ein Indexdatenfeld, ein Zeigerdatenfeld oder ein Datenfeld der Klasse objekt darf nicht als Operand in einer MOVE-Anweisung auftreten.
2. bezeichner-2 darf kein Gruppendatenfeld sein, das Datenfelder der Klasse objekt oder zeiger enthält. Ist bezeichner-2 ein stark typisiertes Gruppendatenfeld, so muss bezeichner-1 als Gruppendatenfeld des gleichen Typs beschrieben sein.
3. bezeichner-1 bzw. literal sind Sendefelder.
4. bezeichner-2 ist ein Empfangsfeld.
5. Die figurative Konstante SPACE darf nicht in numerische oder numerisch druckaufbereitete Datenelemente übertragen werden.
6. Die figurative Konstante ZERO darf nicht in alphabetische Datenelemente übertragen werden.
7. Tabelle 28 zeigt die in einer MOVE-Anweisung zulässigen Übertragungen und deren Art an.

Sendefeld \ **Empfangsfeld**		Gruppe	Alphabetisch	Alphanumerisch	National	Extern dezimal, Binär, Intern dezimal, Numerisch druckaufbereitet	Alphanumerisch druckaufbereitet	Extern Gleitpunkt, Intern Gleitpunkt,	Stark typisiert
Gruppe		nn0	nn0	nn0	-	nn0	nn0	nn0	-
Alphabetisch		nn0	nn	nn	nn1	-	nn	-	-
Alphanumerisch, Alphanumerisches Literal		nn0	nn	nn	nn1	nu2	nn	nu2	-
National, Nationales Literal		-	-	-	nn	-	-	-	-
Extern dezimal, Binär, Intern dezimal, Numerisches Literal	ganzzahlig	nn0	-	nn	nn1	nu	nn	nu	-
	nicht ganzzahlig	nn0	-	-	-	nu	-	nu	-
Numerisch druckaufbereitet		nn0	-	nn	nn1	nu1	nn	nu1	-
Alphanumerisch druckaufbereitet		nn0	nn	nn	nn1	-	nn	-	-

Extern Gleitpunkt	nn0	-	-	-	nu	-	nu	-
Intern Gleitpunkt, Gleitpunktliteral	nn0	-	-	-	nu3	-	nu	-
Stark typisiert	nn0	nn0	nn0	-	nn0	nn0	nn0	nn0[1)]

Tabelle 28: Zulässige Übertragungen bei Element- und Gruppenübertragungen [2)]

[1)] nur für Empfangsfeld vom gleichen Typ

[2)] Tabelleneinträge:

--	=	Übertragung ist nicht erlaubt
nn	=	nichtnumerische Übertragung für Datenelemente
nn0	=	nichtnumerische Übertragung für Gruppen (ohne Konvertierung, ohne Editierung, ohne Deeditierung)
nn1	=	nichtnumerische Übertragung mit Konvertierung der Darstellung jedes Zeichens von EBCDIC zu UTF-16
nu	=	numerische Übertragung
nu1	=	numerische Übertragung nach Deeditierung des Sendefeldes
nu2	=	numerische Übertragung, wobei das Sendefeld wie ein extern dezimales Datenelement behandelt wird
nu3	=	numerische Übertragung mit Rundung

Allgemeine Regeln

1. bezeichner-1 oder literal werden in jedes durch bezeichner-2 angesprochene Datenfeld in der angegebenen Reihenfolge übertragen.
2. Jede Subskribierung, Indizierung, Teilfeldselektion und Längenberechnung im Zusammenhang mit bezeichner-2 wird unmittelbar vor der Übertragung der Daten in das entsprechende Datenfeld durchgeführt. Ist bezeichner-2 ein null-längiges Datenfeld, so wird dieses durch die MOVE-Anweisung nicht verändert.
3. Jede Subskribierung, Indizierung, Teilfeldselektion, Funktionswertberechnung und Längenberechnung im Zusammenhang mit bezeichner-1 wird nur einmal vor der ersten Übertragung durchgeführt. Ist bezeichner-1 ein null-längiges Datenfeld, so wird die MOVE-Anweisung ausgeführt, als wäre SPACE als Sendefeld angegeben.
4. Jede erforderliche Konvertierung der Daten von einer internen Darstellungsform in eine andere findet während gültiger Übertragungen statt, ebenso wie die ggf. geforderte Druckaufbereitung des Empfangsfeldes.
5. Jede Übertragung, bei der das Sendefeld ein Literal oder Datenelement ist und das Empfangsfeld ebenfalls ein Datenelement ist, ist eine **Elementübertragung**. In einer MOVE-Anweisung von Format 1 wird eine nationale Gruppe wie ein Datenelement behandelt.

   Jede andere Übertragung ist eine **Gruppenübertragung**. Die Gruppenübertragung entspricht einer Elementübertragung mit alphanumerischem Sende- und Empfangsfeld mit dem Unterschied, dass dabei keine Konvertierung erfolgt, d.h. das Empfangsfeld wird aufgefüllt ohne die Datengruppen bzw. Datenfelder zu berücksichtigen, die im Sende- und Empfangsfeld enthalten sind.
   (Besonderheiten im Zusammenhang mit der OCCURS DEPENDING-Angabe siehe Allgemeine Regel 14 der OCCURS-Klausel auf "OCCURS-Klausel").
6. Jede Druckaufbereitung, die für ein Empfangsdatenfeld angegeben ist, wird während einer Elementübertragung durchgeführt (einige Beispiele sind unten aufgeführt).
7. Deeditierung findet nur dann statt, wenn das Sendefeld numerisch druckaufbereitet und das Empfangsfeld numerisch oder numerisch druckaufbereitet ist.
8. Es gibt zwei Arten von Elementübertragungen:
   a. Eine **nichtnumerische Übertragung** findet statt, wenn das Empfangsfeld ein alphanumerisch druckaufbereitetes, ein alphanumerisches, ein alphabetisches oder ein nationales Feld ist.

- Ausrichtung und Auffüllen mit Leerzeichen erfolgen entsprechend Abschnitt " 5. Ausrichtung von Daten" auf "Konzept der maschinenunabhängigen Datenbeschreibung".
- Ist das Sendefeld als numerisch mit Vorzeichen beschrieben, wird das Vorzeichen nicht übertragen. Belegt das Vorzeichen eine eigene Zeichenposition, wird es nicht übertragen und die Länge des Sendefeldes wird um 1 Zeichen vermindert angenommen.
- Weicht die USAGE-Klausel des Sendefeldes von der des Empfangsfeldes ab, so findet eine Konvertierung des Sendefeldes zur internen Darstellung des Empfangsfeldes statt.
- Ist das Sendefeld numerisch und enthält das Maskenzeichen 'P', werden alle diese Zeichenstellen bei der Größe des Sendefeldes mitgezählt und angenommen, dass sie den Wert 0 enthalten.

b. Eine **numerische Übertragung** findet statt, wenn das Empfangsfeld numerisch oder numerisch druckaufbereitet ist.

- Ausrichtung und Auffüllen mit Nullen erfolgen entsprechend Abschnitt "5. Ausrichtung von Daten" auf "Konzept der maschinenunabhängigen Datenbeschreibung".
- Ist das Sendefeld numerisch druckaufbereitet, wird es deeditiert, um seinen numerischen Wert zu bestimmen. Dieser Wert wird in das Empfangsfeld übertragen.
- Ist das Empfangsfeld mit Vorzeichen beschrieben, so wird das Vorzeichen des Sendefeldes übertragen. Ist das Sendefeld ohne Vorzeichen, wird im Empfangsfeld ein positives Vorzeichen erzeugt.
- Ist das Empfangsfeld ohne Vorzeichen, wird der absolute Wert des Sendefeldes übertragen und im Empfangsfeld kein Vorzeichen erzeugt.
- Ist das Sendefeld alphanumerisch, wird es wie ein extern dezimales Datenelement behandelt. Es darf nur numerische Zeichen enthalten. Ist es länger als 31 Zeichen, so werden nur die 31 am weitesten rechts stehenden Zeichen verwendet.

9. Jede Konvertierung zwischen verschiedenen Arten der internen Darstellung erfolgt während erlaubter Elementübertragungen. Ebenso erfolgt die Konvertierung von alphanumerischer zu nationaler Zeichendarstellung. Existiert dabei für ein alphanumerisches Zeichen im Sendefeld kein entsprechendes nationales Zeichen, so wird im Empfangsfeld dafür das Ersatzzeichen "Punkt" verwendet und die Ausnahmesituation EC-DATA-CONVERSION tritt auf.

> **i** Tritt die Ausnahmesituation EC-DATA-CONVERSION auf, so wird das Empfangsfeld immer verändert, bevor eine ggf. vorhandene USE-Prozedur aktiviert wird (siehe auch Abschnitt "NATIONAL-OF - nationale Zeichendarstellung").

10. Belegen die Sende- oder Empfangsoperanden einer MOVE-Anweisung den gleichen Internspeicherbereich (d.h. überlagern sich die Operanden), so ist das Ergebnis der Übertragung unvorhersagbar.
11. Figurative Konstanten werden wie numerische, alphanumerische oder nationale Literale behandelt, abhängig von der Art des Empfangsfeldes. Die folgende Tabelle zeigt die sich ergebende Kategorie des Literals.

figurative Konstante	**Art des Empfangsfeldes**	**Kategorie der figurativen Konstante**
ALL Alphanumerisches Literal	unabhängig davon	Alphanumerisch
ALL Nationales Literal	unabhängig davon	National
	unabhängig davon	Alphanumerisch

ALL Symbolisches-Zeichen		
HIGH-VALUE, LOW-VALUE, QUOTE	Gruppe, Alphanumerisch, Alphanumerisch druckaufbereitet	Alphanumerisch
	National	National
SPACE	Gruppe, Alphanumerisch, Alphabetisch, Alphanumerisch druckaufbereitet	Alphanumerisch
	National	National
ZERO	Gruppe, Alphanumerisch, Alphanumerisch druckaufbereitet	Alphanumerisch
	National	National
	Extern dezimal, Binär, Intern dezimal, Numerisch druckaufbereitet, Extern Gleitpunkt, Intern Gleitpunkt	Numerisch

Tabelle 29: Übertragung von figurativen Konstanten

Beispiel 8-55

Sendefeld		**Empfangsfeld**		**Begründung**
Masken-zeichenfolge	Inhalt	Masken-zeichenfolge	Inhalt	
XXX	M8N	XXXXX	M8N 'BLANK' 'BLANK'	Ist für das Empfangsfeld keine JUSTIFIED RIGHT-Klausel angegeben, werden die Zeichen von links nach rechts übertragen. Da die zu übertragenden Daten das Empfangsfeld nicht vollständig ausfüllen, werden die verbleibenden Stellen mit Leerzeichen aufgefüllt.
AAAAAA	XYZABC	AAA AAA JUST RIGHT	XYZ ABC	Da das Sendefeld länger als das Empfangsfeld ist, wird die Übertragung abgebrochen, sobald das Empfangsfeld gefüllt ist. Die übrigen Zeichen werden ignoriert. Ist für das Empfangsfeld die JUSTIFIED RIGHT-Klausel angegeben, werden die Zeichen rechtsbündig ins Empfangsfeld übertragen.
999PPP	456	XXXXX	45600	An Stellen des Maskenzeichens P werden Nullen angenommen. Das Sendefeld hat somit 6 Zeichen. Sie werden linksbündig in das Empfangsfeld übertragen. Die übrigen Zeichen werden ignoriert.
S999	-333	XXX	333	Hat das Sendefeld ein Vorzeichen, wird der absolute Wert übertragen.
9V999	8765	V99	76	Die Felder werden am Dezimalpunkt ausgerichtet. Das Sendefeld hat sowohl links als auch rechts vom Dezimalpunkt mehr Stellen als das Empfangsfeld. Es werden an beiden Enden Stellen abgeschnitten.

9V9	12	99V99	0120	Ist das Empfangsfeld größer als das Sendefeld, werden die nicht verwendeten Zeichenstellen mit Nullen oder mit Druckaufbereitungszeichen aufgefüllt.
99V99	6789	$$$99.99	'BLANK' 'BLANK' $67.89	
S999	-333	9999	333	Hat das Sendefeld ein Vorzeichen und das Empfangsfeld kein Vorzeichen, wird der absolute Wert des Sendefeldes übertragen.
99	15	999	015	Ist kein Rechendezimalpunkt vorhanden, werden die Daten rechtsbündig ins Empfangsfeld übertragen.
XXX	123	99V9	230	Das Sendefeld wird wie ein ganzzahliges numerisches Feld mit 3 Ziffernstellen behandelt. Die führende Hunderterstelle hat kein Platz im Empfangsfeld und wird abgeschnitten.

Format 2

```
MOVE {CORRESPONDING | CORR} bezeichner-1 TO {bezeichner-2}...
```

Syntaxregeln

Die folgenden Regeln für bezeichner-2 beziehen sich gleichfalls auf alle weiteren Bezeichner, die nach bezeichner-2 angegeben sind.

1. CORR ist die Abkürzung für CORRESPONDING.
2. bezeichner-1 bezeichnet eine Datengruppe, die die zu übertragenden Datenelemente enthält.
3. bezeichner-2 bezeichnet eine Datengruppe, die die Empfangsfelder für die Übertragung enthält.
4. Die ausgewählten Datenfelder innerhalb des ersten Operanden (bezeichner-1) werden in die entsprechenden Datenfelder innerhalb des zweiten Operanden (bezeichner-2) übertragen. Die Datenfelder einer jeden Gruppe werden als entsprechend betrachtet, wenn beide Datenfelder gleiche Namen und Kennzeichnung haben, bis zu, aber nicht unbedingt einschließlich bezeichner-1 und bezeichner-2.
5. Jede Subskribierung oder Indizierung im Zusammenhang mit bezeichner-2 wird unmittelbar vor der Übertragung der Daten in das entsprechende Datenfeld aufgelöst.
6. Jede erforderliche Konvertierung der Daten von einer internen Darstellungsform in eine andere findet während gültiger Übertragungen statt, ebenso wie die ggf. geforderte Druckaufbereitung des Empfangsfeldes. Dies wird in den folgenden Allgemeinen Regeln genauer beschrieben.

Allgemeine Regeln

1. Die Ergebnisse der MOVE CORRESPONDING-Anweisung sind die gleichen, als ob der Benutzer jedes Paar von entsprechenden Bezeichnern in einer getrennten MOVE-Anweisung des Formats 1 angegeben hätte (weitere Regeln siehe "CORRESPONDING-Angabe").
2. Mindestens eines der Datenfelder in einem Paar entsprechender Datenfelder muss ein Datenelement sein. (Man beachte, dass die MOVE-Anweisung sich in dieser Hinsicht von arithmetischen Anweisungen unterscheidet: In arithmetischen Anweisungen, die die CORRESPONDING-Angabe verwenden, müssen die beiden entsprechenden Felder Datenelemente sein.)
3. Ein Datenfeld, das bezeichner-1 oder bezeichner-2 untergeordnet ist und eine OCCURS-, REDEFINES-, USAGE IS INDEX-, USAGE IS POINTER-, USAGE ISPROGRAM-POINTER-, USAGE ISOBJECT REFERENCE- oder RENAMES-Klausel enthält, wird ignoriert. bezeichner-1 oder bezeichner-2 können jedoch selbst REDEFINES- oder OCCURS-Klauseln haben oder einem Datenfeld mit einer REDEFINES- oder OCCURS-Klausel untergeordnet sein.

4. In einer MOVE-Anweisung von Format 2 wird eine nationale Gruppe wie eine Gruppe behandelt (und nicht wie ein Datenelement).

Beispiel 8-56

Anweisung in der PROCEDURE DIVISION:

```
MOVE CORRESPONDING ARBEITER-SATZ TO LOHN-KONTROLLE.
```

Einträge in der DATA DIVISION:

```
01 ARBEITER-SATZ.
   02 ARBEITER-NR.
      03 ARBEITS-ORT...
      03 STECHUHR-NR.
         04 SCHICHT-
KENNZAHL...
         04 KONTROLL-NR...
   02 LOHN.
      03 ARBEITS-STUNDEN...
      03 AUSZAHLUNG...
   02 STEUER-SATZ...
   02 ABZUEGE...
```

```
01 LOHN-KONTROLLE.
   02 ARBEITER-NR.
      03 STECHUHR-NR...
      03 FILLER...
   02 ABZUEGE.
      03 STEUER-SATZ...
      03 STEUER-BETRAG...
      03 VORSCHUSS...
   02 LOHN.
      03 ARBEITS-STUNDEN...
      03 AUSZAHLUNG...
   02 NETTO-ZAHLUNG...
   02 ARBEITER-NAME...
      03 SCHICHT-
KENNZAHL...
```

Entsprechend den Regeln der MOVE CORRESPONDING-Anweisung werden folgende Datenfelder übertragen:

Sendefeld	**Empfangsfeld**
STECHUHR-NR OF ARBEITER-NR OF ARBEITER-SATZ	STECHUHR-NR OF ARBEITER-NR OF LOHN-KONTROLLE
ARBEITS-STUNDEN OF LOHN OF ARBEITER-SATZ	ARBEITS-STUNDEN OF LOHN OF LOHN-KONTROLLE
AUSZAHLUNG OF LOHN OF ARBEITER-SATZ	AUSZAHLUNG OF LOHN OF LOHN-KONTROLLE
ABZUEGE OF ARBEITER-SATZ	ABZUEGE OF LOHN-KONTROLLE

Die folgenden Felder werden aus den angegebenen Gründen nicht übertragen:

Feld	**Begründung**
ARBEITER-NR	Feld ist in keiner der beiden Gruppen ein Datenelement.
ARBEITS-ORT OF ARBEITER-NR OF ARBEITER-SATZ	Feld tritt in LOHN-KONTROLLE nicht auf.
SCHICHT-KENNZAHL OF STECHUHR-NR OF ARBEITER-NR OF ARBEITER-SATZ	Kennzeichnung ist nicht die gleiche wie in LOHN-KONTROLLE

KONTROLL-NR OF STECHUHR-NR OF ARBEITER-SATZ	Feld kommt in LOHN-KONTROLLE nicht vor.
LOHN	Feld ist in keiner der beiden Gruppen ein Datenelement
STEUER-SATZ OF ARBEITER-SATZ	Kennzeichnung ist nicht die gleiche wie in LOHN-KONTROLLE.

8.10.30 MULTIPLY-Anweisung

Funktion

Die MULTIPLY-Anweisung führt die Multiplikation zweier numerischer Operanden durch und speichert das Ergebnis ab.

Format 1 der MULTIPLY-Anweisung speichert die Produkte in den angegebenen Multiplikatoren ab.

Format 2 der MULTIPLY-Anweisung verwendet die GIVING-Angabe.

Format 1

```
MULTIPLY {bezeichner-1 | literal-1} BY {bezeichner-2 [ROUNDED]}...
    [ON SIZE ERROR unbedingte-anweisung-1]
    [NOT ON SIZE ERROR unbedingte-anweisung-2]
    [END-MULTIPLY]
```

Syntaxregeln

1. Jeder Bezeichner der dem Wort GIVING vorangeht, muss sich auf ein numerisches Datenelement beziehen.
2. bezeichner-3... kann sich auf ein numerisches Datenelement oder ein numerisch druckaufbereitetes Datenelement beziehen.
3. Der Wert von bezeichner-1 oder literal-1 wird mit bezeichner-2 oder literal-2 multipliziert und das Produkt in bezeichner-3 abgespeichert (entsprechendes gilt für weitere Empfangsfelder).
4. Die Maximalgröße des Produktes beträgt 31 Dezimalziffern.

Für weitere Regeln siehe unter "Angaben in Anweisungen"); die ROUNDED-Angabe, (NOT) ON SIZE ERROR-Angabe und GIVING-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-57

Anweisung	Maskenzeichenfolge des Ergebnisfeldes	Rechnung
`MULTIPLY A BY B`	`999`	`A*B` abgespeichert in `B` als nnn

Format 2

```
MULTIPLY {bezeichner-1 | literal-1} BY {bezeichner-2 | literal-2}
    GIVING {bezeichner-3 [ROUNDED]}...
    [ON SIZE ERROR unbedingte-anweisung-1]
    [NOT ON SIZE ERROR unbedingte-anweisung-2]
    [END-MULTIPLY]
```

Syntaxregeln

1. Jeder Bezeichner der dem Wort GIVING vorangeht, muss sich auf ein numerisches Datenelement beziehen.
2. bezeichner-3... kann sich auf ein numerisches Datenelement oder ein numerisch druckaufbereitetes Datenelement beziehen.
3. Der Wert von bezeichner-1 oder literal-1 wird mit bezeichner-2 oder literal-2 multipliziert und das Produkt in bezeichner-3 abgespeichert (entsprechendes gilt für weitere Empfangsfelder).
4. Die Maximalgröße des Produktes beträgt 31 Dezimalziffern.

Für weitere Regeln siehe unter "Angaben in Anweisungen"); die ROUNDED-Angabe, (NOT) ON SIZE ERROR-Angabe und GIVING-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-58

Anweisung	Maskenzeichenfolge des Ergebnisfeldes (C)	Rechnung
`MULTIPLY A BY B GIVING C`	`9(5)`	`A*B` abgespeichert in `C` als nnnnn

8.10.31 OPEN-Anweisung

Funktion

Die OPEN-Anweisung eröffnet Dateien für die Verarbeitung und für **sequenziell organi sierte Dateien** führt sie noch die Prüfung und Ausgabe von Kennsätzen durch.

Format 1 für sequenzielle Dateiorganisation

```
OPEN {  INPUT {dateiname-1 [ REVERSED | WITH NO REWIND]}...
      | OUTPUT {dateiname-1 [WITH NO REWIND]}...
      | I-O {dateiname-1}...
      | EXTEND {dateiname-1}...
     }...
[END-OPEN]
```

Syntaxregeln

1. Für Mehrdateispulen darf die Angabe EXTEND nicht gemacht werden (siehe auch „I-O-CONTROL-Paragraf").
2. Die EXTEND-Angabe darf nur für Dateien verwendet werden, für die keine LINAGE-Klausel angegeben wurde.
3. Ein Dateiname darf in einer OPEN-Anweisung nicht mehrmals auftreten.
4. Die Angabe I-O ist nur für Plattenspeicherdateien erlaubt.
5. Die Organisation oder Zugriffsart der in einer OPEN-Anweisung aufgeführten Dateien kann unterschiedlich sein.
6. Bei zeilensequenziellen Dateien sind als Eröffnungsarten nur OPEN INPUT und OPEN OUTPUT ohne die Angaben REVERSED und NO REWIND zulässig.

Format 2 für relative und indizierte Dateiorganisation

```
OPEN {  INPUT {dateiname-1}...
      | OUTPUT {dateiname-1}...
      | I-O {dateiname-1}...
      | EXTEND {dateiname-1}...
     }...
[END-OPEN]
```

Syntaxregeln

1. Der gleiche Dateiname darf in einer OPEN-Anweisung nicht mehrmals auftreten.
2. Die Organisation oder Zugriffsart der in einer OPEN-Anweisung aufgeführten Dateien kann unterschiedlich sein.
3. Die EXTEND-Angabe darf nur für Dateien mit sequenziellem Zugriff verwendet werden.

Allgemeine Regeln

Für **sequenzielle, relative und indizierte Dateiorganisation:**

1. Nach erfolgreicher Ausführung einer OPEN-Anweisung ist die durch dateiname-1 angegebene Datei verfügbar und befindet sich in eröffnetem Zustand.
2. Nach erfolgreicher Ausführung einer OPEN-Anweisung steht der zugehörige Satzbereich dem Programm zur Verfügung. Für eine externe Datei existiert nur ein Satzbereich, der allen Programmen, die diese Datei beschreiben, zur Verfügung steht.
3. Vor der ersten erfolgreichen Durchführung einer OPEN-Anweisung für eine Datei darf (außer einer SORT- oder MERGE-Anweisung mit der USING- oder GIVING-Angabe) keine Anweisung ausgeführt werden, die explizit oder implizit auf die Datei Bezug nimmt.
4. INPUT bedeutet, dass die Datei als Eingabedatei verarbeitet werden soll (Eingabemodus).
5. OUTPUT bedeutet, dass die Datei als Ausgabedatei verarbeitet werden soll (Ausgabemodus).
6. I-O bedeutet, dass Ein- und Ausgabeoperationen (d.h. Lese- und Aktualisierungsoperationen) auf die Datei angewendet werden sollen.
   Da diese Angabe die Existenz der Datei voraussetzt, kann sie nicht verwendet werden, falls die Datei neu eingerichtet werden soll (Aktualisierungsmodus).
7. Die Angabe EXTEND bewirkt eine Positionierung der Datei unmittelbar hinter den letzten logischen Datensatz der Datei. Bei sequenziell organisierten Dateien ist dies der letzte in die Datei geschriebene Satz, bei relativer Organisation der Satz mit dem größten Relativ-Schlüssel, bei indizierter Dateiorganisation der Satz mit dem höchsten Wert des Primär-Schlüssels. Durch nachfolgende WRITE-Anweisungen für diese Datei werden Datensätze in gleicher Weise an die Datei angefügt, als wäre sie mit der Angabe OUTPUT eröffnet worden (Erweiterungsmodus).
8. Alle Angaben INPUT, OUTPUT, I-O oder EXTEND sind bei verschiedenen OPEN-Anweisungen in demselben Programm für eine Datei möglich. Bevor eine weitere OPEN-Anweisung nach der logisch ersten für dieselbe Datei in einem Programm durchlaufen werden kann, muss eine CLOSE-Anweisung durchlaufen worden sein. Dabei darf REEL/UNIT oder LOCK in der CLOSE-Anweisung nicht angegeben werden.
9. Die minimalen und maximalen Satzlängen einer Datei werden bei deren Erzeugung festgelegt und dürfen nachfolgend nicht geändert werden.
10. Ist eine optionale Datei nicht verfügbar, bewirkt die erfolgreiche Ausführung einer OPEN-Anweisung mit I-O- oder EXTEND-Angabe, dass die Datei angelegt wird.
11. Nach Ausführung einer OPEN-Anweisung wird der Inhalt des in der FILE STATUS-Klausel angegebenen Datenfeldes (falls eine solche Klausel vorhanden war) aktualisiert (siehe auch „FILE STATUS-Klausel").
12. Sind mehrere Dateinamen angegeben, so ist die Wirkung die gleiche, wie wenn für jeden Dateinamen eine eigene OPEN-Anweisung gegeben worden wäre.
13. Ist eine mit OPEN INPUT eröffnete optionale Datei nicht vorhanden, dann wird der Ein-/Ausgabestatus auf den entsprechenden Wert gesetzt.
14. Werden Dateien mit INPUT oder I-O geöffnet, so wird wie folgt verfahren:
    - Bei relativen und sequenziellen Dateien wird der Dateipositionsindikator auf den ersten Datensatz gesetzt,
    - Bei indizierten Dateien wird der Dateipositionsindikator auf den kleinsten möglichen Wert für den primären Satzschlüssel gesetzt.

    Für **sequenziell organisierte Dateien** gilt noch:
15. REVERSED zeigt an, dass die Datensätze einer Datei in umgekehrter Reihenfolge eingelesen werden sollen, d.h. beginnend beim letzten Datensatz der Datei.

    Anmerkung:
    Für eine Datei mit nicht standardisierten Kennsätzen sollte die Angabe REVERSED nicht verwendet werden, außer wenn auf den letzten nicht standardisierten Kennsatz eine Bandabschnittsmarke folgt. In allen anderen Fällen liest das System die Kennsätze in gleicher Weise wie Datensätze.

16. NO REWIND kann für eine Banddatei oder für eine Plattenspeicherdatei angegeben werden, die mit OUTPUT eröffnet wurde. Die für Band- und Plattenspeicherdateien durchgeführten Maßnahmen sind im Folgenden aufgeführt:

    Banddateien

    NO REWIND bedeutet, dass die Magnetbandspule nicht zurückgespult werden soll, wenn die Datei eröffnet wird.

    Plattenspeicherdateien

    NO REWIND bedeutet, dass in der Datei bereits vorhandene Datensätze nicht überschrieben werden sollen. NO REWIND ist nur aus Kompatibilitätsgründen vorhanden. OPEN OUTPUT WITH NO REWIND hat die gleiche Funktion wie OPEN EXTEND.

17. Falls das Speichermedium, auf dem sich die Datei befindet, ein Rückspulen zulässt, gelten die folgenden Regeln:
    a. Falls die Angaben REVERSED, EXTEND oder NO REWIND fehlen, wird die Datei bei Ausführung der OPEN-Anweisung auf Dateianfang positioniert.
    b. Falls NO REWIND angegeben wurde, wird die Datei bei Ausführung der OPEN-Anweisung nicht neu positioniert; vielmehr wird erwartet, dass die Datei bereits auf den Dateianfang positioniert ist.
    c. Falls REVERSED angegeben wurde, werden die Datensätze der Datei in umgekehrter Reihenfolge, d. h. beginnend mit dem letzten Datensatz der Datei, zur Verfügung gestellt.

18. Bei Ausführung der OPEN-Anweisung werden Kennsatzbehandlungsroutinen durchlaufen, außer wenn LABEL RECORDS ARE OMITTED in der Dateierklärung angegeben ist.

    Für die OPEN-Anweisung mit INPUT-Angabe werden folgende Schritte durchlaufen:
    a. Standarddateikennsätze, falls vorhanden, werden überprüft.
    b. Falls Benutzerkennsätze oder nicht standardisierte Kennsätze vorhanden sind und eine dafür zuständige USE-Prozedur existiert, wird diese ausgeführt.
    c. Die Datei wird danach so positioniert, dass der erste Datensatz gelesen werden kann.

    Für die OPEN-Anweisung mit OUTPUT-Angabe werden folgende Schritte durchlaufen:
    a. Falls Standardkennsätze für die Datei vereinbart sind, werden sie erzeugt und geschrieben.
    b. Falls eine USE-Prozedur definiert ist, wird diese ausgeführt und Benutzerkennsätze oder nicht standardisierte Kennsätze geschrieben.
    c. Der Satzbereich wird dem Programm zur Aufnahme von Daten zur Verfügung gestellt.

    Für die OPEN-Anweisung mit I-O-Angabe werden folgende Schritte durchlaufen:
    a. Falls Standardkennsätze vorhanden sind, werden diese überprüft.
    b. Falls Benutzerkennsätze vorhanden sind und eine dafür zuständige USE-Prozedur existiert, wird diese ausgeführt.
    c. Die Datei wird zum Lesen oder Ersetzen von Daten positioniert.

    Für die OPEN-Anweisung mit EXTEND-Angabe und bei Vorhandensein der Klausel LABEL RECORDS STANDARD/datenname besteht die Ausführung der OPEN-Anweisung aus folgenden Schritten:
    a. Die Dateianfangskennsätze werden nur im Falle einer Einspulen- oder Eindatenträgerdatei verarbeitet.
    b. Die Spulen-/Datenträgeranfangskennsätze auf der/dem letzten existierenden Spule/Datenträger werden verarbeitet wie im Falle der OPEN-Anweisung mit INPUT-Angabe.
    c. Die vorhandenen Dateiendekennsätze werden wie im Falle der OPEN-Anweisung mit INPUT-Angabe verarbeitet. Diese Kennsätze werden dann gelöscht.
    d. Die weitere Verarbeitung geht dann in gleicher Weise vonstatten wie bei der OPEN-Anweisung mit OUTPUT-Angabe.

Übersicht über die für einen OPEN-Modus zulässigen Anweisungen.

ACCESS-Klausel	Anweisung	Eröffnungsmodus			
		INPUT	OUTPUT	I-O	EXTEND
SEQUENTIAL	READ	R,I,S,L		R,I,S	
	WRITE		R,I,S,L		R,I,S
	REWRITE			R,I,S	
	START	R,I		R,I	
	DELETE			R,I	
RANDOM	READ	R,I		R,I	
	WRITE		R,I	R,I	
	REWRITE			R,I	
	START				
	DELETE			R,I	
DYNAMIC	READ	R,I		R,I	
	WRITE		R,I	R,I	
	REWRITE			R,I	
	START	R,I		R,I	
	DELETE			R,I	

Tabelle 30: Zulässige Ein-/Ausgabe-Anweisungen für jeden OPEN-Modus

Zulässige Angaben sind durch folgende Einträge gekennzeichnet:

S für sequenzielle Dateiorganisation

L für zeilensequenzielle Dateiorganisation

R für relative Dateiorganisation

I für indizierte Dateiorganisation.

8.10.32 PERFORM-Anweisung

Funktion

Die PERFORM-Anweisung dient dazu, eine oder mehrere Prozeduren oder eine Reihe von Anweisungen auszuführen.

Forma 1 gilt für die einmalige Ausführung der angegebenen Prozeduren oder Anweisungen.

Format 2 gilt für die Ausführung der angegebenen Prozeduren oder Anweisungen mit einer gegebenen Anzahl von Wiederholungen.

Format 3 gilt für die Ausführung der angegebenen Prozeduren oder Anweisungen, die so oft ausgeführt werden, bis eine bestimmte Bedingung erfüllt ist.

Format 4 gilt für die Veränderung des Wertes von ein oder mehreren Bezeichnern oder Indexnamen in auf- oder absteigender Folge. Davon abhängig werden eine Reihe von Prozeduren oder die angegebenen Anweisungen ein oder mehrmals ausgeführt.

Format 1

"out-of-line"

```
PERFORM prozedurname-1 [{THRU | THROUGH} prozedurname-2]
```

"in-line"

```
PERFORM [unbedingte-anweisung END-PERFORM]
```

Syntaxregeln

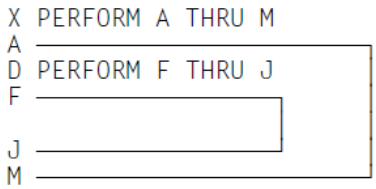
1. Die Wörter THROUGH und THRU sind gleichbedeutend.
2. Wenn prozedurname-1 und prozedurname-2 angegeben sind und jeweils Namen einer Prozedur innerhalb von Prozedurvereinbarungskapiteln sind, so müssen beide Prozedurnamen innerhalb desselben Prozedurvereinbarungskapitels liegen.

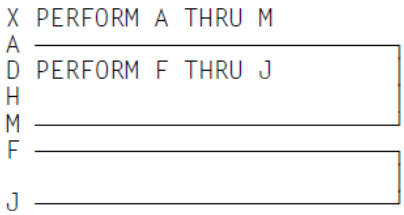
Allgemeine Regeln

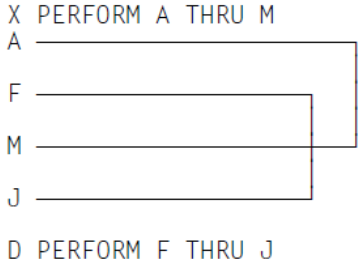
1. END-PERFORM beendet den Gültigkeitsbereich der „in-line“-PERFORM-Anweisung.
2. Eine „in-line“-PERFORM-Anweisung funktioniert in Übereinstimmung mit den folgenden Allgemeinen Regeln genauso wie eine „out-of-line“-PERFORM-Anweisung, mit der Ausnahme, dass die in einem „in-line “-PERFORM enthaltenen Anweisungen an Stelle der im Bereich von prozedurname-1 (bis prozedurname-2, falls angegeben) stehenden Anweisungen ausgeführt werden. Alle Allgemeinen Regeln gelten für „in-line“- und „out-of-line“-PERFORM gleichermaßen, außer es wird gesondert auf die beiden PERFORM-Arten eingegangen.
3. Wenn eine PERFORM-Anweisung ausgeführt wird, so geht die Ablaufsteuerung zu der ersten Anweisung der Prozedur über, die mit prozedurname-1 bezeichnet ist. Der implizite Rücksprung zur nächsten ausführbaren Anweisung nach der PERFORM-Anweisung erfolgt folgendermaßen:
   a. Wenn prozedurname-1 ein Paragrafenname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung von prozedurname-1.
   b. Wenn prozedurname-1 ein Kapitelname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung des letzten Paragrafen von prozedurname-1.
   c. Wenn prozedurname-2 angegeben und ein Paragrafenname ist, erfolgt der Rücksprung nach der letzten Anweisung des Paragrafen.

   d. Wenn prozedurname-2 angegeben und ein Kapitelname ist, so erfolgt der Rücksprung nach der letzten Anweisung im letzten Paragrafen des Kapitels.
   e. Wird ein „in-line“-PERFORM verwendet, ist die Ausführung der PERFORM-Anweisung nach der letzten in der PERFORM-Anweisung enthaltenen Anweisung abgeschlossen.

4. Es gibt keine notwendige Verbindung zwischen prozedurname-1 und prozedurname-2, außer dass eine fortlaufende Reihe von Operationen, beginnend bei prozedurname-1, ausgeführt werden soll, die mit der Ausführung der mit prozedurname-2 bezeichneten Prozedur endet. Sogar GO TO- und PERFORM-Anweisungen können zwischen prozedurname-1 und dem Ende von prozedurname-2 auftreten. Wenn es zwei oder mehr logische Wege zum Rücksprungpunkt gibt, kann prozedurname-2 der Name eines Paragrafen sein, der eine EXIT-Anweisung enthält, auf die alle diese Ablaufwege führen müssen.
5. Die Anweisungen innerhalb einer PERFORM-Anweisung werden einmal ausgeführt, und der Ablauf wird bei der Anweisung fortgesetzt, die der PERFORM-Anweisung folgt.
6. Der Bereich einer PERFORM-Anweisung besteht logisch aus all jenen Anweisungen, die infolge der PERFORM-Anweisung ausgeführt werden; dies erfolgt auf Grund des impliziten Übergangs der Ablaufsteuerung zum Ende der PERFORM-Anweisung. Der Bereich schließt alle Anweisungen ein, die als Folge von CALL-, EXIT-, GO TO- und PERFORM-Anweisungen im Bereich einer PERFORM-Anweisung ausgeführt werden; dies gilt ebenso für alle Anweisungen, die im Rahmen von Prozedurvereinbarungen infolge der Ausführung von Anweisungen im Bereich der PERFORM-Anweisung ausgeführt werden. Die Anweisungen im Bereich einer PERFORM-Anweisung müssen nicht aufeinanderfolgend in der Übersetzungseinheit stehen.
7. Anweisungen, die auf Grund einer EXIT PROGRAM-Anweisung ausgeführt werden, gelten nicht als Teil des Bereichs der PERFORM-Anweisung, wenn:
   a. die EXIT PROGRAM-Anweisung im selben Programm angegeben ist, in dem auch die PERFORM-Anweisung steht, und
   b. die EXIT PROGRAM-Anweisung innerhalb des Bereichs der PERFORM-Anweisung steht.
8. Steht im Bereich einer PERFORM-Anweisung eine weitere PERFORM-Anweisung, muss der Bereich der eingebetteten PERFORM-Anweisung entweder völlig innerhalb oder völlig außerhalb des Bereichs der ersten PERFORM-Anweisung liegen. Infolgedessen darf eine PERFORM-Anweisung, deren Ausführung im Bereich einer anderen PERFORM-Anweisung beginnt, nicht zum Ausgang der anderen PERFORM-Anweisung führen; außerdem dürfen zwei oder mehr solcher PERFORM-Anweisungen keinen gemeinsamen Ausgang haben.
   Die folgenden Beispiele zeigen zugelassene PERFORM-Konstruktionen:







9. Dieser Punkt enthält Angaben zur Segmentierung (für weitere Angaben siehe unter Kapitel „Segmentierung").
   a. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer kleiner als die Angabe in der SEGMENT-LIMIT-Klausel ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:

- entweder Kapitel, die alle eine Segmentnummer kleiner 50 haben,
- oder Kapitel, die als Ganzes in einem einzigen Segment enthalten sind, dessen Segmentnummer größer als 49 ist.

b. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer größer als 49 ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
   - entweder Kapitel, die alle die gleiche Segmentnummer haben wie das Kapitel, das die PERFORM-Anweisung enthält,
   - oder Kapitel, die alle eine Segmentnummer kleiner 50 haben.

c. Wird ein Prozedurname in einem Segment mit einer Segmentnummer größer 49 durch eine PERFORM-Anweisung angesprochen, die in einem Segment mit einer unterschiedlichen Segmentnummer liegt, so wird das angesprochene Segment in seinem Initialzustand zur Verfügung gestellt (d.h. also, dass alle durch ausgeführte ALTER-Anweisungen veränderten GO TO-Anweisungen in diesem Segment auf ihren Ursprungswert gesetzt werden).

Beispiel 8-59

```
PERFORM X-PAR.
ADD A TO B.
```

X-PAR ist der Paragrafenname. In diesem Fall werden alle unter X-PAR stehenden Anweisungen ausgeführt und der Ablauf bei der ADD-Anweisung, die der PERFORM-Anweisung folgt, fortgesetzt.

Beispiel 8-60

```
    ...
    PERFORM X1-PAR THRU X3-PAR.
    ...
X-KAP SECTION.
X1-PAR.
    ...
X2-PAR.
    ...
X3-PAR.
    ...
Y-KAP SECTION.
    ...
```

Die PERFORM-Anweisung bewirkt, dass alle Anweisungen in den Paragrafen X1-PAR, X2-PAR und X3-PAR ausgeführt werden.

Beispiel 8-61

Die gleiche Wirkung wie im vorausgehenden Beispiel würde mit der folgenden Anweisung erzielt:

```
PERFORM X-KAP.
```

Format 2

"out-of-line"

```
PERFORM prozedurname-1 [{THRU | THROUGH} prozedurname-2] {bezeichner-1 | ganzzahl-1} TIMES
```

"in-line"

```
PERFORM {bezeichner-1 | ganzzahl-1} TIMES unbedingte-anweisung END-PERFORM
```

Syntaxregeln

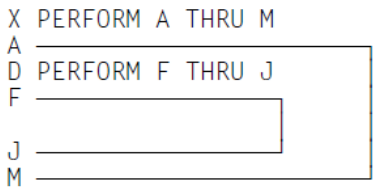
1. bezeichner-1 bezeichnet ein als ganzzahlig beschriebenes numerisches Datenfeld.
2. Die Wörter THROUGH und THRU sind gleichbedeutend.
3. Wenn prozedurname-1 und prozedurname-2 angegeben sind und jeweils Namen einer Prozedur innerhalb von Prozedurvereinbarungsprozeduren sind, so müssen beide Prozedurnamen innerhalb desselben Prozedurvereinbarungskapitels liegen.

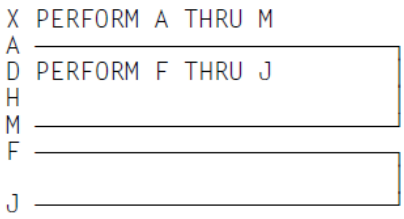
Allgemeine Regeln

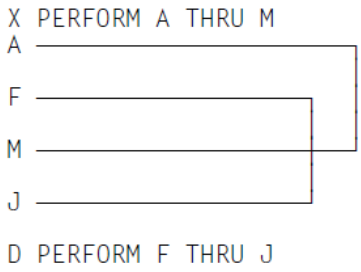
1. END-PERFORM beendet den Gültigkeitsbereich der „in-line“-PERFORM-Anweisung.
2. Eine „in-line“-PERFORM-Anweisung funktioniert in Übereinstimmung mit den folgenden Allgemeinen Regeln genauso wie eine „out-of-line“-PERFORM-Anweisung, mit der Ausnahme, dass die in einem „in-line “-PERFORM enthaltenen Anweisungen an Stelle der im Bereich von prozedurname-1 (bis prozedurname-2, falls angegeben) stehenden Anweisungen ausgeführt werden. Alle Allgemeinen Regeln gelten für „in-line“- und „outof-line“-PERFORM gleichermaßen, außer es wird gesondert auf die beiden PERFORM-Arten eingegangen.
3. Wenn eine PERFORM-Anweisung ausgeführt wird, so geht die Ablaufsteuerung zu der ersten Anweisung der Prozedur über, die mit prozedurname-1 bezeichnet ist. Der implizite Rücksprung zur nächsten ausführbaren Anweisung nach der PERFORM-Anweisung erfolgt folgendermaßen:
   a. Wenn prozedurname-1 ein Paragrafenname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung von prozedurname-1.
   b. Wenn prozedurname-1 ein Kapitelname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung des letzten Paragrafen von prozedurname-1.
   c. Wenn prozedurname-2 angegeben und ein Paragrafenname ist, erfolgt der Rücksprung nach der letzten Anweisung des Paragrafen.
   d. Wenn prozedurname-2 angegeben und ein Kapitelname ist, so erfolgt der Rücksprung nach der letzten Anweisung im letzten Paragrafen des Kapitels.
   e. Wird ein „in-line“-PERFORM verwendet, ist die Ausführung der PERFORM-Anweisung nach der letzten in der PERFORM-Anweisung enthaltenen Anweisung abgeschlossen.
4. Es gibt keine notwendige Verbindung zwischen prozedurname-1 und prozedurname-2, außer dass eine fortlaufende Reihe von Operationen, beginnend bei prozedurname-1, ausgeführt werden soll, die mit der Ausführung der mit prozedurname-2 bezeichneten Prozedur endet. Sogar GO TO- und PERFORM-Anweisungen können zwischen prozedurname-1 und dem Ende von prozedurname-2 auftreten. Wenn es zwei oder mehr logische Wege zum Rücksprungpunkt gibt, kann prozedurname-2 der Name eines Paragrafen sein, der eine EXIT-Anweisung enthält, auf die alle diese Ablaufwege führen müssen.
5. Die Anweisungen, die im Bereich der PERFORM-Anweisung liegen, werden so oft ausgeführt, wie durch ganzzahl-1 oder den Wert von bezeichner-1 angegeben ist. Nach der Ausführung der angegebenen Anweisungsfolge geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.
6. Falls der Wert von bezeichner-1 bei Ausführung der PERFORM-Anweisung Null oder negativ ist, geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.
7. Der Bereich einer PERFORM-Anweisung besteht logisch aus all jenen Anweisungen, die infolge der PERFORM-Anweisung ausgeführt werden; dies erfolgt auf Grund des impliziten Übergangs der Ablaufsteuerung zum Ende der PERFORM-Anweisung. Der Bereich schließt alle Anweisungen ein, die als Folge von CALL-, EXIT-, GO TO- und PERFORM-Anweisungen im Bereich einer PERFORM-Anweisung ausgeführt werden; dies gilt ebenso für alle Anweisungen, die im Rahmen von Prozedurvereinbarungen

infolge der Ausführung von Anweisungen im Bereich der PERFORM-Anweisung ausgeführt werden. Die Anweisungen im Bereich einer PERFORM-Anweisung müssen nicht aufeinanderfolgend in der Übersetzungseinheit stehen.

8. Anweisungen, die auf Grund einer EXIT PROGRAM-Anweisung ausgeführt werden, gelten nicht als Teil des Bereichs der PERFORM-Anweisung, wenn:
   a. die EXIT PROGRAM-Anweisung im selben Programm angegeben ist, in dem auch die PERFORM-Anweisung steht, und
   b. die EXIT PROGRAM-Anweisung innerhalb des Bereichs der PERFORM-Anweisung steht.
9. Steht im Bereich einer PERFORM-Anweisung eine weitere PERFORM-Anweisung, muss der Bereich der eingebetteten PERFORM-Anweisung entweder völlig innerhalb oder völlig außerhalb des Bereichs der ersten PERFORM-Anweisung liegen. Infolgedessen darf eine PERFORM-Anweisung, deren Ausführung im Bereich einer anderen PERFORM-Anweisung beginnt, nicht zum Ausgang der anderen PERFORM-Anweisung führen; außerdem dürfen zwei oder mehr solcher PERFORM-Anweisungen keinen gemeinsamen Ausgang haben.
   Die folgenden Beispiele zeigen zugelassene PERFORM-Konstruktionen:







10. Dieser Punkt enthält Angaben zur Segmentierung (für weitere Angaben siehe unter Kapitel „Segmentierung").
    a. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer kleiner als die Angabe in der SEGMENT-LIMIT-Klausel ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
       - entweder Kapitel, die alle eine Segmentnummer kleiner 50 haben,
       - oder Kapitel, die als Ganzes in einem einzigen Segment enthalten sind, dessen Segmentnummer größer als 49 ist.
    b. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer größer als 49 ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
       - entweder Kapitel, die alle die gleiche Segmentnummer haben wie das Kapitel, das die PERFORM-Anweisung enthält,
       - oder Kapitel, die alle eine Segmentnummer kleiner 50 haben.
    c. Wird ein Prozedurname in einem Segment mit einer Segmentnummer größer 49 durch eine PERFORM-Anweisung angesprochen, die in einem Segment mit einer unterschiedlichen Segmentnummer liegt, so wird das angesprochene Segment in seinem Initialzustand zur Verfügung gestellt (d.h. also, dass alle durch ausgeführte ALTER-Anweisungen veränderten GO TO-Anweisungen in diesem Segment auf ihren Ursprungswert gesetzt werden).

Beispiel 8-62

```
PERFORM X-PAR 5 TIMES.
```

Alle Anweisungen im Paragrafen X-PAR werden fünfmal ausgeführt. Der Ablauf wird dann bei der auf die PERFORM-Anweisung folgenden Anweisung fortgesetzt.

Beispiel 8-63

```
...
77 A PICTURE 9.
...
    MOVE 3 TO A.
    ...
    PERFORM X-PAR A TIMES.
    ...
X-PAR.
    ...
    ADD 1 TO A.
Y-PAR.
    ...
```

Da der Wert von A zur Ausführungszeit der PERFORM-Anweisung 3 beträgt, wird der Paragraf X-PAR dreimal ausgeführt.

Die Bezugname auf A innerhalb X-PAR hat keine Auswirkung auf die PERFORM-Anweisung.

Format 3

"out-of-line"

```
PERFORM prozedurname-1 [{THRU | THROUGH} prozedurname-2]
    [WITH TEST {BEFORE | AFTER}] UNTIL bedingung-1
```

"in-line"

```
PERFORM [WITH TEST {BEFORE | AFTER}] UNTIL bedingung-1
    unbedingte-anweisung END-PERFORM
```

Syntaxregeln

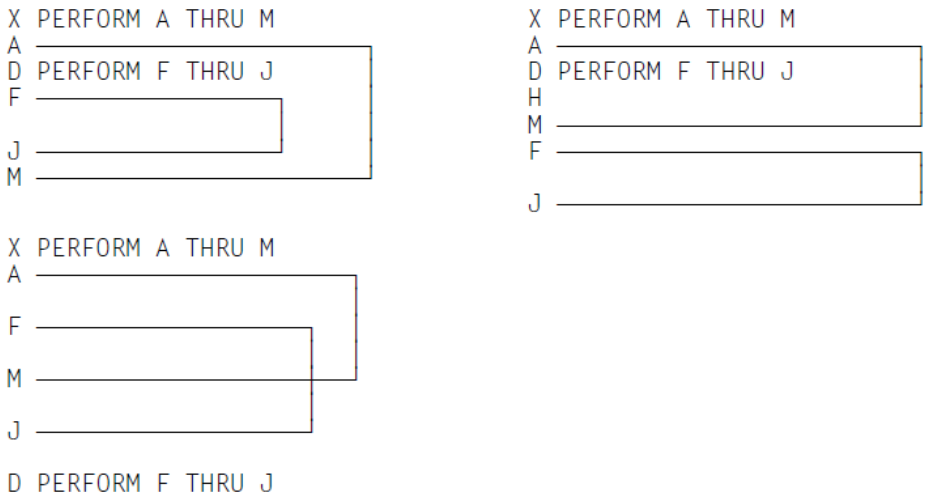
1. Die Wörter THROUGH und THRU sind gleichbedeutend.
2. Wenn prozedurname-1 und prozedurname-2 angegeben sind und jeweils Namen einer Prozedur innerhalb von Prozedurvereinbarungsprozeduren sind, so müssen beide Prozedurnamen innerhalb desselben Prozedurvereinbarungskapitels liegen.
3. Ist weder TEST BEFORE noch TEST AFTER angegeben, wird TEST BEFORE angenommen.

Allgemeine Regeln

1. END-PERFORM beendet den Gültigkeitsbereich der „in-line“-PERFORM-Anweisung.
2. Eine „in-line“-PERFORM-Anweisung funktioniert in Übereinstimmung mit den folgenden Allgemeinen Regeln genauso wie eine „out-of-line“-PERFORM-Anweisung, mit der Ausnahme, dass die in einem „in-line “-PERFORM enthaltenen Anweisungen an Stelle der im Bereich von prozedurname-1 (bis prozedurname-2, falls angegeben) stehenden Anweisungen ausgeführt werden. Alle Allgemeinen Regeln gelten für „in-line“- und „out-of-line“-PERFORM gleichermaßen, außer es wird gesondert auf die beiden PERFORM-Arten eingegangen.

3. Wenn eine PERFORM-Anweisung ausgeführt wird, so geht die Ablaufsteuerung zu der ersten Anweisung der Prozedur über, die mit prozedurname-1 bezeichnet ist. Der implizite Rücksprung zur nächsten ausführbaren Anweisung nach der PERFORM-Anweisung erfolgt folgendermaßen:
   a. Wenn prozedurname-1 ein Paragrafenname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung von prozedurname-1.
   b. Wenn prozedurname-1 ein Kapitelname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung des letzten Paragrafen von prozedurname-1.
   c. Wenn prozedurname-2 angegeben und ein Paragrafenname ist, erfolgt der Rücksprung nach der letzten Anweisung des Paragrafen.
   d. Wenn prozedurname-2 angegeben und ein Kapitelname ist, so erfolgt der Rücksprung nach der letzten Anweisung im letzten Paragrafen des Kapitels.
   e. Wird ein „in-line“-PERFORM verwendet, ist die Ausführung der PERFORM-Anweisung nach der letzten in der PERFORM-Anweisung enthaltenen Anweisung abgeschlossen.
4. Es gibt keine notwendige Verbindung zwischen prozedurname-1 und prozedurname-2, außer dass eine fortlaufende Reihe von Operationen, beginnend bei prozedurname-1, ausgeführt werden soll, die mit der Ausführung der mit prozedurname-2 bezeichneten Prozedur endet. Sogar GO TO- und PERFORM-Anweisungen können zwischen prozedurname-1 und dem Ende von prozedurname-2 auftreten. Wenn es zwei oder mehr logische Wege zum Rücksprungpunkt gibt, kann prozedurname-2 der Name eines Paragrafen sein, der eine EXIT-Anweisung enthält, auf die alle diese Ablaufwege führen müssen.
5. Die Anweisungen innerhalb einer PERFORM-Anweisung werden einmal ausgeführt, und der Ablauf wird bei der Anweisung fortgesetzt, die der PERFORM-Anweisung folgt.
6. Der Bereich einer PERFORM-Anweisung besteht logisch aus all jenen Anweisungen, die infolge der PERFORM-Anweisung ausgeführt werden; dies erfolgt auf Grund des impliziten Übergangs der Ablaufsteuerung zum Ende der PERFORM-Anweisung. Der Bereich schließt alle Anweisungen ein, die als Folge von CALL-, EXIT-, GO TO- und PERFORM-Anweisungen im Bereich einer PERFORM-Anweisung ausgeführt werden; dies gilt ebenso für alle Anweisungen, die im Rahmen von Prozedurvereinbarungen infolge der Ausführung von Anweisungen im Bereich der PERFORM-Anweisung ausgeführt werden. Die Anweisungen im Bereich einer PERFORM-Anweisung müssen nicht aufeinanderfolgend in der Übersetzungseinheit stehen.
7. Anweisungen, die auf Grund einer EXIT PROGRAM-Anweisung ausgeführt werden, gelten nicht als Teil des Bereichs der PERFORM-Anweisung, wenn:
   a. die EXIT PROGRAM-Anweisung im selben Programm angegeben ist, in dem auch die PERFORM-Anweisung steht, und
   b. die EXIT PROGRAM-Anweisung innerhalb des Bereichs der PERFORM-Anweisung steht.
8. Steht im Bereich einer PERFORM-Anweisung eine weitere PERFORM-Anweisung, muss der Bereich der eingebetteten PERFORM-Anweisung entweder völlig innerhalb oder völlig außerhalb des Bereichs der ersten PERFORM-Anweisung liegen. Infolgedessen darf eine PERFORM-Anweisung, deren Ausführung im Bereich einer anderen PERFORM-Anweisung beginnt, nicht zum Ausgang der anderen PERFORM-Anweisung führen; außerdem dürfen zwei oder mehr solcher PERFORM-Anweisungen keinen

gemeinsamen Ausgang haben.
Die folgenden Beispiele zeigen zugelassene PERFORM-Konstruktionen:



9. Die Anweisungen, die im Bereich der PERFORM-Anweisung liegen, werden solange ausgeführt, bis die Bedingung wahr ist. Wenn die Bedingung wahr ist, geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über. Ist die Bedingung zu Beginn der PERFORM-Anweisung wahr, und ist TEST BEFORE angegeben oder angenommen, findet kein Übergang zu prozedurname-1 statt, und die Ablaufsteuerung geht zum Ende der PERFORM-Anweisung über.
10. Ist TEST AFTER angegeben, wird die PERFORM-Anweisung durchgeführt wie wenn TEST BEFORE angegeben wäre, außer dass die Bedingung nach Ausführung der angegebenen Anweisungsfolge geprüft wird.
11. Alle Modifizierungen der in bedingung-1 angegebenen Operanden werden jedesmal ausgewertet, wenn die Bedingung geprüft wird.
12. Dieser Punkt enthält Angaben zur Segmentierung (für weitere Angaben siehe unter Kapitel „Segmentierung").
    a. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer kleiner als die Angabe in der SEGMENT-LIMIT-Klausel ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
       - entweder Kapitel, die alle eine Segmentnummer kleiner 50 haben,
       - oder Kapitel, die als Ganzes in einem einzigen Segment enthalten sind, dessen Segmentnummer größer als 49 ist.
    b. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer größer als 49 ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
       - entweder Kapitel, die alle die gleiche Segmentnummer haben wie das Kapitel, das die PERFORM-Anweisung enthält,
       - oder Kapitel, die alle eine Segmentnummer kleiner 50 haben.
    c. Wird ein Prozedurname in einem Segment mit einer Segmentnummer größer 49 durch eine PERFORM-Anweisung angesprochen, die in einem Segment mit einer unterschiedlichen Segmentnummer liegt, so wird das angesprochene Segment in seinem Initialzustand zur Verfügung gestellt (d.h. also, dass alle durch ausgeführte ALTER-Anweisungen veränderten GO TO-Anweisungen in diesem Segment auf ihren Ursprungswert gesetzt werden).

Beispiel 8-64

```
    PERFORM X-PAR UNTIL A GREATER THAN 3.
    ...
X-PAR.
    ...
    COMPUTE A = A + 1.
```

```
    ...
Y-PAR.
    ...
```

Es sei angenommen, dass A = 1 ist, wenn die PERFORM-Anweisung angesprochen wird. In diesem Fall werden die Anweisungen im Paragrafen X-PAR dreimal ausgeführt:

Wenn X-PAR das erste Mal ausgeführt wird, wird A auf 2 gesetzt.

Wenn X-PAR das zweite Mal ausgeführt wird, wird A auf 3 gesetzt.

Wenn X-PAR das dritte Mal ausgeführt wird, wird A auf 4 gesetzt.

Da vier größer als drei ist, ist die in der PERFORM-Anweisung angegebene Bedingung erfüllt. Daher wird der Ablauf mit der auf die PERFORM-Anweisung folgenden Anweisung fortgesetzt.

Format 4

"out-of-line"

```
PERFORM prozedurname-1 [{THRU | THROUGH} prozedurname-2] [WITH TEST {BEFORE | AFTER }]
     VARYING {index-1 | bezeichner-2} FROM {index-2 | literal-1 | bezeichner-3} BY
     {literal-2 | bezeichner-4} UNTIL bedingung-1
    [AFTER {index-3 | bezeichner-5} FROM {index-4 | literal-3 | bezeichner-6} BY
     {literal-4 | bezeichner-7} UNTIL bedingung-2]...
```

"in-line"

```
PERFORM [WITH TEST {BEFORE | AFTER}]
    VARYING {index-1| bezeichner-2} FROM {index-2 | literal-1 | bezeichner-3} BY
    {literal-2 | bezeichner-4} UNTIL bedingung-1
    unbedingte-anweisung END-PERFORM
```

Syntaxregeln

1. Die Wörter THROUGH und THRU sind gleichbedeutend.
2. Wenn prozedurname-1 und prozedurname-2 angegeben sind und jeweils Namen einer Prozedur innerhalb von Prozedurvereinbarungsprozeduren sind, so müssen beide Prozedurnamen innerhalb desselben Prozedurvereinbarungskapitels liegen.
3. Ist weder TEST BEFORE noch TEST AFTER angegeben, wird TEST BEFORE angenommen.
4. Jeder Bezeichner stellt ein numerisches, ganzzahlig beschriebenes Datenfeld dar. Der Compiler lässt jedoch zu, dass alle Bezeichner als nicht ganzzahlige numerische Datenfelder beschrieben werden.
5. Jedes Literal muss numerisch sein.
6. Das Literal in der BY-Angabe muss eine Ganzzahl ungleich Null sein.
7. Ist in der VARYING- oder AFTER-Angabe index angegeben, gilt:
   a. Der Bezeichner in der zugehörigen FROM- und BY-Angabe muss ganzzahlig sein.
   b. Das Literal in der zugehörigen FROM-Angabe muss eine positive Ganzzahl sein.
   c. Das Literal in der zugehörigen BY-Angabe muss eine Ganzzahl ungleich Null sein.
8. Ist in der FROM-Angabe index angegeben, gilt:

   a. Der Bezeichner in der zugehörigen VARYING- oder AFTER-Angabe muss ganzzahlig sein.
   b. Der Bezeichner in der zugehörigen BY-Angabe muss ganzzahlig sein.
   c. Das Literal in der BY-Angabe muss eine Ganzzahl sein.
9. bedingung-1, bedingung-2 kann jede Art von Bedingungsausdruck sein.
10. Für die „out-of-line“-PERFORM-Anweisung sind maximal 6 AFTER-Angaben zugelassen.

Allgemeine Regeln

1. bezeichner-4 und bezeichner-7 dürfen nicht den Wert Null haben.
2. Ist in der VARYING- oder AFTER-Angabe ein Index und in der zugehörigen FROM-Angabe ein Bezeichner angegeben, muss der Bezeichner einen positiven Wert haben.
3. Format 4 der PERFORM-Anweisung wird benutzt, um während der Ausführung einer PERFORM-Anweisung den Wert von einem oder mehreren Bezeichnern in bestimmter Weise zu erhöhen oder zu vermindern. Die Ausführung hängt ab von der Anzahl von Bezeichnern oder Indexnamen, die variiert werden. Die folgenden Regeln beschreiben, was geschieht, wenn ein, zwei und drei Bezeichner oder Indexnamen variiert werden.
4. END-PERFORM beendet den Gültigkeitsbereich der „in-line“-PERFORM-Anweisung.
5. Eine „in-line“-PERFORM-Anweisung funktioniert in Übereinstimmung mit den folgenden Allgemeinen Regeln genauso wie eine „out-of-line“-PERFORM-Anweisung, mit der Ausnahme, dass die in einem „in-line “-PERFORM enthaltenen Anweisungen an Stelle der im Bereich von prozedurname-1 (bis prozedurname-2, falls angegeben) stehenden Anweisungen ausgeführt werden. Alle Allgemeinen Regeln gelten für „in-line“- und „out-of-line“-PERFORM gleichermaßen, außer es wird gesondert auf die beiden PERFORM-Arten eingegangen.
6. Wenn eine PERFORM-Anweisung ausgeführt wird, so geht die Ablaufsteuerung zu der ersten Anweisung der Prozedur über, die mit prozedurname-1 bezeichnet ist. Der implizite Rücksprung zur nächsten ausführbaren Anweisung läuft folgendermaßen ab:
   a. Wenn prozedurname-1 ein Paragrafenname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung von prozedurname-1.
   b. Wenn prozedurname-1 ein Kapitelname und prozedurname-2 nicht angegeben ist, erfolgt der Rücksprung nach der letzten Anweisung des letzten Paragrafen von prozedurname-1.
   c. Wenn prozedurname-2 angegeben und ein Paragrafenname ist, erfolgt der Rücksprung nach der letzten Anweisung des Paragrafen.
   d. Wenn prozedurname-2 angegeben und ein Kapitelname ist, so erfolgt der Rücksprung nach der letzten Anweisung im letzten Paragrafen des Kapitels.
   e. Wird ein „in-line“-PERFORM verwendet, ist die Ausführung der PERFORM-Anweisung nach der letzten in der PERFORM-Anweisung enthaltenen Anweisung abgeschlossen.
7. Es gibt keine notwendige Verbindung zwischen prozedurname-1 und prozedurname-2, außer dass eine fortlaufende Reihe von Operationen, beginnend bei prozedurname-1, ausgeführt werden soll, die mit der Ausführung der mit prozedurname-2 bezeichneten Prozedur endet. Sogar GO TO- und PERFORM-Anweisungen können zwischen prozedurname-1 und dem Ende von prozedurname-2 auftreten. Wenn es zwei oder mehr logische Wege zum Rücksprungpunkt gibt, kann prozedurname-2 der Name eines Paragrafen sein, der eine EXIT-Anweisung enthält, auf die alle diese Ablaufwege führen müssen.
8. Alle nachfolgend beschriebenen Bezüge auf einen Bezeichner als Objekt der VARYING-, AFTER- und FROM-(aktueller Wert) Angaben gelten auch für Indizes.
   a. Ist index-1 oder index-3 angegeben, muss der Wert des zugehörigen Index zu Beginn der PERFORM-Anweisung auf eine vorkommende Zahl eines Tabellenelements gesetzt werden.
   b. Ist index-2 oder index-4 angegeben, muss der Wert des durch bezeichner-2 oder bezeichner-5 dargestellten Datenfeldes zu Beginn der PERFORM-Anweisung gleich sein dem einer mit index-2 oder index-4 verbundenen Tabellenelementnummer.

c. Die nachfolgende In- oder Dekrementierung von index-1 oder index-3 darf nicht enden bei einem Indexwert, der außerhalb des durch index-1 oder index-3 festgelegten Tabellenbereichs gesetzt ist; ausgenommen davon ist, dass bei Beendigung der PERFORM-Anweisung index-1 einen um einen Schritt vergrößerten oder verminderten Wert außerhalb des Tabellenbereichs enthalten darf.

d. Ist bezeichner-2 oder bezeichner-5 subskribiert, werden die Subskripte jedesmal ausgewertet, wenn der Inhalt des jeweiligen Bezeichners gesetzt oder erhöht (vermindert) wird.

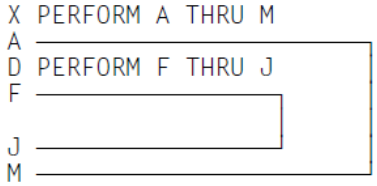
e. Ist bezeichner-3, bezeichner-4, bezeichner-6 oder bezeichner-7 subskribiert, werden die Subskripte jedesmal ausgewertet, wenn der Inhalt des jeweiligen Bezeichners gesetzt oder erhöht (vermindert) wird.

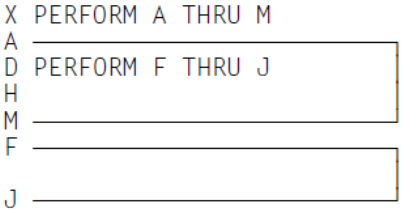
9. Anweisungen, die auf Grund einer EXIT PROGRAM-Anweisung ausgeführt werden, gelten nicht als Teil des Bereichs der PERFORM-Anweisung, wenn:

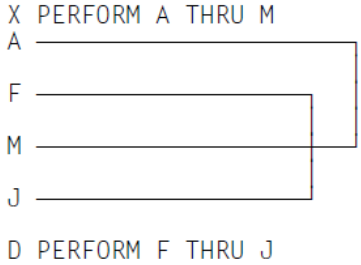
   a. die EXIT PROGRAM-Anweisung im selben Programm angegeben ist, in dem auch die PERFORM-Anweisung steht, und

   b. die EXIT PROGRAM-Anweisung innerhalb des Bereichs der PERFORM-Anweisungsteht.

10. Steht im Bereich einer PERFORM-Anweisung eine weitere PERFORM-Anweisung, muss der Bereich der eingebetteten PERFORM-Anweisung entweder völlig innerhalb oder völlig außerhalb des Bereichs der ersten PERFORM-Anweisung liegen. Infolgedessen darf eine PERFORM-Anweisung, deren Ausführung im Bereich einer anderen PERFORM-Anweisung beginnt, nicht zum Ausgang der anderen PERFORM-Anweisung führen; außerdem dürfen zwei oder mehr solcher PERFORM-Anweisungen keinen gemeinsamen Ausgang haben.
    Die folgenden Beispiele zeigen zugelassene PERFORM-Konstruktionen:







11. Ist TEST BEFORE angegeben oder angenommen, laufen folgende Vorgänge ab:

    Veränderung **eines** Bezeichners:

    - Zu Beginn der Ausführung der PERFORM-Anweisung wird der Inhalt von bezeichner-2 auf literal-1 oder auf den laufenden Wert von bezeichner-3 gesetzt.
    - Danach wird, wenn die Bedingung der UNTIL-Angabe falsch ist, die angegebene Anweisungsfolge einmal ausgeführt.
    - Der Wert von bezeichner-2 wird um den angegebenen Wert (Wert von literal-2 oder von bezeichner-4) vergrößert oder vermindert und bedingung-1 wird erneut geprüft.
    - Dieser Zyklus wird fortgesetzt, bis die Bedingung wahr ist; nun geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.
    - Ist bedingung-1 zu Beginn der Ausführung der PERFORM-Anweisung wahr, geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.

    Veränderung von **zwei** Bezeichnern:

- Zu Beginn der Ausführung der PERFORM-Anweisung wird der Inhalt von bezeichner-2 auf literal-1 oder auf den laufenden Wert von bezeichner-3 gesetzt.
- Danach wird der Inhalt von bezeichner-5 auf literal-3 oder auf den laufenden Wert von bezeichner-6 gesetzt.
- Anschließend wird bedingung-1 geprüft; ist sie wahr, geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.
- Ist bedingung-1 falsch, wird bedingung-2 geprüft.
- Ist bedingung-2 falsch, wird die angegebene Anweisungsfolge einmal ausgeführt.
- Hierauf wird der Inhalt von bezeichner-5 um den Wert von literal-4 oder den Inhalt von bezeichner-7 vergrößert (vermindert) und bedingung-2 erneut geprüft.
- Dieser Zyklus wird fortgesetzt, bis bedingung-2 wahr ist.
- Ist bedingung-2 wahr, wird bezeichner-2 um den Wert von literal-2 oder bezeichner-4 vergrößert (vermindert) und der Inhalt von bezeichner-5 auf literal-3 oder den momentanen Wert von bezeichner-6 gesetzt.
- bedingung-1 wird nochmals geprüft.
- Die PERFORM-Anweisung ist abgeschlossen, wenn bedingung-1 wahr ist; andernfalls wird der Zyklus solange fortgesetzt, bis bedingung-1 wahr ist.

Bei Beendigung der PERFORM-Anweisung enthält bezeichner-5 den Wert von literal-3 oder den momentanen Wert von bezeichner-6.
bezeichner-2 enthält einen Wert, der den letzten Inkrement- oder Dekrementschritt um 1 übersteigt, es sei denn, bedingung-1 war zu Beginn der PERFORM-Anweisung wahr. In diesem Fall enthält bezeichner-2 den Wert von literal-1 oder den momentanen Wert von bezeichner-3.

12. Ist TEST AFTER angegeben, laufen folgende Vorgänge ab:

    Veränderung **eines** Bezeichners:

    - Zu Beginn der Ausführung der PERFORM-Anweisung wird der Inhalt von bezeichner-2 auf literal-1 oder auf den laufenden Wert von bezeichner-3 gesetzt.
    - Danach wird die angegebene Anweisungsfolge einmal ausgeführt und bedingung-1 der UNTIL-Angabe geprüft.
    - Ist die Bedingung falsch, wird der Wert von bezeichner-2 um den angegebenen Wert (Wert von literal-2 oder von bezeichner-4) vergrößert oder vermindert und die angegebene Anweisungsfolge erneut ausgeführt.
    - Dieser Zyklus wird fortgesetzt, bis bedingung-1 geprüft und für wahr befunden wird; zu diesem Zeitpunkt geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.

    Veränderung von **zwei** Bezeichnern:

    - Zu Beginn der Ausführung der PERFORM-Anweisung wird der Inhalt von bezeichner-2 auf literal-1 oder auf den laufenden Wert von bezeichner-3 gesetzt.
    - Danach wird der Inhalt von bezeichner-5 auf literal-3 oder auf den laufenden Wert von bezeichner-6 gesetzt.
    - Anschließend wird die angegebene Anweisungsfolge ausgeführt.
    - Hierauf wird bedingung-2 ausgewertet.
    - Ist bedingung-2 falsch, wird der Inhalt von bezeichner-5 um den Wert von literal-4 oder den Inhalt von bezeichner-7 vergrößert (vermindert) und die angegebene Anweisungsfolge erneut ausgeführt.
    - Dieser Zyklus wird fortgesetzt, bis bedingung-2 wiederum geprüft und für wahr befunden wird.
    - Hierauf wird bedingung-1 geprüft.

- Ist bedingung-1 falsch, wird bezeichner-2 um den Wert von literal-2 oder bezeichner-4 vergrößert (vermindert) und der Inhalt von bezeichner-5 auf literal-3 oder den momentanen Wert von bezeichner-6 gesetzt.
- Anschließend wird die angegebene Anweisungsfolge erneut ausgeführt.
- Dieser Zyklus wird fortgesetzt, bis bedingung-1 geprüft und für wahr befunden wird; dann geht die Ablaufsteuerung zum Ende der PERFORM-Anweisung über.

Nach Beendigung der PERFORM-Anweisung enthält jedes durch die AFTER- oder VARYING-Angabe veränderte Datenfeld den Wert, den es nach dem Ende der letzten Ausführung der angegebenen Anweisungsfolge hatte.

13. Während der Ausführung der innerhalb der PERFORM-Anweisung angegebenen Anweisungsfolge wird jede Veränderung der VARYING-Variablen (bezeichner-2, index-1), BY-Variable (bezeichner-4), AFTER-Variablen (bezeichner-5, index-3) oder FROM-Variablen (bezeichner-3, index-2) berücksichtigt und beeinflusst die Ausführung der PERFORM-Anweisung.

    Wird der Inhalt von zwei Bezeichnern verändert, durchläuft bezeichner-5 einen vollständigen Zyklus (FROM, BY, UNTIL), jedesmal wenn der Inhalt von bezeichner-2 verändert wird.

    Wird der Inhalt von drei oder mehr Bezeichnern verändert, ist der Ablauf der gleiche wie bei zwei Bezeichnern, außer dass das durch jede AFTER-Angabe veränderte Datenfeld einen vollständigen Zyklus durchläuft, jedesmal wenn das durch die vorhergehende AFTER-Angabe veränderte Datenfeld in- oder dekrementiert wird.

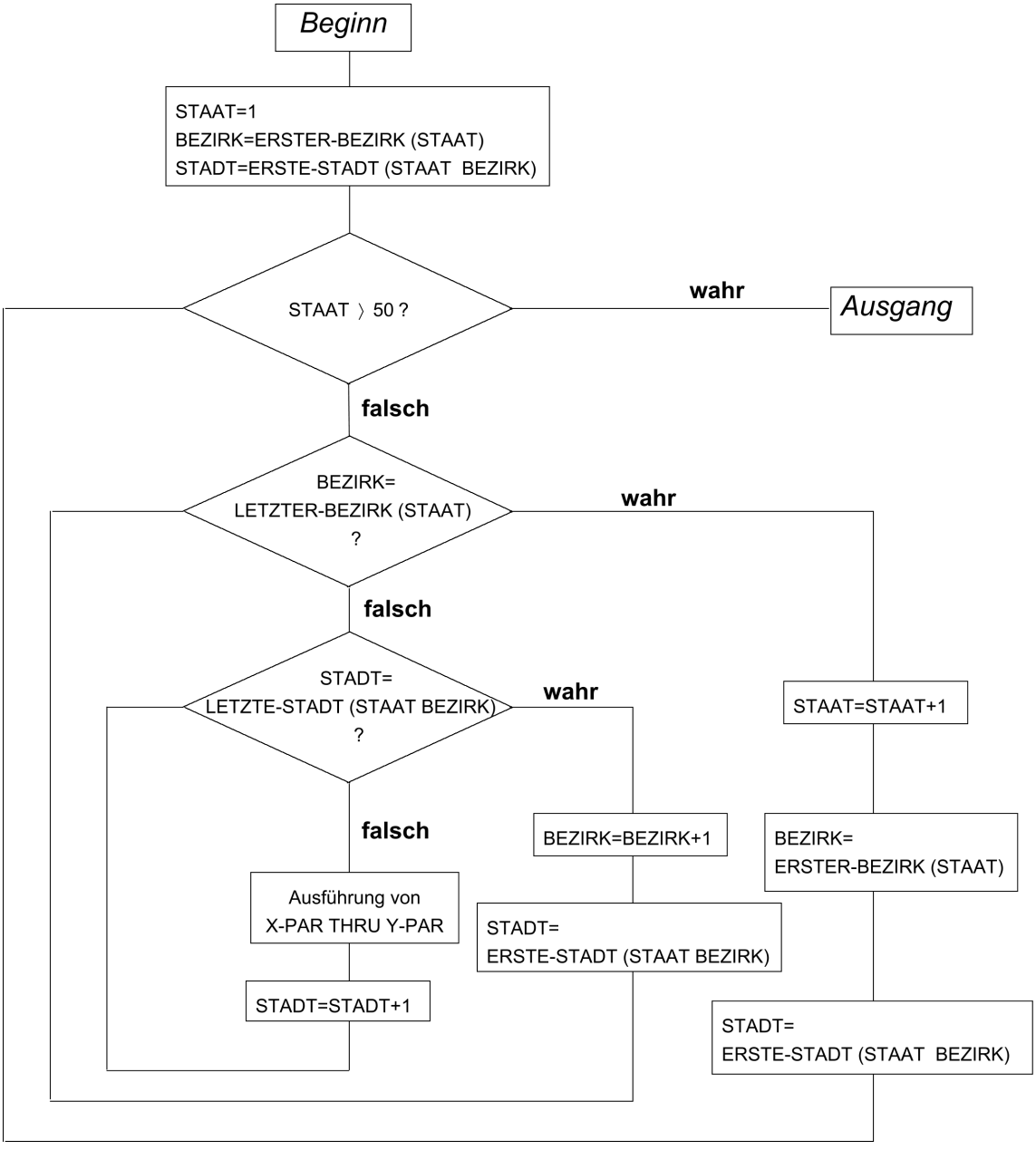
14. Dieser Punkt enthält Angaben zur Segmentierung (für weitere Angaben siehe unter Kapitel „Segmentierung").
    a. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer kleiner als die Angabe in der SEGMENT-LIMIT-Klausel ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
        - entweder Kapitel, die alle eine Segmentnummer kleiner 50 haben,
        - oder Kapitel, die als Ganzes in einem einzigen Segment enthalten sind, dessen Segmentnummer größer als 49 ist.
    b. Tritt eine PERFORM-Anweisung in einem Kapitel auf, dessen Segmentnummer größer als 49 ist, so darf sie in ihrem Bereich nur folgende Prozeduren enthalten:
        - entweder Kapitel, die alle die gleiche Segmentnummer haben wie das Kapitel, das die PERFORM-Anweisung enthält
        - oder Kapitel, die alle eine Segmentnummer kleiner 50 haben.
    c. Wird ein Prozedurname in einem Segment mit einer Segmentnummer größer 49 durch eine PERFORM-Anweisung angesprochen, die in einem Segment mit einer unterschiedlichen Segmentnummer liegt, so wird das angesprochene Segment in seinem Initialzustand zur Verfügung gestellt (d.h. also, dass alle durch ausgeführte ALTER-Anweisungen veränderte GO TO-Anweisungen in diesem Segment auf ihren Ursprungswert gesetzt werden).

Beispiel 8-65

```
    ...
01  STAATEN.
    02  BEZIRKE OCCURS 51 INDEXED STAAT.
        03  ERSTER-BEZIRK    PIC 9(3).
        03  LETZTER-BEZIRK   PIC 9(3).
        03  STAEDTE OCCURS 100 INDEXED BEZIRK.
            04  ERSTE-STADT  PIC 9(3).
            04  LETZTE-STADT PIC 9(3).
01  STADT                    PIC 9(3).
     ...
PROCEDURE DIVISION.
```

```
K1.
    PERFORM X-PAR THRU Y-PAR
        VARYING STAAT FROM 1 BY 1
            UNTIL STAAT IS GREATER THAN 50;
        AFTER BEZIRK FROM ERSTER-BEZIRK (STAAT) BY 1
            UNTIL BEZIRK IS EQUAL TO LETZTER-BEZIRK (STAAT)
        AFTER STADT FROM ERSTE-STADT (STAAT BEZIRK)
       UNTIL STADT IS EQUAL TO LETZTE-STADT (STAAT BEZIRK)
. . .
```

Ablaufplan:



Beispiel 8-66

für Format 3 und 4, WITH TEST AFTER/BEFORE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PWTA.
ENVIRONMENT DIVISION.
```

```
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 N PIC 9.
77 K PIC 9(3).
77 Z PIC 9(4).
77 E PIC 9(4).
PROCEDURE DIVISION.
P1 SECTION.
BERECHNUNG.
    MOVE 0 TO Z.
    DISPLAY "Einstellige Zahl als obere Grenze N eingeben" UPON T.
    ACCEPT N FROM T.
    PERFORM WITH TEST AFTER VARYING K FROM 1 BY 1 UNTIL K >= N
      COMPUTE E = K ** 3
      ADD E TO Z
    END-PERFORM
    DISPLAY "Ergebnis =" Z UPON T.
ENDE.
    STOP RUN.
```

Das Programm berechnet die n-te Summe der 3.Potenzen einer Ganzzahl K. Zunächst werden die Anweisungen COMPUTE und ADD ausgeführt, dann erst wird geprüft, ob die Abbruchbedingung erfüllt ist. Bei Angabe von TEST BEFORE wird zuerst die Abbruchbedingung geprüft. Ist K <= N, werden die „in-line“-Anweisungen ausgeführt. Ist K > N, wird die PERFORM-Anweisung mit END-PERFORM beendet.

8.10.33 RAISE-Anweisung

Funktion

Die RAISE-Anweisung bewirkt, dass ein bestimmter Ausnahmezustand ausgelöst wird.

Format

```
RAISE EXCEPTION ausnahmesituationsname
```

Syntaxregel

1. ausnahmesituationsname muss einer der in der Tabelle 45 aufgeführten Namen sein.

Allgemeine Regel

1. Der ausnahmesituationsname entsprechende Ausnahmezustand wird ausgelöst.

8.10.34 READ-Anweisung

Funktion

Durch die READ-Anweisung wird dem Programm

- bei sequenziellem Zugriff der nächste Datensatz einer Datei (für **sequenzielle, relative und indizierte Dateiorganisation**),
- bei wahlfreiem Zugriff (für **relative und indizierte Dateiorganisation**) ein durch den Schlüssel festgelegter Datensatz einer Datei

zur Verfügung gestellt.

Format 1 **sequenzieller Dateizugriff**.

Gilt für sequenziell organisierte Dateien und wird bei relativer und indizierter Dateiorganisation mit sequenziellem oder dynamischem Zugriffsmodus verwendet, um Datensätze sequenziell zu lesen.

Format 2 **wahlfreier Dateizugriff**.
Wird für relative und indizierte Dateiorganisation verwendet, bei wahlfreiem oder dynamischem Zugriffsmodus, um Datensätze wahlfrei (RELATIVE KEY bei **relativ organisierten Dateien**, bzw. RECORD KEY oder ALTERNATE RECORD KEY bei **indiziert organisierten Dateien**) zu lesen.

> Die Formaterweiterung (WITH NO LOCK) für beide Formate, die bei der Simultanverarbeitung wirksam wird, ist im Handbuch „COBOL2000 Benutzerhandbuch“ [1] beschrieben.

Format 1 sequenzieller Dateizugriff

```
READ dateiname [WITH NO LOCK] {NEXT | PREVIOUS } RECORD [INTO bezeichner]
    [AT END unbedingte-anweisung-1]
    [NOT AT END unbedingte-anweisung-2]
    [END-READ]
```

Syntaxregeln

1. In der READ-Anweisung bezieht sich das Empfangsfeld auf denselben Speicherbereich wie die Datensatzbeschreibung.
2. Falls keine USE-Prozedur für die Datei vorhanden ist, muss AT END in der READ-Anweisung angegeben werden.
3. PREVIOUS darf nur bei indizierter oder relativer Dateiorganisation angegeben werden.
4. Wenn im FILE-CONTROL-Paragrafen für die Datei ACCESS MODE RANDOM eingetragen ist, dürfen die Klauseln der Klauseln AT END, NOT AT END, NEXT und PREVIOUS nicht angegeben werden.
5. Wenn im FILE-CONTROL-Paragrafen für die Datei ACCESS MODE SEQUENTIAL eingetragen und weder NEXT noch PREVIOUS angegeben ist, wird NEXT angenommen.
6. Wenn die folgenden Bedingungen gleichzeitig erfüllt sind, wird NEXT angenommen:
   - Im FILE-CONTROL-Paragrafen ist für die Datei ACCESS MODE DYNAMIC eingetragen.
   - Weder NEXT noch PREVIOUS ist angegeben.
   - AT END oder NOT AT AND ist angegeben.

Allgemeine Regeln

Für **sequenzielle, relative und indizierte Dateiorganisation** gilt:

1. Bevor eine READ-Anweisung für eine Datei durchgeführt werden kann, muss eine OPEN-Anweisung mit der Angabe INPUT oder I-O vorausgegangen sein.
2. Die Ausführung der READ-Anweisung bewirkt, dass der Inhalt des Datenfeldes einer für diese Datei vorhandenen FILE STATUS-Klausel aktualisiert wird (siehe „FILE STATUS-Klausel").
3. Sind die Datensätze einer Datei mit mehr als einer Datensatzerklärung beschrieben, teilen sich diese Sätze automatisch denselben Speicherbereich; dies entspricht einer impliziten Redefinition des Speicherbereichs. Die Inhalte aller Datenfelder, die außerhalb der Grenzen des aktuellen Datensatzes liegen, sind nach der Ausführung der READ-Anweisung undefiniert.
4. INTO kann angegeben werden,
   - wenn der Dateierklärung nur eine Datensatzbeschreibung untergeordnet ist
   - wenn sowohl alle Datensatznamen, die dateiname zugeordnet sind, als auch das durch bezeichner repräsentierte Datenfeld Datengruppen oder alphanumerische Datenfelder sind.

   Ist bezeichner eine stark typisierte Datengruppe, so darf der Dateierklärung nur eine Datensatzbeschreibung untergeordnet sein. Diese Datensatzbeschreibung muss eine stark typisierte Datengruppe vom gleichen Typ wie bezeichner sein.
5. Falls nach einer READ-Anweisung ohne INTO-Angabe der Eingabebereich explizit angesprochen werden soll, ist der Benutzer für die Verwendung der richtigen (d.h. der Länge des eingelesenen Datensatzes entsprechenden) Datensatzerklärung verantwortlich.
6. Tritt keine Ausnahme-Bedingung auf, ist der Satz im Satzbereich verfügbar und eine implizite Übertragung auf Grund der INTO-Angabe wird ausgeführt.
7. Die Fortsetzung des Ablaufs der READ-Anweisung hängt davon ab, ob AT END oder NOT AT END angegeben ist (siehe Abschnitt „Ende-Bedingung").
8. Nach einer erfolglosen READ-Anweisung sind der Inhalt des zur Datei gehörenden Eingabebereichs sowie der Dateipositionsindikator undefiniert und der Ein-/Ausgabestatus zeigt an, dass kein gültiger nächster Satz zur Verfügung gestellt wurde.
9. Wenn die Anzahl der Zeichenpositionen eines gelesenen Datensatzes kleiner ist als die kleinste in den Datensatzbeschreibungen angegebene Länge, ist der Teil der Satzbereiches, der rechts vom letzten gültigen gelesenen Zeichen steht, undefiniert. Ist die Anzahl der Zeichenpositionen größer als die größte in den Datensatzbeschreibungen angegebene Länge, wird der Datensatz rechts von der maximalen Länge abgeschnit-ten. In beiden Fällen war die Leseoperation erfolgreich, aber es wird ein FILE STATUS-Wert gesetzt, der einen Satzlängenkonflikt anzeigt.
10. Die Ausführung einer READ-Anweisung mit INTO-Angabe ist gleichbedeutend mit:

    ```
    READ dateiname
    MOVE datensatzname TO bezeichner
    ```

    Die Übertragung findet entsprechend den Regeln für die MOVE-Anweisung ohne CORRESPONDING-Angabe statt.
    Der Datensatz steht nach Ausführung der READ-Anweisung mit INTO-Angabe sowohl im Eingabebereich als auch im durch bezeichner festgelegten Bereich zur Verfügung. Die Länge des Sendefeldes ist durch die Länge des eingelesenen Datensatzes bestimmt (siehe „RECORD-Klausel"). Enthält der Dateibeschreibungseintrag eine RECORD IS VARYING-Klausel, ist die implizit durchgeführte Übertragung eine Gruppenübertragung. Die implizite MOVE-Anweisung wird nicht durchgeführt, wenn die Ausführung der READ-Anweisung erfolglos war. Die Berechnung der Subskripte für bezeichner findet nach Ausführung der READ-Anweisung und unmittelbar vor der Übertragung statt.
11. Der Dateipositionsindikator bestimmt, ob ein Satz geliefert wird und ggf. welcher Satz geliefert wird:
    a. Wenn der Dateipositionsindikator „ungültig“ anzeigt, ist die READ-Anweisung erfolglos.

b. Wenn der Dateipositionsindikator durch eine vorausgehende OPEN- oder START-Anweisung gesetzt worden ist, wird folgendes Ergebnis geliefert:
   - Falls explizit oder implizit NEXT angegeben wurde, wird der erste Satz der Datei mit einer Satznummer oder einem Satzschlüssel Dateipositionsindikator geliefert.
   - Falls PREVIOUS angegeben wurde, wird der erste Satz der Datei mit einer Satznummer oder einem Satzschlüssel Dateipositionsindikator geliefert.

     Anmerkung:
     Nach OPEN liefert READ PREVIOUS bei relativen Dateien den ersten Satz, bei indizierten Dateien normalerweise die Ende-Bedingung.

c. Wenn der Dateipositionsindikator durch eine vorausgehende READ-Anweisung gesetzt worden ist, wird folgendes Ergebnis geliefert:
   - Falls explizit oder implizit NEXT angegeben wurde, wird der erste Satz der Datei mit einer Satznummer oder einem Satzschlüssel > Dateipositionsindikator geliefert.
   - Falls PREVIOUS angegeben wurde, wird der erste Satz der Datei mit einer Satznummer oder einem Satzschlüssel < Dateipositionsindikator geliefert.

Wenn ein Satz geliefert wird, steht er im Satzbereich der Datei dateiname zur Verfügung. Der Dateipositionsindikator wird auf die Satznummer bzw. auf den Satzschlüssel gesetzt. Wenn kein Satz geliefert wird, zeigt der Dateipositionsindikator „ungültig“ und an und es wird entsprechend Regel 5 verfahren.

Für **sequenzielle Dateiorganisation** gilt außerdem:

12. Nach Auftreten der Ende-Bedingung darf für diese Datei keine READ-Anweisung gegeben werden, bevor nicht eine erfolgreiche CLOSE-Anweisung, gefolgt von einer erfolgreichen OPEN-Anweisung, durchgeführt wurde.
13. Falls das physische Ende einer Spule während der Ausführung einer READ-Anweisung für eine auf mehreren Datenträgern gespeicherte Datei erkannt wird, finden die folgenden Operationen statt:
    a. Die Standarddatenträger-Endekennsatzroutinen und, falls eine entsprechende USE-Prozedur vorliegt, die Benutzerdatenträger-Endekennsatzroutinen werden durchlaufen. Dabei wird die Reihenfolge der Durchführung der beiden Prozeduren von den USE-Prozeduren festgelegt.
    b. Ein Datenträgerwechsel findet statt.
    c. Die Standard- und, falls angegeben, Benutzer-Anfangskennsatzroutinen werden ausgeführt. Wieder wird die Reihenfolge der Ausführung von den Angaben der USE-Prozedur festgelegt.

Für **relative und indizierte Dateiorganisation** gilt außerdem:

14. Wenn RELATIVE KEY angegeben ist, bewirkt die READ-Anweisung die Übertragung der relativen Satznummer des verfügbaren Satzes in das Datenfeld des relativen Satzschlüssels, entsprechend den Regeln für die MOVE-Anweisung.
15. Nach Auftreten der Ende-Bedingung darf für diese Datei keine READ-Anweisung gegeben werden, bevor nicht entweder mit START neu positioniert wurde oder die Datei erfolgreich geschlossen und wieder geöffnet wurde.
16. In einer indizierten Datei mit sequenziellem Zugriff ist die Reihenfolge der Datensätze, deren Schlüssel Duplikate zulässt, festgelegt durch die Reihenfolge, in der diese doppelten Schlüsselwerte durch WRITE- bzw REWRITE-Anweisungen erzeugt wurden. Beim Lesen mit explizitem oder impliziten NEXT beginnt das Lesen mit dem zuerst geschriebenen Satz, bei PREVIOUS mit dem zuletzt geschriebenen Satz.

Format 2 wahlfreier Dateizugriff

```
READ dateiname [WITH NO LOCK] RECORD [INTO bezeichner]
    [KEY IS datenname]
    [INVALID KEY unbedingte-anweisung-1]
```

```
[NOT INVALID KEY unbedingte-anweisung-2]

[END-READ]
```

Syntaxregeln

1. In der READ-Anweisung bezieht sich das Empfangsfeld auf denselben Speicherbereich wie die Datensatzbeschreibung.
2. Falls keine USE-Prozedur für die Datei vorhanden ist, muss INVALID KEY angegeben werden.
3. Die Angabe `KEY IS datenname` darf nur bei **indizierter Dateiorganisation** angegeben werden.
4. datenname muss der Name eines für diese Datei erklärten RECORD KEY oder ALTERNATE RECORD KEY-Datenfeldes sein.
5. datenname darf gekennzeichnet sein.

Allgemeine Regeln

1. Bevor eine READ-Anweisung für eine Datei durchgeführt werden kann, muss eine OPEN-Anweisung mit der Angabe INPUT oder I-O vorausgegangen sein.
2. Falls eine Datei mehr als einen logischen Satztyp enthält, teilen sich diese Datensätze automatisch den gleichen Speicherbereich; dies entspricht einer impliziten Redefinition des Speicherbereichs.
3. Die INTO-Angabe kann in folgenden Fällen gemacht werden:
   - wenn der Dateierklärung nur eine Datensatzbeschreibung untergeordnet ist
   - wenn sowohl alle Datensatznamen, die dateiname zugeordnet sind, als auch das durch bezeichner repräsentierte Datenfeld Datengruppen oder alphanumerische Datenfelder sind.

   Ist bezeichner eine stark typisierte Datengruppe, so darf der Dateierklärung nur eine Datensatzbeschreibung untergeordnet sein. Diese Datensatzbeschreibung muss eine stark typisierte Datengruppe vom gleichen Typ wie bezeichner sein.
4. Die Ausführung einer READ-Anweisung mit der INTO-Angabe ist gleichbedeutend mit:

   ```
   READ dateiname
   MOVE datensatzname TO bezeichner
   ```

   Die Übertragung findet entsprechend den Regeln für die MOVE-Anweisung ohne CORRESPONDING-Angabe statt. Der Datensatz steht nach Ausführung der READ-Anweisung mit INTO-Angabe sowohl im Eingabebereich als auch im durch bezeichner festgelegten Bereich zur Verfügung. Die Länge des Sendefeldes ist durch die Länge des eingelesenen Datensatzes bestimmt (siehe „RECORD-Klausel").War die READ-Anweisung nicht erfolgreich, so findet keine Übertragung statt.Die Berechnung der Indizes für bezeichner findet nach Ausführung der READ-Anweisung unmittelbar vor dem MOVE statt.
5. Falls nach einer READ-Anweisung ohne INTO-Angabe der Eingabebereich explizit angesprochen werden soll, ist der Benutzer für die Verwendung der richtigen (d.h. der Länge des eingelesenen Datensatzes entsprechenden) Datensatzerklärung verantwortlich.
6. Tritt während der Ausführung keine Schlüsselfehler-Bedingung auf, wird die INVALID KEY-Angabe, falls vorhanden, ignoriert, und folgende Vorgänge laufen ab:
   a. Der Ein-/Ausgabe-Zustand für dateiname wird aktualisiert.
   b. Tritt eine Ausnahme-Bedingung auf, die keine Schlüsselfehler-Bedingung ist, geht die Ablaufsteuerung zur USE-Prozedur über.
   c. Tritt keine Ausnahme-Bedingung auf, ist der Satz im Satzbereich verfügbar und eine implizite Übertragung auf Grund der INTO-Angabe wird ausgeführt. Der Ablauf verzweigt zum Ende der READ-Anweisung oder, falls angegeben, zu unbedingte-anweisung-2 der NOT INVALID KEY-Angabe. Im letzteren Fall wird die Ausführung gemäß den Regeln für die angegebene unbedingte Anweisung fortgesetzt. Je nach Art dieser Anweisung wird der Ablauf entweder in einem anderen Teil des Programms oder am Ende der READ-Anweisung fortgesetzt.

7. Nach einer erfolglosen READ-Anweisung sind der Inhalt des zur Datei gehörigen Eingabebereichs sowie der Dateipositionsindikator undefiniert.
8. Ist die Anzahl der Zeichenpositionen eines gelesenen Datensatzes kleiner als die kleinste in den Datensatzbeschreibungen angegebene Länge, ist der Teil der Satzbereiches, der rechts vom letzten gültigen gelesenen Zeichen steht, undefiniert. Ist die Anzahl der Zeichenpositionen größer als die größte in den Datensatzbeschreibungen angegebene Länge, wird der Datensatz rechts von der maximalen Länge abgeschnit-ten. In beiden Fällen war die Leseoperation erfolgreich, aber es wird ein FILE STATUS-Wert gesetzt, der einen Satzlängenkonflikt anzeigt.

   Nur für **relative Dateiorganisation** gilt:

9. Wenn eine Datei wahlfrei gelesen wird, wird nach einem Datensatz gesucht, dessen relative Satznummer gleich dem Wert des RELATIVE KEY-Datenfeldes dieser Datei ist. Falls kein solcher Datensatz in der Datei vorhanden ist, tritt eine Schlüsselfehler-Bedingung auf und die Ausführung der READ-Anweisung ist erfolglos (siehe „Schlüsselfehler-Bedingung").
10. Ist eine optionale Datei nicht vorhanden, tritt die Schlüsselfehler-Bedingung auf und die Ausführung der READ-Anweisung ist erfolglos.

    Nur für **indizierte Dateiorganisation** gilt:

11. Mit der Ausführung der READ-Anweisung wird die Datei auf den Wert des Bezugsschlüssels positioniert. Dieser Wert wird verglichen mit dem Wert des entsprechenden Datenfeldes im abgespeicherten Satz, bis der erste Satz mit dem gleichen Wert gefunden ist. Wird als Bezugsschlüssel ein Alternativschlüssel verwendet und sind für diesen Alternativschlüssel Duplikate vorhanden, so wird der zuerst geschriebene Satz mit diesem Schlüsselwert zur Verfügung gestellt. Der so gefundene Satz wird im Satzbereich von dateiname zur Verfügung gestellt. Kann kein Satz auf o.g. Weise gefunden werden, tritt eine Schlüsselfehler-Bedingung auf, und die Ausführung der READ-Anweisung ist erfolglos (siehe „Schlüsselfehler-Bedingung").
12. Wenn KEY angegeben ist, wird für die aktuelle Leseoperation datenname als Bezugsschlüssel festgelegt. Bei dynamischem Zugriff wird dieser Bezugsschlüssel auch für alle folgenden READ-Anweisungen vom Format 1 verwendet, bis für die Datei ein anderer Bezugsschlüssel vereinbart wird.
13. Wenn kein KEY angegeben ist, gilt der Primärschlüssel als Bezugsschlüssel. Bei dynamischem Zugriff wird dieser Bezugsschlüssel auch für alle folgenden READ-Anweisungen vom Format 1 verwendet, bis für die Datei ein anderer Bezugsschlüssel vereinbart wird.

8.10.35 RELEASE-Anweisung

Funktion

Die RELEASE-Anweisung kann nur bei einem Sortiervorgang benutzt werden. Sie übergibt Datensätze aus einer Eingabedatei an eine Sortierdatei.

Format

```
RELEASE sortierdatensatzname [FROM bezeichner]
```

Syntaxregeln

1. sortierdatensatzname muss der Name eines logischen Datensatzes in der zugehörigen Sortierdateibeschreibung sein.
2. sortierdatensatzname und bezeichner dürfen nicht den gleichen Internspeicherbereich belegen.

Allgemeine Regeln

1. Die Ausführung einer RELEASE-Anweisung bewirkt, dass der mit sortierdatensatzname bezeichnete Datensatz an die Sortierdatei zur Weiterverarbeitung durch die Sortierroutine des Systems übergeben wird.
2. Durch die FROM-Angabe entspricht die RELEASE-Anweisung einer MOVE-Anweisung, gefolgt von einer RELEASE-Anweisung.

   Wenn diese Angabe gemacht wurde, werden die Daten aus dem von bezeichner angegebenen Bereich in den durch sortierdatensatzname bezeichneten Bereich übertragen und dann an die Sortierdatei übergeben. Die Übertragung findet entsprechend den Regeln für eine MOVE-Anweisung ohne CORRESPONDING-Angabe statt.
3. Eine RELEASE-Anweisung darf nur innerhalb einer Eingabeprozedur im Zusammenhang mit einer SORT-Anweisung für die Sortierdatei, zu der sortierdatensatzname gehört, benutzt werden.
4. Nach Ausführung einer RELEASE-Anweisung ist der logische Datensatz im Satzbereich nicht länger verfügbar, außer wenn die zugehörige Sortierdatei in einer SAME RECORD AREA-Klausel vorkommt. Der logische Datensatz ist ebenfalls bei Programmablauf verfügbar als Datensatz anderer Dateien, die in der gleichen SAME RECORD AREA-Klausel angegeben sind, wie die zugehörige Sortierdatei; ebenso steht er der zu sortierdatensatzname gehörigen Datei zur Verfügung.

   Wenn die Steuerung an die Eingabeprozedur übergeht, besteht diese Datei aus allen jenen Datensätzen, die durch die Ausführung von RELEASE-Anweisungen übernommen wurden.
5. Nach Ausführung einer RELEASE-Anweisung mit FROM-Angabe steht der Datensatz nach wie vor in bezeichner zur Verfügung.

8.10.36 RESUME-Anweisung

Funktion

Die RESUME-Anweisung bewirkt eine Verzweigung zu einem expliziten Sprungziel oder zu der Anweisung hinter der Anweisung, die zur Ausführung einer USE-Prozedur geführt hat.

Format

```
RESUME AT {NEXT STATEMENT | prozedurname}
```

Syntaxregeln

1. Die RESUME-Anweisung darf nur in einer USE-Prozedur vom Format 2 oder 4 angegeben werden.
2. prozedurname muss außerhalb der Prozedurvereinbarungen (DECLARATIVES) definiert sein.

Allgemeine Regeln

1. Würde die Ausführung der RESUME-Anweisung zum Verlassen einer globalen USE-Prozedur führen, so wird die RESUME-Anweisung wie CONTINUE behandelt.
2. Die RESUME-Anweisung mit der NEXT STATEMENT-Angabe darf nur durchlaufen werden, wenn eine USE-Prozedur durch das Auslösen eines Ausnahmezustands oder eine Ein-Ausgabe-Ausnahmesituation aktiviert wurde, d.h. die Anweisung darf nicht durch eine PERFORM-Anweisung von außerhalb der Prozedurvereinbarungen ausgeführt werden.
3. Ist NEXT STATEMENT angegeben, so wird der Programmablauf bei einer angenommenen CONTINUE-Anweisung fortgesetzt. Diese wird, sofern nicht für eine Anweisung explizit anders definiert, direkt hinter der Anweisung, durch die die USE-Prozedur aktiviert wurde, angenommen. Für Anweisungen mit Bereichsbegrenzern wird die CONTINUE-Anweisung direkt hinter dem expliziten oder impliziten Bereichsbegrenzer angenommen.
4. Ist prozedurname angegeben, so wird der Programmlauf bei prozedurname fortgesetzt.

8.10.37 RETURN-Anweisung

Funktion

Die RETURN-Anweisung übernimmt einzelne Datensätze in sortierter Reihenfolge von der Sortierdatei.

Format

```
RETURN sortierdateiname RECORD [INTO bezeichner]
     AT END unbedingte-anweisung-1
    [NOT AT END unbedingte-anweisung-2]
    [END-RETURN]
```

Syntaxregeln

1. sortierdateiname muss in einer Sortierdateierklärung definiert sein.
2. Der zu bezeichner gehörige Internspeicherbereich und der zu sortierdateiname gehörige Datensatzbereich dürfen nicht den gleichen Internspeicherbereich belegen.

Allgemeine Regeln

1. Die Ausführung einer RETURN-Anweisung bewirkt, dass der nächste Datensatz gemäß der in der SORT-/MERGE-Anweisung durch die Schlüssel festgelegten Reihenfolge verfügbar gemacht wird. Danach kann er in den Datensatzbereichen der Sortierdatei verarbeitet werden.
2. Wenn für den logischen Datensatz einer Datei mehr als eine Datensatzbeschreibung angegeben ist, benutzen diese Datensätze automatisch den gleichen Internspeicherbereich; dies entspricht einer impliziten Redefinierung des Bereichs. Der Inhalt eines jeden Datenelementes, das außerhalb des Bereichs des aktuellen Datensatzes liegt, ist nach Ausführung einer RETURN-Anweisung undefiniert.
3. Die INTO-Angabe kann in zwei Fällen gemacht werden:
   - wenn der Sortier-/Misch-Dateierklärung nur eine Datensatzbeschreibung untergeordnet ist,
   - wenn alle Datensatznamen, die dateiname zugeordnet sind, und das durch bezeichner repräsentierte Datenfeld eine Datengruppe oder ein alphanumerisches Datenfeld beschreiben.

   Ist bezeichner eine stark typisierte Datengruppe, so darf der Sortier-/Misch-Dateierklärung nur eine Datensatzbeschreibung untergeordnet sein. Diese Datensatzbeschreibung muss eine stark typisierte Datengruppe vom gleichen Typ wie bezeichner sein.
4. Durch die INTO-Angabe entspricht die RETURN-Anweisung einer RETURN-Anweisung, die von einer MOVE-Anweisung gefolgt wird.
5. Wurde diese Angabe gemacht, so werden die Datensätze aus der Sortierdatei übernommen und dann in den durch bezeichner angegebenen Bereich übertragen;
   bezeichner darf nicht ein Datenfeld innerhalb des Datensatzes dieser Sortierdatei beschreiben. Die Übertragung erfolgt entsprechend den Regeln für eine MOVE-Anweisung ohne die CORRESPONDING-Angabe. Die Größe des aktuellen Datensatzes wird gemäß den Regeln der RECORD-Klausel bestimmt (siehe „RECORD-Klausel").
   Enthält die Sortierdateierklärung eine RECORD IS VARYING-Klausel, ist der implizite MOVE eine Gruppenübertragung.
6. Die anschließende MOVE-Anweisung wird nicht ausgeführt, wenn eine AT END-Bedingung auftritt.
7. Eine angegebene Subskribierung oder Indizierung für bezeichner wird berechnet, nachdem der Datensatz übernommen wurde und unmittelbar bevor er in den Datenbereich übertragen wird.
8. Wurde die INTO-Angabe benutzt, stehen die Daten sowohl im Datensatzeingabebereich als auch im durch bezeichner angegebenen Datenbereich zur Verfügung.

9. Die Ausführung der RETURN-Anweisung bewirkt, dass der nächste vorhandene Satz aus der durch sortierdateiname repräsentierten Datei so, wie er durch die Schlüssel in der SORT- oder MERGE-Anweisung bezeichnet ist, im Satzbereich zur Verfügung steht.
   Wenn zur Ausführungszeit einer RETURN-Anweisung kein nächster logischer Datensatz mehr existiert, tritt die Ende-Bedingung ein. Die Ausführung wird fortgesetzt entsprechend den Regeln für jede in unbedingte-anweisung-1 angegebene Anweisung. Wird dabei ein Prozedursprung oder eine Bedingungsanweisung ausgeführt, die einen expliziten Übergang der Steuerung bewirkt, erfolgt dieser Übergang entsprechend den Regeln für diese Anweisung; im anderen Fall geht nach Ende der Ausführung von unbedingte-anweisung-1 die Steuerung zum Ende der RETURN-Anweisung über, und die NOT AT END-Angabe wird, falls vorhanden, ignoriert. Tritt die Ende-Bedingung auf, ist die Ausführung der RETURN-Anweisung erfolglos, und der Inhalt des Satzbereichs von sortierdateiname ist unbestimmt. Nach der Ausführung von unbedingte-anweisung-1 kann keine RETURN-Anweisung als Teil der laufenden Ausgabe-Prozedur ausgeführt werden.
10. Tritt während der Ausführung einer RETURN-Anweisung keine Ende-Bedingung auf, geht die Steuerung, nachdem der Satz verfügbar gemacht wurde und jede in Verbindung mit einer INTO-Angabe erforderliche implizite Übertragung ausgeführt wurde, zu unbedingte-anweisung-2 über, falls NOT AT END angegeben wurde; anderenfalls geht die Steuerung zum Ende der RETURN-Anweisung über.
11. Eine RETURN-Anweisung darf nur innerhalb der Ausgabeprozedur angegeben werden, die zu einer SORT-/MERGE-Anweisung für sortierdateiname gehört.

8.10.38 REWRITE-Anweisung

Funktion

Mit Hilfe der REWRITE-Anweisung können Datensätze einer Plattenspeicherdatei ersetzt werden.

Format 1 für sequenzielle Dateiorganisation

```
REWRITE datensatzname [FROM bezeichner] [END-REWRITE]
```

Format 2 für relative und indizierte Dateiorganisation

```
REWRITE datensatzname [FROM bezeichner]
    [INVALID KEY unbedingte-anweisung-1]
    [NOT INVALID KEY unbedingte-anweisung-2]
    [END-REWRITE]
```

Syntaxregeln

1. datensatzname und bezeichner dürfen sich nicht auf den gleichen Speicherbereich beziehen.
2. datensatzname muss einer Dateierklärung (FD) der DATA DIVISION des Programmes zugeordnet sein und darf gekennzeichnet werden.
3. INVALID KEY muss bei relativ und indiziert organisierten Dateien mit wahlfreiem Zugriff angegeben werden, es sei denn, es wurde eine entsprechende USE-Prozedur vereinbart. INVALID KEY darf nicht angegeben werden bei relativ organisierten Dateien mit sequenziellem Zugriff.
4. Für zeilensequenzielle Dateien ist die REWRITE-Anweisung nicht zulässig.

Allgemeine Regeln

Für **sequenzielle, relative und indizierte Dateiorganisation** gilt:

1. datensatzname bezeichnet den Datensatz, der ersetzt werden soll.
2. Die mit datensatzname verknüpfte Datei muss eine Plattenspeicherdatei sein und muss bei Ausführung der REWRITE-Anweisung mit I-O eröffnet worden sein.
3. Die Ausführung einer REWRITE-Anweisung mit der Angabe FROM entspricht den folgenden Anweisungen:

   ```
   MOVE bezeichner TO datensatzname
   der gleichen REWRITE-Anweisung ohne FROM Angabe
   ```

   Der durch datensatzname beschriebene Inhalt des Speicherbereichs vor Ausführung der impliziten MOVE-Anweisung hat keine Bedeutung für die Ausführung der REWRITE-Anweisung.

   Bei Verwendung der FROM-Angabe werden die Daten von bezeichner nach datensatzname übertragen und dann in die entsprechende Datei ausgegeben.Mit bezeichner kann jeder Datenbereich bezeichnet werden, der außerhalb der gerade angesprochenen Dateierklärung liegt.

   Die Datenübertragung durch die implizite MOVE-Anweisung findet entsprechend den Regeln der MOVE-Anweisung ohne die CORRESPONDING-Angabe statt. Nach Ausführung der REWRITE-Anweisung steht der Satzinhalt nach wie vor im durch bezeichner beschriebenen Bereich zur Verfügung, jedoch nicht in dem durch datensatzname bezeichneten Bereich.
4. Für Dateien im sequenziellen Zugriffsmodus gilt:
   a. Einer REWRITE-Anweisung muss eine erfolgreich abgelaufene READ-Anweisung als letzte Ein-/Ausgabe-Anweisung für die zugehörige Datei vorangegangen sein.
   b. Bei Ausführung der REWRITE-Anweisung wird der durch die vorhergehende READ-Anweisung zur Verfügung gestellte Datensatz in der Datei ersetzt.

c. Der Inhalt des durch die RELATIVE KEY-Angabe (bei **relativ organisierten** Dateien) bzw. durch die RECORD-KEY-Klausel (bei **indiziert organisierten** Dateien) angegebenen Datenfeldes darf zwischen der READ- und REWRITE-Anweisung nicht geändert werden.

5. Einer REWRITE-Anweisung muss eine erfolgreich abgelaufene READ-Anweisung als letzte Ein-/Ausgabe-Anweisung für die zugehörige Datei vorangegangen sein.
6. Bei Ausführung der REWRITE-Anweisung wird der durch die vorhergehende READ-Anweisung zur Verfügung gestellte Datensatz in der Datei ersetzt.
7. Der Datensatz, der durch eine erfolgreich abgelaufene REWRITE-Anweisung zurückgeschrieben wurde, steht im Satzbereich nicht mehr zur Verfügung; eine Ausnahme stellt die Verwendung der SAME RECORD AREA-Klausel dar. In diesem Fall steht der Datensatz sowohl allen anderen Dateien, die in der SAME RECORD AREA-Klausel aufgeführt wurden, als auch der gerade bearbeiteten Datei als Datensatz zur Verfügung.
8. Die Ausführung einer REWRITE-Anweisung bewirkt, dass der Inhalt des Datenfeldes aktualisiert wird, das in der FILE STATUS-Klausel der zugehörigen Dateierklärung angegeben wurde (siehe auch „FILE STATUS-Klausel").

Für **relative und indizierte Dateiorganisation** gilt:

9. Die Länge des zu ersetzenden Datensatzes kann geändert werden.
10. Für Dateien im wahlfreien oder dynamischen Zugriffsmodus wird der Datensatz ersetzt, der bei relativ organisierten Dateien durch den Inhalt des mit der Datei verknüpften RELATIVE KEY-Datenfeldes angesprochen wurde, bei indiziert organisierten Dateien durch den Inhalt des durch RECORD KEY angegebenen Datenfeldes.
11. Die Anzahl der Zeichenpositionen in dem durch datensatzname bezeichneten Datensatz darf nicht größer als die größte und nicht kleiner als die kleinste Anzahl von Zeichenpositionen sein, die laut RECORD IS VARYING-Klausel für den Datensatz erlaubt ist. Andernfalls ist die Ausführung der REWRITE-Anweisung erfolglos, die Aktualisierungsoperation findet nicht statt, der Inhalt des Satzbereichs bleibt unbeeinflusst und der Ein-/Ausgabe-Zustand für die entsprechende Datei wird auf den Wert gesetzt, der das Überschreiten der Satzlängengrenzen anzeigt (siehe „Ein-/Ausgabe-Zustand").

Nur für **sequenzielle Dateiorganisation** gilt:

12. Die Länge des durch datensatzname angesprochenen Datensatzes muss gleich der Länge des zu ersetzenden Datensatzes der Datei sein.
13. Ist die Anzahl der Zeichenpositionen von datensatzname und des zu ersetzenden Datensatzes nicht gleich, ist die Ausführung der REWRITE-Anweisung erfolglos, die Aktualisierung findet nicht statt, der Inhalt des Satzbereichs bleibt unberührt und der Ein-/Ausgabe-Zustand der Datei zeigt das Überschreiten der Bereichsgrenzen an.
14. Das Zurückschreiben eines Datensatzes bewirkt erst dann eine Veränderung des Satzinhaltes auf der zugehörigen Plattenspeicherdatei, wenn der nächste Block der Datei gelesen oder die Datei geschlossen wird.

Nur für **relative Dateiorganisation** gilt:

15. Falls die Datei keinen Datensatz enthält, der dem Inhalt des Datenfeldes des RELATIVE KEY entspricht, tritt die Schlüsselfehler-Bedingung auf (siehe „Schlüsselfehler-Bedingung"). In diesem Fall ist die Ausführung der REWRITE-Anweisung erfolglos, die Aktualisierung findet nicht statt, die Inhalte des Satzbereichs bleiben unberührt und der Ein-/Ausgabe-Zustand der Datei wird auf einen Wert gesetzt, der die Schlüsselfehler-Bedingung anzeigt.
16. Die Fortsetzung des Ablaufs der REWRITE-Anweisung hängt davon ab, ob INVALID KEY oder NOT INVALID KEY angegeben ist (siehe „Schlüsselfehler-Bedingung").

17. Beim Zurückschreiben eines Datensatzes ist darauf zu achten, dass im ersten Byte des Satzes nicht der Wert X’FF’ enthalten ist, da dies das logische Löschen des Satzes bewirkt.

    Nur für **indizierte Dateiorganisation** gilt:

18. Die Schlüsselfehler-Bedingung tritt unter einer der folgenden Situationen auf:
    a. im sequenziellen Zugriffsmodus: wenn der Primärschlüsselwert des zu ersetzenden Datensatzes nicht gleich dem Primärschlüsselwert des letzten aus der Datei gelesenen Datensatzes ist,
    b. im dynamischen oder wahlfreien Zugriffsmodus: wenn der Primärschlüsselwert des zu ersetzenden Datensatzes nicht gleich ist dem Primärschlüsselwert irgendeines Datensatzes der Datei,
    c. wenn der Alternativschlüsselwert des zu ersetzenden Satzes, für den Duplikate nicht zugelassen sind, gleich ist dem Wert des zugehörigen Datenfeldes eines schon in der Datei vorhandenen Datensatzes.
19. Für einen Datensatz mit Alternativschlüssel wird die REWRITE-Anweisung folgendermaßen ausgeführt:
    a. Ist der Wert eines bestimmten Alternativschlüssels unverändert, bleibt die Reihenfolge der Leseoperationen unverändert, falls dieser Schlüssel der Bezugsschlüssel ist.
    b. Ist der Wert eines bestimmten Alternativschlüssels verändert, kann die Reihenfolge der Leseoperationen für diesen Satz, geändert werden, wenn sein Schlüssel der Bezugsschlüssel ist. Sind Duplikate zugelassen, ist der Satz der logisch letzte aus der Gruppe der Duplikate, die denselben Alternativschlüsselwert besitzen wie der zu ersetzende Datensatz.

8.10.39 SEARCH-Anweisung

Funktion

Die SEARCH-Anweisung wird verwendet, um in einer Tabelle nach einem Element zu suchen, das eine angegebene Bedingung erfüllt, und um den Wert des zugehörigen Index auf die entsprechende Tabellenelementnummer zu setzen.

Format 1	Damit kann eine Tabelle sequenziell durchsucht werden.
Format 2	Damit kann eine Tabelle binär durchsucht werden.

Format 1

```
SEARCH bezeichner-1 [VARYING {index-1 | bezeichner-2}]
    [AT END unbedingte anweisung-1]
    {WHEN bedingung-1 {unbedingte anweisung-2 | NEXT SENTENCE}}...
[END-SEARCH]
```

Syntaxregeln

1. bezeichner-1 legt die Tabelle fest, die durchsucht werden soll.
2. Die Datenerklärung von bezeichner-1 muss eine OCCURS-Klausel mit einer INDEXED BY-Angabe enthalten.
3. bezeichner-1 darf weder subskribiert noch indiziert noch einer Teilfeldselektion unterzogen werden.
4. index-1 oder bezeichner-2 gibt ein Feld an, dessen Wert während der Ausführung der SEARCH-Anweisung verändert werden soll.
5. index-1 kann einer der Indizes für bezeichner-1 sein, oder ein Index für eine andere Tabelle.
6. bezeichner-2 muss als ein Indexdatenfeld (USAGE IS INDEX) beschrieben oder ein numerisches Festpunktdatenfeld sein, das als ganze Zahl beschrieben ist.
7. Die AT END-Angabe bezeichnet eine Anweisung, die ausgeführt werden soll, falls die Suche erfolglos verläuft.
8. Jede Bedingung (bedingung-1, bedingung-2,...) muss eine zulässige Bedingung sein (siehe „Bedingungen").
9. bedingung-1... geben die Bedingungen an, die während der Ausführung der SEARCH-Anweisung geprüft werden.
10. unbedingte anweisung-2... oder NEXT SENTENCE gibt an, was getan werden soll, wenn die zugehörige WHEN-Bedingung erfüllt ist:
    Die Kontrolle geht über auf die unbedingte Anweisung oder auf den nächsten Programmsatz (das ist die Anweisung, die der SEARCH-Anweisung folgt), je nachdem, welche Möglichkeit angegeben wurde.
11. Das sequenzielle Durchsuchen einer Tabelle beginnt bei dem Tabellenelement, auf das der zur Suche verwendete Index von bezeichner-1 verweist.
12. Entspricht der Wert dieses Index von bezeichner-1 zu Beginn der SEARCH-Anweisung einer größeren als der höchsten erlaubten Tabellenelementnummer von bezeichner-1, so wird die Suche sofort abgebrochen. Ist der AT END-Zusatz angegeben, wird die unbedingte anweisung-1 ausgeführt. Fehlt der Zusatz, wird mit der nächsten Anweisung fortgesetzt.
13. Entspricht der Wert des zur Suche verwendeten Index von bezeichner-1 zu Beginn der SEARCH-Anweisung einer erlaubten Tabellenelementnummer von bezeichner-1, findet die sequenzielle Suche folgendermaßen statt:
    a. Die WHEN-Bedingungen werden in der Reihenfolge ausgewertet, in der sie angegeben sind.

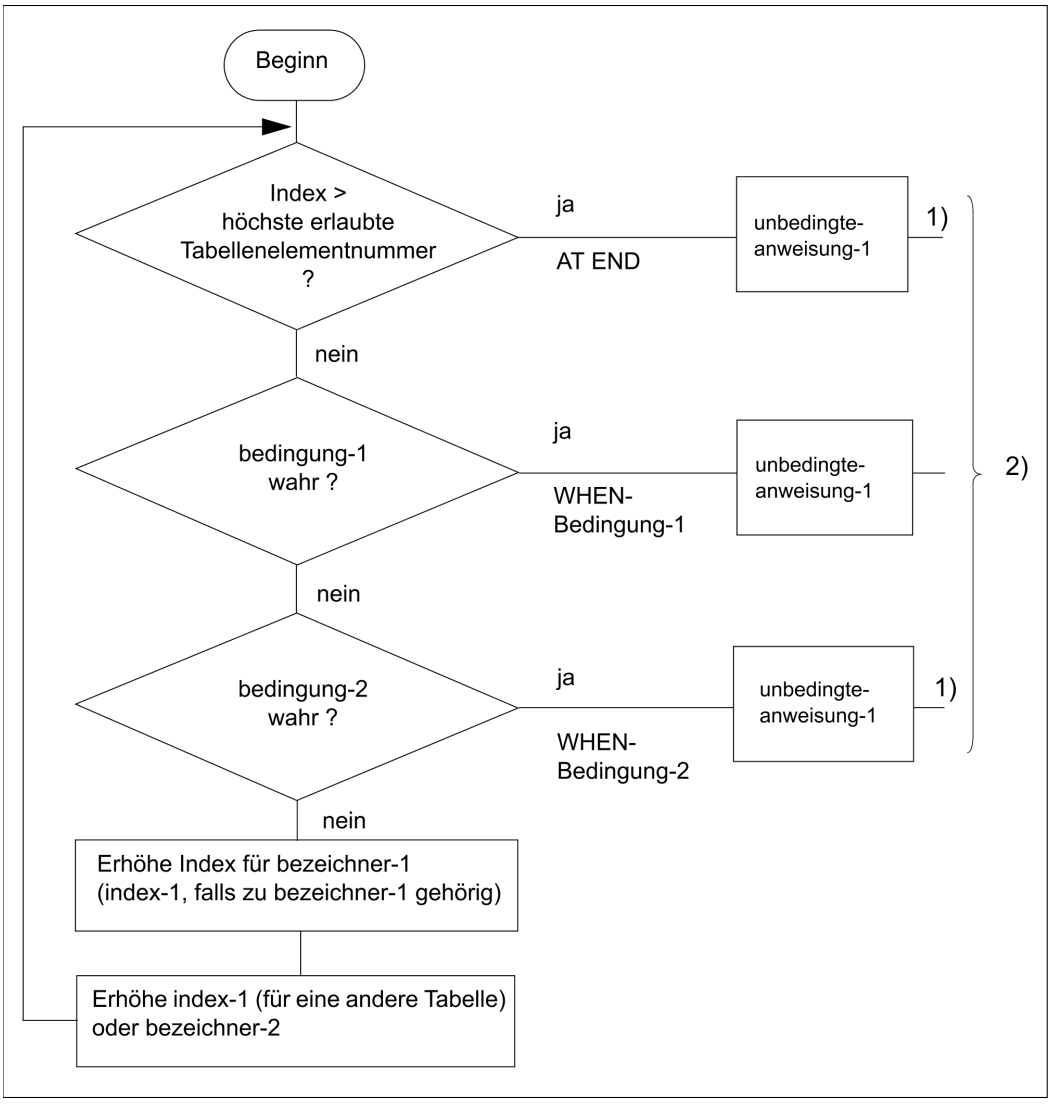
b. Ist keine der Bedingungen erfüllt, wird der Index für bezeichner-1 um soviel erhöht, dass er auf das nächste Tabellenelement verweist, und Punkt a) wird wiederholt. Dies geschieht nicht, wenn der neue Wert des Index einer nicht mehr zulässigen Tabellenelementnummer entspricht. In diesem Fall wird Punkt d) ausgeführt.

c. Ist eine WHEN-Bedingung erfüllt, wird die Suche beendet. Der Index verweist auf das Tabellenelement, das die Bedingung erfüllt. Die mit der Bedingung verknüpfte unbedingte Anweisung wird ausgeführt.

d. Ist das Ende der Tabelle erreicht, ohne dass eine WHEN-Bedingung erfüllt ist, wird die Suche beendet. Ist AT END angegeben, wird die unbedingte anweisung-1 ausgeführt. Fehlt der Zusatz, wird mit der nächsten Anweisung fortgesetzt.

14. Eine mehrdimensionale Tabelle kann durchsucht werden, wenn bezeichner-1 ein Datenfeld ist, das einem Datenfeld mit einer OCCURS-Klausel untergeordnet ist. In die-sem Fall muss für jede Dimension der Tabelle ein Index durch die INDEXED BY-Angabe der OCCURS-Klausel angegeben sein. Durch die Ausführung der SEARCH-Anweisung wird nur der Wert des zu bezeichner-1 gehörenden Index und, wenn angegeben, der Wert von index-1 oder bezeichner-2 verändert. Um alle Einträge einer mehrdimensionalen Tabelle durchsuchen zu können, muss die SEARCH-Anweisung für alle möglichen Werte der übergeordneten Indizes ausgeführt werden.
15. Durch SET-Anweisungen oder mit PERFORM VARYING sind die entsprechenden Indizes vor Ausführung der SEARCH-Anweisungen mit den gewünschten Werten vorzubesetzen (siehe Beispiel 8-69).
16. Endet in der AT END-Angabe und in den WHEN-Bedingungen keine der angegebenen Anweisungen mit einer GO TO-Anweisung, wird der Programmlauf nach Ausführung der unbedingten Anweisung fortgesetzt.
17. Ist in der VARYING-Angabe index-1 angegeben, findet Folgendes statt:
    a. Ist index-1 einer der Indizes für bezeichner-1, wird er für die Suche verwendet. Andere Indizes werden nicht erhöht.
    b. Ist index-1 ein Index für eine andere Tabelle, wird der erste (oder einzige) Index von bezeichner-1 für die Suche verwendet. Wird der Index von bezeichner-1 erhöht, wird gleichzeitig index-1 erhöht, so dass auch er auf das nächste Element seiner Tabelle verweist.
18. Ist in der VARYING-Angabe bezeichner-2 angegeben, verhält es sich wie folgt:
    a. Der erste (oder einzige) Index von bezeichner-1 wird für die Überprüfung der Tabellengrenzen verwendet.
    b. Wird der Index von bezeichner-1 erhöht, wird gleichzeitig bezeichner-2 erhöht.
    c. bezeichner-2 wird um 1 erhöht, wenn er ein numerisches Datenfeld bezeichnet.
    d. Ist bezeichner-2 ein Indexdatenfeld, wird es um denselben Wert erhöht wie der Index von bezeichner-1.

    Wird die VARYING-Angabe nicht verwendet, wird der erste (oder einzige) Indexname von bezeichner-1 (d. h. der, der in der Datenerklärung von bezeichner-1 in der INDEXED BY-Angabe steht) für die Suche verwendet.
19. Ist END-SEARCH angegeben, darf nicht die NEXT SENTENCE-Angabe gemacht werden.
20. Der Bereich einer SEARCH-Anweisung darf folgendermaßen begrenzt werden:
    a. mit END-SEARCH auf derselben Schachtelungsebene,
    b. mit einem Punkt am Ende der SEARCH-Anweisung,
    c. mit der ELSE- oder END-IF-Angabe einer vorhergehenden IF-Anweisung.



Suchvorgang für eine SEARCH-Anweisung gemäß Format 1

[1] Diese Operationen werden nur ausgeführt, wenn sie in der SEARCH-Anweisung ange-geben sind.
[2] Nach jeder dieser Anweisungen wird der Programmablauf bei der nächsten Anweisung fortgesetzt, es sei denn, die unbedingte Anweisung endet mit einer GO TO-Anweisung.

Beispiel 8-67

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUCHEN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ZEICHEN-TEST        PIC X VALUE LOW-VALUE.
    88 ZEICHEN-GEFUNDEN VALUE HIGH-VALUE.
01  ZEICHEN-EIN PICTURE A.
01  ZEICHEN-GEWICHTS-TAB.
        03  ZEICHEN-TABELLE OCCURS 26 TIMES INDEXED BY PI
            VALUE FROM (1)                          "A01" "B03"
               "C03" "D02" "E01" "F04" "G02" "H04" "I01" "J08"
               "K05" "L01" "M03" "N01" "O01" "P03" "Q10" "R01"
```

```
                "S01" "T01" "U01" "V04" "W04" "X08" "Y04" "Z10".
04  ZEICHEN PICTURE A.
04  WERT PICTURE 99.
PROCEDURE DIVISION.
MAIN SECTION.
ZEICHEN-SUCHEN.
    PERFORM UNTIL ZEICHEN-GEFUNDEN
      ACCEPT ZEICHEN-EIN FROM T
      SET PI TO 1
      SEARCH ZEICHEN-TABELLE VARYING PI
         AT END DISPLAY "Zeichen nicht alphabetisch" UPON T
                EXIT TO TEST OF PERFORM
         WHEN ZEICHEN (PI) = FUNCTION UPPER-CASE (ZEICHEN-EIN)
              SET ZEICHEN-GEFUNDEN TO TRUE
      END-SEARCH
    END-PERFORM.
GEFUNDEN.
    DISPLAY ZEICHEN (PI) " ist " WERT (PI) " zugeordnet" UPON T.
    STOP RUN.
```

Hier besteht die Tabelle ZEICHEN-TABELLE aus 26 Elementen.

Jedes Element enthält einen Buchstaben des Alphabets, gefolgt von einer Zahl, die mit dem Buchstaben verbunden ist.
Die Tabelle wird mit Hilfe der Indizes LI und PI indiziert.
Die SEARCH-Anweisung sucht in der Tabelle nach demjenigen Element ZEICHEN, das mit dem aktuellen Inhalt des Feldes ZEICHEN-EIN übereinstimmt. Der bei der Suche verwendete Index ist PI.
Die Suche beginnt am Anfang der Tabelle, da PI auf das erste Tabellenelement zeigt.Ist die Suche erfolgreich, wird die Anweisung GO TO GEFUNDEN ausgeführt. In diesem Fall zeigt der Index PI auf das Element, das die Bedingung erfüllt. Wenn z.B. ZEICHEN-EIN den Buchstaben B enthält, zeigt der Index auf das zweite Tabellenelement.
Ist die Suche erfolglos, wird die Anweisung ausgeführt, die der AT END-Angabe folgt.

Format 2

```
SEARCH ALL bezeichner-1 [AT END unbedingte-anweisung-1]

       WHEN bedingung {unbedingte-anweisung-2 | NEXT STATEMENT}

   [END-SEARCH]
```

Syntaxregeln

1. bezeichner-1 legt die Tabelle fest, die durchsucht werden soll.
2. Die Datenerklärung von bezeichner-1 muss eine OCCURS-Klausel mit der INDEXED BY-Angabe und der ASCENDING-/DESCENDING-Angabe enthalten.
3. bezeichner-1 darf weder subskribiert noch indiziert noch einer Teilfeld-Selektion unterzogen werden.
4. Die AT END-Angabe gibt die Anweisung an, die ausgeführt werden soll, wenn bedingung-1 für keinen Indexwert im erlaubten Bereich erfüllt werden kann (siehe Regel 10).
5. bedingung gibt die Bedingung an, die während der Ausführung der SEARCH-Anweisung geprüft wird.
6. bedingung muss eine der folgenden Arten von Bedingungen sein (siehe auch unter „Bedingungen"):
   a. Eine Vergleichsbedingung mit EQUAL TO oder dem Gleichheitszeichen (=) als Vergleichsoperator. Es muss entweder das Subjekt oder das Objekt der Vergleichsbedingung (aber nicht beide gleichzeitig) nur aus einem der Datennamen bestehen, die in der ASCENDING-/DESCENDING-Angabe von

bezeichner-1 angegeben sind. Jeder Datenname muss mit dem ersten Index von bezeichner-1 indiziert sein. Er darf gekennzeichnet, aber nicht einer Teilfeldselektion unterzogen worden sein.

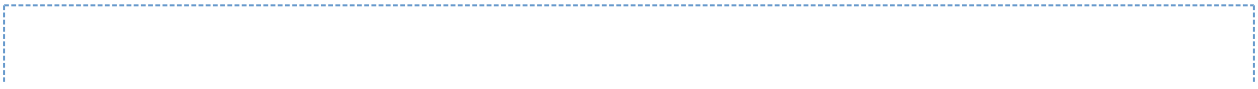
b. Eine Bedingungsnamen-Bedingung, in der die VALUE-Klausel, die den Bedingungsnamen beschreibt, nur ein einziges Literal enthält.
Der Bedingungsname muss mit dem ersten Index von bezeichner-1 indiziert und darf gekennzeichnet sein. Die Bedingungsvariable, die mit dem Bedingungsnamen verknüpft ist, muss einer der Datennamen sein, die in der ASCENDING-/DESCENDING-Angabe von bezeichner-1 angegeben sind.

c. Eine zusammengesetzte Bedingung, die aus einfachen Bedingungen der oben beschriebenen Arten gebildet wurde und als einzigen Verknüpfer nur AND enthält.

7. Jeder Datenname, der in der ASCENDING-/DESCENDING-Angabe von bezeichner-1 auftritt, kann in bedingung geprüft werden; dabei müssen alle Datennamen, die in der ASCENDING-/DESCENDING-Angabe vor dem geprüften Datennamen stehen, auch in einer Bedingung vorkommen. Andere Prüfungen können in bedingung nicht gemacht werden.
8. unbedingte anweisung-2 oder NEXT SENTENCE gibt an, was getan werden soll, wenn bedingung erfüllt ist. Der Programmlauf wird mit unbedingte anweisung-2 oder mit der Anweisung, die der SEARCH-Anweisung folgt, fortgesetzt.
9. Der **erste** Index von bezeichner-1 wird für die Suche verwendet. Dieser Index muss nicht mit einer SET-Anweisung initialisiert werden, da sein Anfangswert für den Suchvorgang belanglos ist.
10. Die SEARCH ALL-Anweisung wird folgendermaßen ausgeführt, wobei die Tabelle in aufsteigender bzw. absteigender Reihenfolge der in der ASCENDING-/DESCENDING-Angabe angeführten Schlüsselfelder geordnet sein muss:
    a. Während der Suche wird der Wert des Index von bezeichner-1 verändert.
    b. Dabei ist der Wert niemals kleiner als der, der dem ersten und niemals größer als der, der dem letzten Tabellenelement entspricht.
    c. Kann bedingung für keinen Indexwert im erlaubten Bereich erfüllt werden, und ist die AT END-Angabe angegeben, wird mit unbedingte anweisung-1 fortgesetzt oder mit der nächsten Anweisung, wenn AT END nicht angegeben ist. In beiden Fällen ist der Endwert des Index unbestimmt.
11. Kann bedingung erfüllt werden, verweist der Index auf das Tabellenelement, das bedingung erfüllt. Der Programmlauf wird mit unbedingte anweisung-2 oder mit der nächsten Anweisung fortgesetzt.
12. Eine zwei- oder dreidimensionale Tabelle kann durchsucht werden, wenn bezeichner-1 ein Datenfeld ist, das einem Datenfeld mit einer OCCURS-Klausel untergeordnet ist. In diesem Fall muss für jede Dimension der Tabelle ein Index durch die INDEXED BY-Angabe der OCCURS-Klausel angegeben sein. Durch die Ausführung der SEARCH ALL-Anweisung wird nur der Wert des zu bezeichner-1 gehörenden Index verändert. Um alle Einträge einer mehrdimensionalen Tabelle durchsuchen zu können, muss die SEARCH ALL-Anweisung für alle möglichen Werte der übergeordneten Indizes ausgeführt werden.
13. Endet in der AT END-Angabe und in den WHEN-Bedingungen keine der angegebenen Anweisungen mit einer GO TO-Anweisung, wird der Programmlauf nach Ausführung der unbedingten Anweisungen mit der nächsten Anweisung fortgesetzt.
14. Ist END-SEARCH angegeben, darf NEXT SENTENCE nicht angegeben sein.
15. Der Bereich einer SEARCH-Anweisung darf folgendermaßen begrenzt werden:
    a. mit END-SEARCH auf derselben Schachtelungsebene
    b. mit einem Punkt am Ende der SEARCH-Anweisung
    c. mit der ELSE- oder END-IF-Angabe einer vorhergehenden IF-Anweisung.

Beispiel 8-68 (für WHEN-Bedingungen)

Angaben in der DATA DIVISION



```
       ...
77  A-WERT PICTURE 9.
       ...
    02  TABELLE OCCURS 5 TIMES ASCENDING KEY IS A B C;
                INDEXED BY I.
        03  A  PICTURE 99.
        03  B  PICTURE 9.
            88 UNTER-30 VALUE 1.
            88 UEBER-30 VALUE 2.
        03  C  PICTURE 9.
       ...
```

Gültige WHEN-Bedingung in der PROCEDURE DIVISION

```
WHEN A(I) = 10
WHEN A(I) = 20 AND UNTER-30(I)
WHEN A(I) = 15 AND UEBER-30(I) AND C(I) = A-WERT
```

Beispiel 8-69

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUCHALL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T
    SYSIPT IS EINDAT.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  NR PIC 9(3).
77  EINGABE PIC 9(6).
01  ANGESTELLTEN-TAB.
    02  PERSONEN-TAB OCCURS 100 TIMES INDEXED BY PI,
                  ASCENDING KEY IS BEREICH KENN-NR.
        03  BEREICH PIC 9(3).
        03  KENN-NR PIC 9(6).
        03  NAME    PIC X(20).   .
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    PERFORM VARYING PI FROM 1 BY 1 UNTIL PI > 100
      ACCEPT PERSONEN-TAB (PI) FROM EINDAT
    END-PERFORM
    SEARCH ALL PERSONEN-TAB
    AT END
      DISPLAY "Person fehlt" UPON T
    WHEN BEREICH (PI) = NR AND KENN-NR (PI) = EINGABE
      DISPLAY BEREICH KENN-NR NAME UPON T
    END-SEARCH
    STOP RUN.
```

Hier besteht die Tabelle PERSONEN-TAB aus 100 Elementen.

Jedes Tabellenelement besteht aus einem drei Zeichen großen numerischen Feld BEREICH, einem sechs Zeichen langen numerischen Feld KENN-NR und einem 20 Zeichen langen alphanumerischen Feld NAME.

Die Tabelle ist in aufsteigender Reihenfolge nach BEREICH, bei gleichen Werten von BEREICH in aufsteigender Reihenfolge nach KENN-NR geordnet.

Der Anfang der Tabelle könnte folgenden Inhalt haben:

BEREICH	KENN-NR	NAME
101	123456	ADAM, D.
101	234561	LANGEWIESCHE, W.
101	123456	ADAM, D.
101	523618	EBERLE, F.
183	200305	DAUTZENBERG, K.
183	328512	REINHARDT, M.
183	433333	GRUEN, L.
183	987245	RICHTER, L.
557	328835	SCHMIDT, S.
557	775247	ALBRECHT, N.

Die SEARCH-Anweisung sucht in der Tabelle nach einem Element, dessen BEREICH mit dem aktuellen Inhalt des Feldes NR übereinstimmt, und dessen KENN-NR mit dem aktuel-len Inhalt des Feldes EINGABE übereinstimmt. Ist die Suche erfolgreich, wird die DISPLAY-Anweisung ausgeführt. Der Indexname PI zeigt dann auf das Element, das der Bedingung genügt.

Enthält z.B. NR 183 und EINGABE 328512, zeigt der Indexname PI auf das fünfte Tabellenelement.

Ist die Suche erfolglos, wird eine entsprechende Meldung ausgegeben.

8.10.40 SET-Anweisung

Funktion

Die SET-Anweisung bietet mit ihren verschiedenen Formaten folgende Möglichkeiten:

Format 1	ein ganzzahliges Datenfeld, einen Index oder ein Indexdatenfeld auf einen angegebenen Wert setzen
Format 2	den Wert eines Index vermindern oder erhöhen, so dass er eine neue Tabellenelementnummer darstellt
Format 3	den Zustand von externen Schaltern ändern
Format 4	den Wert von Bedingungsvariablen setzen
Format 5	Objektreferenzen zuweisen
Format 6	Datenzeiger zuweisen
Format 7	mit Datenzeigern rechnen
Format 8	Programmzeiger zuweisen
Format 9	den letzten Ausnahmezustand zurück setzen

Format 1

```
SET {index-1 | bezeichner-1}... TO {index-2 | bezeichner-2 | ganzzahl-1}
```

Syntaxregeln

Alles, was nachfolgend über index-1 und bezeichner-1 gesagt wird, trifft in gleicher Weise auf ihre Wiederholungen zu.

1. Jeder Index muss in einer OCCURS-Klausel in der INDEXED BY-Angabe angegeben sein.
2. Jeder Bezeichner muss entweder ein Indexdatenfeld oder ein numerisches Festpunktdatenelement, das als eine Ganzzahl beschrieben ist, bezeichnen.
3. Der Wert von ganzzahl-1 oder bezeichner-2 muss eine gültige Tabellenelementnummer der zugehörigen Tabelle sein. Der Compiler lässt jedoch auch andere ganzzahlige Werte zu (z.B. 0 oder negative Zahlen) - im Rahmen des erlaubten Wertebereichs für index-1 (siehe "Indizierung").
4. Indizes oder Bezeichner, die dem TO vorangehen, geben das Feld an, dessen Wert gesetzt werden soll.
5. index-2, bezeichner-2 und ganzzahl-1, die dem TO folgen, geben den Wert an, auf den das Empfangsfeld (z.B. index-1) gesetzt werden soll.
6. Bei Ausführung der SET-Anweisung geschieht Folgendes:
   a. index-1 wird so gesetzt, dass er auf jenes Tabellenelement Bezug nimmt, dessen Elementnummer dem Tabellenelement entspricht, auf das index-2, bezeichner-2 oder ganzzahl-1 Bezug nehmen.

      Bezeichnet bezeichner-2 ein Indexdatenfeld oder gehört index-2 zu derselben Tabelle wie index-1, findet keine Konvertierung statt.
   b. Ist bezeichner-1 ein Indexdatenfeld, wird es so gesetzt, dass es entweder den gleichen Inhalt wie index-2 oder wie bezeichner-2 hat, wobei bezeichner-2 ebenso ein Indexdatenfeld ist. Es findet keine Konvertierung statt. ganzzahl-1 darf in diesem Fall nicht verwendet werden.
   c. Ist bezeichner-1 kein Indexdatenfeld, wird er auf eine Tabellennummer gesetzt, die dem Wert von index-2 entspricht. In diesem Fall darf weder bezeichner-2 noch ganzzahl-1 verwendet werden.

Dieser Vorgang gilt auch für Wiederholungen von index-1 und bezeichner-1. Der Wert von index-2 oder bezeichner-2 wird jedesmal wie zu Beginn der Ausführung der Anweisung verwendet. Jede Subskribierung oder Indizierung für bezeichner-1 wird ausgewertet, bevor der Wert des betroffenen Datenfeldes verändert wird.

7. Die nachstehende Tabelle gibt an, welche Kombinationen von Operanden in der SET-Anweisung zulässig sind. Die Buchstaben nach den Schrägstrichen verweisen auf die unter Punkt 6. aufgeführten Regeln; z.B. bedeutet gültig/c, dass eine Kombination von Feldern entsprechend der Regel c) gültig ist.

Sendefeld	**Empfangsfeld**		
	ganzzahliges Datenfeld PIC9(n)	Indexname INDEXED BY	Indexname USAGE IS INDEX
ganzzahliges Literal	ungültig/c	gültig/a	ungültig/b
ganzzahliges Datenfeld	ungültig/c/	gültig/a	ungültig/b
Indexname Indexdatenfeld	gültig/c ungültig/c	gültig/a gültig/a*	gültig/b*gültig/b*

Tabelle 31: Gültige Verwendungen der SET-Anweisung

* = es findet keine Konvertierung statt.

8. Wird index-2 verwendet, muss sein Wert vor Ausführung der SET-Anweisung der Nummer eines Elements der zugehörigen Tabelle entsprechen.

Beispiel 8-70

Einträge im Datenteil:

```
     02 TABELLE-A PICTURE XXX OCCURS 50,
                 INDEXED BY IN-A1, IN-A2, IN-A3.
     02 TABELLE-B PICTURE XX OCCURS 55,
                 INDEXED BY IN-B.
 77 ID-1 USAGE IS INDEX.
 77 D-1 PICTURE IS S999, USAGE IS COMPUTATIONAL-3.
```

Anweisungen im Prozedurteil siehe die folgende tabellarische Aufstellung:

Anweisung	**Verwendete Operanden**	**Was getan wird [1)]**
`SET IN-A2 TO IN-A1`	Indexname wird auf Indexname gesetzt	Einfache Übertragung
`SET IN-A1 TO D-1`	Indexname wird auf numerisches Datenfeld gesetzt	Multipliziere (numerisches Datenfeld minus 1) mit der Tabellenelementlänge, um die Distanz zu erhalten.
		Einfache Übertragung

`SET IN-A1 TO ID-1`	Indexname wird auf Indexdatenfeld gesetzt	
`SET IN-A1 TO 4`	Indexname wird auf Literal gesetzt	Multipliziere (Literal minus 1)[2] mit der Tabellenelementlänge, um die Distanz zu erhalten.
`SET ID-1 TO IN-A1`	Indexdatenfeld wird auf Indexname gesetzt	Einfache Übertragung
`SET D-1 TO IN-A1`	Numerisches Datenfeld wird auf Indexname gesetzt	Dividiere Distanz durch Tabellenelement und addiere 1 dazu, um die Tabellenelementnummer zu erhalten.
`SET IN-B TO IN-A1`	Indexname wird auf Indexname gesetzt (verschiedene Tabellen)	Dividiere Distanz (z.B. Inhalt von IN-A1) durch die Tabellenelementlänge von TABELLE-A und addiere 1, um die Tabellenelementnummer zu erhalten. Multipliziere dann (Tabellenelementnummer minus 1) mit der Tabellenelementlänge der TABELLE-B, um die Distanz zu erhalten.

[1] Wegen des Zusammenhangs zwischen Tabellenelementnummer und Distanz siehe Abschnitt "Indizierung".
[2] Wird zum Übersetzungszeitpunkt ausgerechnet, beim Ablauf einfache Übertragung.

Beispiel 8-71

```
02  TABELLE-A OCCURS 50 TIMES
              INDEXED BY IND-1,IND-2,PIC 999.
 .
 .
 .
SET IND-1 TO 5.
SET IND-2 TO 7.
SET IND-1,TABELLE-A (IND-1) TO IND-2.
```

Die dritte SET-Anweisung ist gleichwertig mit folgenden zwei Anweisungen, die in der Reihenfolge ausgeführt werden, in der sie aufgeschrieben sind:

Anweisung	Wirkung
`SET IND-1 TO IND-2`	IND-1 wird auf Tabellenelementnummer 7, den aktuellen Wert von IND-2, gesetzt
`SET TABELLE-A (IND-1) TO IND-2`	Da die erste SET-Anweisung IND-1 auf 7 setzt, gilt TABELLE-A (IND-1) =TABELLE-A (7). Deshalb setzt diese Anweisung TABELLE-A (7) auf 7.

Format 2

```
SET {index-3}... {UP BY | DOWN BY} {bezeichner-3 | ganzzahl-2}
```

Syntaxregeln

1. Jeder Index muss in einer OCCURS-Klausel in der INDEXED BY-Angabe angegeben sein.
2. index-3... gibt den Speicherindex an, dessen Wert verändert werden soll.
3. bezeichner-3 muss ein numerisches Festpunktdatenelement sein, das als Ganzzahl beschrieben ist.
4. ganzzahl-2 muss eine ganze Zahl ungleich 0 sein und darf ein positives Vorzeichen enthalten.
5. UP BY oder DOWN BY gibt an, dass der Inhalt des angegebenen Index um einen bestimmten Wert erhöht (UP BY) oder vermindert (DOWN BY) werden soll. Dieser Wert entspricht der Wiederholungszahl, die den Wert von bezeichner-3 oder ganzzahl-2 darstellt.

Beispiel 8-72

```
02  TABELLE-A PICTURE X(20) OCCURS 5 INDEXED BY IND-1.
     .
     .
     .
SET IND-1 TO 4.
SET IND-1 UP BY 2.
SET IND-1 DOWN BY 3.
```

Die erste SET-Anweisung setzt den Index IND-1 auf Tabellenelementnummer 4, die zweite setzt ihn auf 6 (d.h., 4+2), und die dritte setzt ihn auf 3 (d.h. 6-3).

Format 3

```
SET { {merkname-1}... TO {ON | OFF} }...
```

Syntaxregel

1. Jedem Merknamen muss ein externer Schalter zugeordnet sein, dessen Zustand verändert werden kann (siehe "SPECIAL-NAMES-Paragraf").

Format 4

```
SET { {bedingungsname-1}... TO {TRUE | FALSE } }...
```

Syntaxregeln

1. bedingungsname-1 muss mit einer Bedingungsvariablen verknüpft sein (siehe "Bedingungsnamen-Bedingung").
2. Falls die FALSE-Angabe in der SET-Anweisung benutzt wird, muss die FALSE-Angabe in der VALUE-Klausel des Bedingungsnamens vorhanden sein.

Allgemeine Regeln

1. Das in der VALUE-Klausel angegebene, mit bedingungsname-1 verknüpfte Literal wird entsprechend den Regeln der VALUE-Klausel (siehe "VALUE-Klausel ") in die Bedingungsvariable eingesetzt.

   Wenn der Bedingungsvariable jedoch eine Gruppe untergeordnet ist, wird deren Länge entsprechend den Regeln der OCCURS-Klausel bestimmt (siehe Abschnitt "OCCURS-Klausel").

   Ist in der VALUE-Klausel mehr als ein Literal angegeben, wird die Bedingungsvariable auf den Wert des an erster Stelle stehenden Literals der VALUE-Klausel gesetzt.

2. Sind mehrere Bedingungsnamen angegeben, wird so verfahren, als ob für jeden einzelnen Bedingungsnamen eine eigene SET-Anweisung geschrieben worden wäre.
3. Bei "SET...TO FALSE" wird das Literal aus der FALSE-Angabe in der VALUE-Klausel, die mit `bedingungsname` verknüpft ist, entsprechend den Regeln der VALUE-Klausel in die Bedingungsvariable eingesetzt (siehe "SET bedingungsname TO TRUE").

Beispiel 8-73

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SETZEN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  BEISPIEL1.
    02  ARBEITSTAG  PICTURE X.
            88 MONTAG  VALUE "1".
            88 FREITAG VALUE "5".
01 BEISPIEL2.
        02  WOCHENTAG PIC X.
            88 ARBEITSTAGE VALUE "1" "2" "3" "4" "5".
PROCEDURE DIVISION.
P1 SECTION.
SETZEN.
    SET FREITAG TO TRUE.
    DISPLAY "Arbeitstag =" ARBEITSTAG UPON T.
    SET ARBEITSTAGE TO TRUE.
    DISPLAY "Wochentag =" WOCHENTAG UPON T.
SCHLUSS.
    STOP RUN.
```

Nach Ausführung der ersten SET-Anweisung steht im Datenfeld ARBEITSTAG (Bedingungsvariable) das Literal, das in der VALUE-Klausel dem Bedingungsnamen FREITAG zugeordnet ist: "5".

Nach Ausführung der zweiten SET-Anweisung steht im Datenfeld WOCHENTAG das Literal "1".

Format 5

```
SET {bezeichner-5}... TO bezeichner-6
```

Syntaxregeln

1. bezeichner-5 muss eine Objektreferenz sein, die als Empfangsfeld zulässig ist.
2. bezeichner-6 muss ein Klassenname oder eine Objektreferenz sein; die vordefinierte Objektreferenz SUPER darf nicht angegeben werden.
3. Ist das Daten-Element, auf welches bezeichner-5 verweist, eine universelle Objektreferenz, dann sind für bezeichner-6 alle Daten der Klasse objekt zugelassen.
4. Ist das Daten-Element, auf welches bezeichner-5 verweist, mit einem Interfacenamen bezeichnet, der die Schnittstelle int-1 (interface-1) angibt, dann darf das Daten-Element, auf welches bezeichner-6 verweist, nur eines der folgenden sein:
   a. eine Objektreferenz, die einen Interfacenamen für eine Schnittstelle angibt, welche mit int-1 konform ist;
   b. eine Objektreferenz, die einen Klassennamen gemäß den folgenden Regeln angibt:

- ist sie mit einer FACTORY-Angabe beschrieben , muss die Schnittstelle des Fabrikobjektes der spezfizierten Klasse zu int-1 konform sein;
- ist sie ohne FACTORY-Angabe beschrieben, muss die Schnittstelle der Objekte der spezifizierten Klasse zu int-1 konform sein;

c. eine Objektreferenz, mit ACTIVE-CLASS-Angabe, gemäß den folgenden Regeln:

- ist sie mit einer FACTORY-Angabe beschrieben, muss die Schnittstelle des Fabrikobjektes der Klasse, die das Daten-Element enthält, auf welches bezeichner-6 verweist, zu int-1 konform sein;
- ist sie ohne FACTORY-Angabe beschrieben, muss die Schnittstelle der Objekte der Klasse, die das Daten-Element enthält, auf welches bezeichner-6 verweist, mit int-1 konform sein;

d. eine Klasse, bei der die Schnittstelle ihres Fabrikobjektes zu int-1 konform ist;

e. die vordefinierte Objektreferenz SELF, wobei die Schnittstelle der Quelleinheit, welche die SET-Anweisung enthält, zu int-1 konform ist,

f. die vordefinierte Objektreferenz NULL.

5. Wenn das Daten-Element, auf welches bezeichner-5 verweist, mit einem Klassennamen bezeichnet wird, dann muss das Daten-Element, auf welches bezeichner-6 verweist, eines der folgenden sein:

a. eine Objektreferenz, die mit einem Klassennamen bezeichnet ist, gemäß folgender Regeln:

- ist das Daten-Element, auf welches bezeichner-5 verweist, mit der ONLY-Angabe beschrieben, dann muss das Daten-Element, auf welches bezeichner-6 verweist, auch mit der ONLY-Angabe beschrieben werden. Weiter muss der Klassenname, der in der Beschreibung des Daten-Elementes, auf das bezeichner-6 Bezug nimmt, definiert ist, identisch sein mit dem Klassennamen, der in der Beschreibung des Daten-Elementes angegeben ist, auf das bezeichner-5 verweist;
- ist das Daten-Element, auf welches bezeichner-5 verweist, ohne ONLY-Angabe beschrieben, dann muss der Klassenname, der in der Beschreibung des Daten Elementes, auf das bezeichner-6 Bezug nimmt, spezifiziert ist, auf dieselbe Klasse oder untergeordnete Klasse verweisen, die in der Beschreibung des Daten-Elementes angegeben ist, auf welches bezeichner-5 verweist;
- das Vorhandensein oder das Fehlen der FACTORY-Angabe muss der Angabe bei bezeichner-5 entsprechen.

b. eine Objektreferenz, die mit einer ACTIVE-CLASS-Angabe beschrieben wird, gemäß den folgenden Regeln:

- im Daten-Element, auf welches bezeichner-5 verweist, darf keine ONLY-Angabe definiert sein;
- die Klasse, die das Daten-Element enthält, auf welches bezeichner-6 verweist, muss dieselbe Klasse oder eine untergeordnete Klasse von der sein, die in der Beschreibung des Daten-Elementes spezifiziert ist, auf welches bezeichner-5 verweist;
- das Vorhandensein oder das Fehlen der FACTORY-Angabe muss der Angabe bei bezeichner-5 entsprechen.

c. die vordefinierte Objektreferenz SELF, gemäß den folgenden Regeln:

- das Daten-Element, auf welches bezeichner-5 verweist, darf keine ONLY-Angabe enthalten;
- die Klasse des Objektes, welches die SET-Anweisung enthält, muss dieselbe Klasse oder untergeordnete Klasse von der sein, die in der Beschreibung des Daten-Elementes spezifiziert ist, auf welches bezeichner-5 verweist;
- ist im Daten-Element, auf welches bezeichner-5 verweist, keine FACTORY-Angabe beschrieben, dann muss die Quelleinheit, welche die SET-Anweisung enthält, ein Instanzobjekt sein;
- enthält die Beschreibung des Daten-Elementes, auf das bezeichner-5 verweist, eine FACTORY-Angabe, dann muss die Quelleinheit, welche die SET-Anweisung enthält, ein Fabrikobjekt sein.

d. ein Klassenname, wenn Folgendes gilt:

- die Beschreibung des Daten-Elementes, auf das bezeichner-5 verweist, enthält eine FACTORY-Angabe,

- ist das Datenelement, auf welches bezeichner-5 verweist, mit der ONLY-Angabe beschrieben, dann muss das derselbe Klassenname sein. Anderenfalls darf es auch eine Unterklasse derjenigen Klasse sein, mit der das Datenelement, auf das bezeichner-5 verweist, beschrieben ist.

e. die vordefinierte Objektreferenz NULL.

6. Enthält die Beschreibung des Daten-Elementes, auf das bezeichner-5 verweist, eine ACTIVE-CLASS-Angabe, dann muss das Daten-Element, auf welches bezeichner-6 Bezug nimmt, eines der folgenden sein:
   a. eine Objektreferenz, die mit einer ACTIVE-CLASS-Angabe beschrieben ist, wobei das Vorhandensein oder das Fehlen der FACTORY-Angabe analog sein muss, wie im Daten-Element beschrieben, auf das bezeichner-5 verweist;
   b. die vordefinierte Objektreferenz SELF, gemäß den folgenden Regeln:
      - ist im Daten-Element, auf welches bezeichner-5 verweist, keine FACTORY-Angabe beschrieben, dann muss das Objekt, welches die SET-Anweisung enthält, ein Instanzobjekt sein;
      - enthält die Beschreibung des Daten-Elementes, auf das bezeichner-5 verweist, eine FACTORY-Angabe, dann muss das Objekt, welches die SET-Anweisung enthält, ein Fabrikobjekt sein.
   c. die vordefinierte Objektreferenz NULL.

Allgemeine Regeln

1. Ist bezeichner-6 eine Objektreferenz, dann wird diese Referenz in das Datenelementbezeichner-5 übertragen und in jede seiner Wiederholungen in der angegebenen Reihenfolge.
2. Ist bezeichner-6 ein Klassenname, dann wird eine Referenz auf das Fabrikobjekt der Klasse in das Daten-Element bezeichner-5 übertragen und in jede seiner Wiederholungen in der angegebenen Reihenfolge.

Konformität der Objektreferenzen bei SET-Anweisungen

Die folgende Tabelle zeigt zusammenfassend die Kombinationen der Objektreferenzen im Sende- und Empfangsfeld.

Kombination

immer erlaubt:	o.k.
bedingt erlaubt:	Bedingung
verboten:	----

Notation für Bedingungen

A > B	Interface A ist konform zu Interface B:
A >> B	Klasse A ist Unterklasse von B:
A == B	Klasse A ist die gleiche Klasse wie B:

Abkürzungen

i5	Interface des Empfangsfeldes
i6	Interface des Sendefeldes
c5	Klasse des Empfangfeldes
c6	Klasse des Sendefeldes

c	Klasse entsprechend dem Namen
C	Klasse, in der die SET-Anweisung steht[1]
FI(X)	Interface des Factory-Objekts der Klasse X
OI(X)	Interface eines Objekt-Objekts der Klasse X

[1] Darunter ist die Klasse zu verstehen, in der die Methode definiert ist, die die SET-Anweisung enthält und **nicht** eine Klasse, die diese Methode erbt.

		Sendefeld Objektreferenz USAGE							
Empfangsfeld Objektreferenz USAGE		(universell)	interface i6	FACTORY klassen-name c6	FACTORY klassen-name ONLY c6	klassen-name c6	klassen-name ONLY c6	FACTORY ACTIVE-CLASS C	ACTIVE-CLASS C
(universell)		o.k.	o.k.	o.k.	o.k.	o.k.	o.k.	o.k.	o.k.
interface	i5	---	i6 > i5	FI(c6)>i5	FI(c6)>i5	OI(c6)>i5	OI(c6)>i5	FI(C)>i5	OI(C)>i5
FACTORY klassenname	c5	---	---	c6>>==c5	c6>>==c5	---	---	C>>==c5	---
FACTORY klassenname ONLY	c5	---	---	---	c6==c5	---	---	---	---
klassenname	c5	---	---	---	---	c6>>==c5	c6>>==c5	---	C>>==c5
klassenname ONLY	c5	---	---	---	---	---	c6==c5	---	---
FACTORY ACTIVE-CLASS	C	---	---	---	---	---	---	o.k.	---
-ACTIVE-CLASS	C	---	---	---	---	---	---	---	o.k.

Tabelle 32: Kombination der Objektreferenzen im Sende- und Empfangsfeld

		Sendefelder			
		klassenname	**Objektreferenz SELF**		**NULL**
Empfangsfeld Objektreferenz USAGE		c	in Factory-Methode C	in Objekt- Methode C	
(universell)		o.k.	o.k.	o.k.	o.k.
interface	i5	FI(c)>i5	FI(C)>i5	OI(C)>i5	o.k.
FACTORY klassenname	c5	c>>==c5	C>>==c5	---	o.k.
FACTORY klassenname ONLY	c5	c==c5	---	---	o.k.
klassenname	c5	---	---	C>>==c5	o.k.
klassenname ONLY	c5	---	---	---	o.k.
	C	---	o.k.	---	o.k.

FACTORY ACTIVE-CLASS					
ACTIVE-CLASS	C	---	---	o.k.	o.k.

Tabelle 33: Kombination der Objektreferenzen im Sende- und Empfangsfeld

Format 6

```
SET {ADDRESS OF datenname-1 | bezeichner-7 ...} TO bezeichner-8
```

Syntaxregeln

1. bezeichner-7 und bezeichner-8 müssen ein Datenelement der Kategorie datenzeiger referenzieren.
2. datenname-1 muss ein Datenelement sein, das mit der BASED-Klausel spezifiziert ist. Siehe aber auch Compiler-Option PERMIT-STANDARD-DEVIATION.
3. datenname-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.
4. Ist bezeichner-7 ein typbezogener Zeiger, muss bezeichner-8 entweder die vordefinierte Adresse NULL sein oder ein typbezogener Zeiger auf den gleichen Typ. Ist der datenname-1 stark typisiert, muss bezeichner-8 ein typbezogener Zeiger auf den gleichen Typ sein.
   Ist bezeichner-8 ein typbezogener Zeiger, muss entweder bezeichner-7 ein typbezogener Zeiger auf den gleichen Typ sein oder datenname-1 muss vom gleichen Typ typisiert sein, an den bezeichner-8 gebunden ist.

Allgemeine Regeln

1. Ist bezeichner-7 angegeben, dann wird die durch bezeichner-8 spezifizierte Adresse in dem durch bezeichner-7 referenzierten Datenelement abgelegt.
2. Ist datenname-1 angegeben, dann wird die durch bezeichner-8 spezifizierte Adresse dem durch datenname-1 referenzierten Datenelement zugeordnet.

Format 7

```
SET bezeichner-9 ... {UP | DOWN} BY {bezeichner-10 | ganzzahl-3}
```

Syntaxregeln

1. bezeichner-10 muss ein numerisches Festpunktdatenfeld sein, das als Ganzzahl beschrieben ist.
2. ganzzahl-3 muss eine ganze Zahl oder 0 sein, die ein Vorzeichen enthalten darf.
3. bezeichner-9 muss von der Kategorie Datenzeiger sein.
4. bezeichner-9 darf kein typbezogener Zeiger sein, bei der die zugehörige Typdeklaration die Angabe STRONG hat.

Allgemeine Regel

1. Die Adresse, die im Zeigerdatenfeld enthalten ist, wird erhöht, wenn UP angegeben istbzw. erniedrigt, wenn DOWN angegeben ist und zwar um die Anzahl Bytes, die durch bezeichner-10 oder ganzzahl-3 angegeben ist.

Format 8

```
SET {bezeichner-11}... TO bezeichner-12
```

Syntaxregeln

1. bezeichner-11 muss ein Datenfeld der Kategorie Programmzeiger sein. bezeichner-12 muss von der Kategorie Programmzeiger sein.

Allgemeine Regel

1. Die mit bezeichner-12 bezeichnete Adresse wird in jedes Datenfeld bezeichner-11 in der angegebenen Reihenfolge gespeichert.

Format 9

```
SET LAST EXCEPTION TO OFF
```

Allgemeine Regel

1. Der letzte Ausnahmezustand wird zurückgesetzt, d.h. er zeigt an, dass kein Ausnahmezustand ausgelöst wurde.

8.10.41 SORT-Anweisung

Funktion

Format 1	Sortieren von Datensätzen. Mit der SORT-Anweisung können Datensätze, die entweder in einer Eingabe prozedur erzeugt werden oder in einer Datei enthalten sind, nach einer Anzahl von angegebenen Datenfeldern (Sortierschlüssel) sortiert werden. Die sortierten Sätze werden einer Ausgabeprozedur übergeben oder in eine Datei eingetragen.
Format 2	Sortieren von Tabellen. Die SORT-Anweisung bewirkt, dass Tabellenelemente entsprechend einer vom Benutzer festgelegten Vergleichsfolge geordnet werden.

Format 1 Sortieren von Datensätzen

```
SORT sortierdateiname

    {ON {DESCENDING | ASCENDING} {KEY | KEY-YY } {datenname-1}... }...

    [WITH DUPLICATES IN ORDER]

    [COLLATING SEQUENCE IS alphabetname]

    {INPUT PROCEDURE IS paragrafenname-1 [{THRU | THROUGH} paragrafenname-2] |
    USING {dateiname-1}...}

    {OUTPUT PROCEDURE IS paragrafenname-3 [{THRU | THROUGH} paragrafenname-4] |
    GIVING {dateiname-2}...}
```

Syntaxregeln

1. Die SORT-Anweisung darf überall in der PROCEDURE DIVISION angegeben werden, außer
   - in DECLARATIVES und
   - in Ein- und Ausgabeprozeduren, die zu einer SORT-Anweisung gehören.
2. sortierdateiname muss in einer Sortierdateierklärung (SD) in der DATA DIVISION beschrieben sein.
3. sortierdateiname muss mit dem Sortierdateinamen übereinstimmen, der in der SELECT-Klausel (Format 2) angegeben ist.
4. datenname-1... sind Sortierschlüssel. Ein Sortierschlüssel ist ein Teil des Datensatzes, der als Sortiergrundlage verwendet wird. Die Sortierschlüssel müssen in einer Datensatzbeschreibung, die zu einer SD-Erklärung gehört, definiert sein und unterliegen folgenden Regeln:
   a. Die Länge der Datenfelder darf nicht variabel sein.
   b. Datennamen, die die Sortierschlüssel beschreiben, dürfen gekennzeichnet sein (siehe „Kennzeichnung").
   c. Enthält die Sortierdateierklärung von sortierdateiname mehrere Datensatzbeschreibungen, müssen die Schlüssel nur in einer Datensatzbeschreibung definiert sein. Die durch den Schlüssel in einer Datensatzbeschreibung angesprochenenen Bytepositionen gelten auch als Schlüssel in allen anderen Datensatzbeschreibungen der Datei.
   d. Ein Sortierschlüssel darf nicht mit einer OCCURS-Klausel beschrieben und nicht einem Datenfeld untergeordnet sein, das mit einer OCCURS-Klausel beschrieben ist.
   e. Enthält die Sortierdatei (sortierdateiname) Sätze mit variabler Länge, müssen alle Sortierschlüssel in den ersten n Zeichenpositionen des Satzes enthalten sein, wobei n die minimale Satzlänge ist, die für die Sortierdatei angegeben wurde.

f. Für eine Datei können maximal 64 Schlüssel angegeben werden.

g. Jeder Sortierschlüssel muss innerhalb der ersten 4096 Bytes des Satzes beginnen.

h. Sortierschlüssel werden in der Reihenfolge ihrer Wichtigkeit für die Sortierung von links nach rechts angegeben, unabhängig davon, ob sie auf- oder absteigend sind. Die erste Angabe von datenname-1 wäre somit der Hauptsortierbegriff, die zweite Angabe von datenname-1 der nachgeschaltete Sortierbegriff.

i. Die maximale von SORT verarbeitbare Länge eines Sortierschlüssels hängt von dessen Format ab. Sie liegt für das Format PD (packed decimal) bei 16 Bytes, für Zeichenstring maximal bei Satzlänge, für alle übrigen Formate bei 256 Bytes.

j. Die Sortierschlüssel, die der Angabe KEY-YY folgen, müssen entweder mit PIC 99 USAGE DISPLAY oder USAGE PACKED-DECIMAL definiert sein.

5. dateiname-1..., dateiname-2... müssen in einer Dateierklärung (FD) der DATA DIVISION beschrieben sein.
6. Für die Satzlängen der Datensätze aus den Eingabedateien, die an die SORT-Operation übergeben werden, gilt: Hat die Sortierdatei festes Satzformat, wird der Programmlauf beendet, falls der Satz zu lang ist. Ist er zu kurz, werden die fehlenden Stellen mit Blanks aufgefüllt. Hat die Sortierdatei variables Satzformat, werden die Eingabesätze in der eingegebenen Länge übernommen. Diese Länge muss in dem Bereich liegen, der in der RECORD-Klausel für die Sortierdatei festgelegt ist (siehe „RECORD-Klausel").
7. Mindestens eine ASCENDING-/DESCENDING-Angabe muss in einer SORT-Anweisung angegeben sein.
8. Die in der SORT-Anweisung genannten Sortier- und Eingabedateien dürfen nicht zusammen in derselben SAME SORT AREA oder SAME SORT-MERGE AREA angegeben sein.
9. Falls dateiname-2 eine indizierte Datei beschreibt, muss die erste Angabe von datenname-1 mit der ASCENDING-Angabe erfolgen. Das durch datenname-1 referenzierte Datenfeld muss in seinem Datensatz dieselben Zeichenstellen belegen wie der Satzschlüssel innerhalb der durch dateiname-2 beschriebenen Datei.

Allgemeine Regeln

1. Die Sortierreihenfolge wird durch die ASCENDING-/DESCENDING-Angabe festgelegt:
   a. Bei Angabe von ASCENDING wird vom niedrigsten zum höchsten Wert des Sortierschlüssels sortiert (aufsteigende Sortierreihenfolge).
   b. Bei Angabe von DESCENDING wird vom höchsten zum niedrigsten Wert des Sortierschlüssels sortiert (absteigende Sortierreihenfolge).

   Für die Sortierreihenfolge gelten die Regeln für den Vergleich von Operanden in „Vergleichsbedingungen“.
2. Ist DUPLICATES angegeben und haben mehrere Datensätze den gleichen Inhalt in allen Sortierfeldern, gilt für die Rückgabe aus der Sortierdatei nachstehende Reihenfolge:
   a. Ist keine Eingabeprozedur definiert, werden die Sätze in der Reihenfolge zurückgegeben, wie die zugehörigen Dateien in der SORT-Anweisung angegeben sind. Gibt es Datensätze mit gleichem Sortierfeldinhalt in ein und derselben Datei, werden die Sätze in der Reihenfolge, in der sie eingelesen wurden, zurückgegeben.
   b. Ist eine Eingabeprozedur definiert, werden die Sätze in der Reihenfolge, in der sie die Eingabeprozedur verlassen haben, zurückgegeben.
3. Ist DUPLICATES nicht angegeben und haben mehrere Datensätze den gleichen Inhalt in allen Sortierfeldern, ist die Reihenfolge der Rückgabe dieser Sätze undefiniert.
4. Beim Programmablauf ist die Sortierfolge für die Vergleiche von alphanumerischen Sortierfeldern wie folgt festgelegt:
   a. Wenn COLLATING SEQUENCE in der SORT-Anweisung angegeben ist, dann ist diese Angabe Kriterium für die Sortierfolge,
   b. Wenn COLLATING SEQUENCE in der SORT-Anweisung nicht angegeben ist, wird die programmspezifische Sortierfolge verwendet (siehe "OBJECT-COMPUTER-Paragraf").

Für Vergleiche von nationalen Sortierfeldern wird die nationale (maschineneigene) Sortierfolge verwendet.

5. INPUT PROCEDURE bedeutet, dass der Prozedurteil eine Eingabeprozedur enthält, in der Datensätze vor dem Sortieren bearbeitet werden. Ist INPUT PROCEDURE angegeben, wird die Steuerung dann an die Eingabeprozedur übergeben, wenn der Eingabeteil des Programms SORT bereit ist, den ersten Datensatz zu übernehmen. Beim Durchlaufen einer RELEASE-Anweisung übergibt die Eingabeprozedur die Datensätze an die Sortierdatei (siehe „RELEASE-Anweisung"). Der Compiler fügt einen Rückkehrmechanismus am Ende des letzten Kapitels der Eingabeprozedur ein, d.h., ist die letzte Anweisung in der Eingabeprozedur durchlaufen, wird die Eingabeprozedur abgeschlossen und die übergebenen Datensätze werden in der Sortierdatei sortiert. Für die Eingabeprozedur, die ein abgeschlossener Teil innerhalb der PROCEDURE DIVISION ist, gelten folgende Regeln:
   a. Sie muss aus einem oder mehreren zusammenhängenden Kapiteln bestehen.
   b. Sie muss mindestens eine RELEASE-Anweisung enthalten, damit Datensätze an die Sortierdatei übergeben werden können (siehe „RELEASE-Anweisung").
   c. Sie darf nicht zur Ausführung einer MERGE-, RETURN- oder SORT-Anweisung führen.
   d. Sie kann jede Prozedur enthalten, die erforderlich ist, um Datensätze auszuwählen, zu erzeugen oder zu verändern.
   e. Die Eingabeprozedur darf verlassen werden, falls der Benutzer dafür sorgt, dass einem Sprung aus der Eingabeprozedur ein Rücksprung folgt, um ein ordnungsgemäßes Verlassen dieser Prozedur zu gewährleisten (Durchlaufen der letzten
   Anweisung der Eingabeprozedur).
   f. Von außerhalb darf auf Prozedurnamen in der Eingabeprozedur gesprungen werden, falls dabei keine RELEASE-Anweisung oder das Ende der Eingabeprozedur durchlaufen wird.
6. Ist eine Eingabeprozedur angegeben, wird sie durchlaufen, bevor die Datensätze in der Sortierdatei sortiert werden.
7. Ist USING angegeben, werden alle Datensätze der Eingabedateien (dateiname-1...) in die mit sortierdateiname bezeichnete Sortierdatei übertragen. Bei Beginn der Ausfüh-rung der SORT-Anweisung dürfen die Eingabedateien nicht geöffnet sein. Die SORT-Anweisung wird für jede genannte Datei in folgenden Schritten ausgeführt:
   a. Die Verarbeitung der Datei wird eingeleitet. Dies geschieht so, als ob eine OPEN INPUT-Anweisung ausgeführt würde.
   b. Jeder logische Datensatz wird für den Sortiervorgang zur Verfügung gestellt und anschließend freigegeben, als ob eine READ-Anweisung mit NEXT RECORD- und AT END-Angabe ausgeführt würde. Der Inhalt der Satzschlüsselfelder einer relativen Datei ist nach der Ausführung der SORT-Anweisung unbestimmt, falls dateiname-1 nicht auch in der GIVING-Angabe angegeben ist.
   Eine relative Datei muss im FILE-CONTROL-Paragrafen mit ACCESS MODE IS SEQUENTIAL beschrieben sein.
   c. Die Verarbeitung der Datei wird beendet. Dies geschieht so, als ob eine CLOSE-Anweisung ohne Wahlangaben ausgeführt würde. Diese Beendigung ist vollzogen, bevor der Sortiervorgang beginnt.

   Diese impliziten Funktionen werden so durchgeführt, dass alle vereinbarten USE AFTER EXCEPTION /ERROR-Prozeduren vollzogen sind. Gleichwohl darf die Ausführung einer solchen USE-Prozedur nicht die Ausführung irgendeiner Anweisung bewirken, die die Eingabedateien oder deren Satzbereichsvereinbarungen verändert.
8. OUTPUT PROCEDURE bedeutet, dass die PROCEDURE DIVISION eine Ausgabeprozedur enthält, in der Datensätze nach dem Sortieren bearbeitet werden. Ist OUTPUT PROCEDURE angegeben, wird die Steuerung dann an die Ausgabeprozedur übergeben, nachdem die Sortierdatei durch die SORT-Anweisung bearbeitet worden ist. Beim Durchlaufen einer RETURN-Anweisung übernimmt die Ausgabeprozedur die Datensätze aus der Sortierdatei. Der Compiler fügt einen Rückkehrmechanismus am Ende des letzten

Kapitels der Ausgabeprozedur ein, d.h. ist die letzte Anweisung der Ausgabeprozedur durchlaufen, wird die Ausgabeprozedur abgeschlossen und die Steuerung geht an die der SORT-Anweisung folgende Anweisung über.

Für die Ausgabeprozedur, die ein abgeschlossener Teil innerhalb der PROCEDURE DIVISION ist, gelten folgende Regeln:

a. Sie muss aus einem oder mehreren zusammenhängenden Kapiteln bestehen, die nicht Teil einer Eingabeprozedur sind.
b. Sie muss mindestens eine RETURN-Anweisung enthalten, damit die sortierten Datensätze für die Verarbeitung verfügbar sind (siehe „RETURN-Anweisung").
c. Sie darf nicht zur Ausführung einer MERGE-, RELEASE- oder SORT-Anweisung führen.
d. Sie kann jede Prozedur enthalten, die erforderlich ist, um Datensätze auszuwählen, zu erzeugen oder zu verändern, bevor sie weitergegeben werden.
e. Die Ausgabeprozedur darf verlassen werden, falls der Benutzer dafür sorgt, dass einem Sprung aus der Ausgabeprozedur ein Rücksprung folgt, um ein ordnungsgemäßes Verlassen dieser Prozedur zu gewährleisten (Durchlaufen der letzten Anweisung der Ausgabeprozedur).
f. Von außerhalb darf auf Prozedurnamen in der Ausgabeprozedur gesprungen werden, falls dabei keine RETURN-Anweisung oder das Ende der Ausgabeprozedur durchlaufen wird.

9. Wird die Angabe INPUT PROCEDURE oder OUTPUT PROCEDURE verwendet, wird eine Programmverzweigung entsprechend einer PERFORM-Anweisung im Format 1 ausgeführt. Das bedeutet, dass alle Kapitel, die die Prozedur bilden, einmal durchlaufen werden und die Ausführung der Prozedur nach Verarbeitung der letzten Anweisung beendet wird. Deshalb darf jede Prozedur (oder beide Prozeduren) durch eine EXIT-Anweisung abgeschlossen werden.
10. Ist GIVING dateiname-2... angegeben, werden alle sortierten Sätze in die Ausgabedatei (dateiname-2...) geschrieben.
    Bei Beginn der Ausführung der SORT-Anweisung darf die Ausgabedatei nicht geöffnet sein. Die SORT-Anweisung wird für jede genannte Datei in folgenden Schritten ausgeführt:
    a. Die Verarbeitung der Datei wird eingeleitet. Dies geschieht so, als ob eine OPEN OUTPUT-Anweisung ausgeführt würde. Der Startschritt ist nach der Ausführung einer Eingabeprozedur vollzogen.
    b. Die sortierten logischen Datensätze sind verarbeitet und werden in die Ausgabedatei geschrieben, als ob eine WRITE-Anweisung ohne jede Wahlangabe ausgeführt würde. Die Länge dieser Datensätze muss innerhalb der Grenzen liegen, die für die Ausgabedatei festgelegt sind (siehe „RECORD-Klausel").
       Bei einer relativen Datei enthält das Satzschlüsselfeld des ersten verarbeiteten Satzes den Wert „1“, das des zweiten verarbeiteten Satzes den Wert „2“ usw. Nach der Ausführung der SORT-Anweisung zeigt der Inhalt des Satzschlüsselfeldes auf den letzten verarbeiteten Satz der Datei. Die Datei muss im FILE-CONTROL-Paragrafen mit ACCESS MODE IS SEQUENTIAL beschrieben sein.
    c. Die Verarbeitung der Datei wird beendet. Dies geschieht so, als ob eine CLOSE-Anweisung ohne Wahlangaben ausgeführt würde.

    Diese impliziten Funktionen werden so durchgeführt, dass alle vereinbarten USE AFTER EXCEPTION /ERROR-Prozeduren vollzogen sind. Gleichwohl darf die Ausführung einer solchen USE-Prozedur nicht die Ausführung irgendeiner Anweisung bewirken, die die Eingabedateien oder deren Satzbereichsvereinbarungen verändert. Beim ersten Versuch, einen Satz über die für die Datei extern vereinbarten Grenzen hinaus zu schreiben, wird jede für die Datei vereinbarte USE AFTER STANDARD EXCEPTION/ERROR-Prozedur ausgeführt; wenn die Steuerung von dieser USE-Prozedur zurückgekehrt ist oder wenn keine derartige Prozedur vereinbart wurde, wird die Verarbeitung der Datei beendet, wie unter c) beschrieben.
11. Da die SORT-Anweisung nicht satzorientiert arbeitet, entspricht sie nicht den normalen Ein-/Ausgabe-Anweisungen (READ, WRITE usw.). Die READ-Anweisung liest bei ihrer Ausführung nur einen einzigen Datensatz, die WRITE-Anweisung schreibt einen einzelnen Datensatz. Die SORT-Anweisung dagegen

behandelt nicht einen einzelnen Datensatz, sondern eine ganze Datei. Daher muss diese ganze Datei SORT zur Verfügung gestellt werden (entweder durch die USING-Angabe oder durch wiederholte Verwendung einer RELEASE-Anweisung in einer Eingabeprozedur), ehe SORT arbeiten kann. Das Programm SORT verändert die Reihenfolge der Datensätze innerhalb der Datei und daher ist in der Regel der erste Datensatz, der vom Sortierprogramm zurückgegeben wird, nicht der erste Datensatz, der an das Sortierprogramm übergeben wurde. SORT kann keinerlei Ausgabe liefern, ehe es nicht die gesamte Eingabe erhalten hat.

Beispiele für Dateien-SORT

Beispiel 8-74

Sortieren mit einer Ausgabedatei

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORTIER1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EINGABE ASSIGN TO "EINGABE".
    SELECT AUSGABE ASSIGN TO "AUSGABE".
    SELECT SORTIER ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD  EINGABE LABEL RECORD STANDARD.
01  ESATZ.
    02  E0            PIC X.
    02  E1            PIC 9(4).
    02  E2            PIC 9(4).
    02  E3            PIC 9(4).
FD  AUSGABE LABEL RECORD STANDARD.
01  ASATZ
    02  A0            PIC X.
    02  A1            PIC 9(4).
    02  A2            PIC 9(4).
    02  A3            PIC 9(4).
SD  SORTIER LABEL RECORD STANDARD.
01  SSATZ.
    02  S0            PIC X.
    02  S1            PIC 9(4).
    02  S2            PIC 9(4).
    02  S3            PIC 9(4).
PROCEDURE DIVISION.
P1 SECTION.
SORTIEREN.
    SORT SORTIER ASCENDING S1 S2 S3          (1)
                                              |
         USING EINGABE GIVING AUSGABE.       (1)
    STOP RUN.
```

(1) Das Sortieren läuft in folgenden Schritten ab:

1. Übergabe der Sätze aus der Eingabedatei EINGABE an die Sortierdatei SORTIER.
2. Sortieren der Sätze in der Sortierdatei nach aufsteigendem S1. Bei Gleichheit von S1 in mehreren Sätzen entsprechend nach aufsteigendem S2 und bei Gleichheit von S1 und S2 nach aufsteigendem S3.

3. Übergabe der Sätze aus der Sortierdatei an die Ausgabedatei AUSGABE.

Beispiel 8-75

Sortieren mit zwei Ausgabedateien

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORTIER2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EINGABE ASSIGN TO "EINGABE".
    SELECT AUSGABE1 ASSIGN TO "AUSGABE1".
    SELECT AUSGABE2 ASSIGN TO "AUSGABE2".
    SELECT SORTIER ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD  EINGABE LABEL RECORD STANDARD.
01  ESATZ.
    02  E0          PIC X.
    02  E1          PIC 9(4).
    02  E2          PIC 9(4).
    02  E3          PIC 9(4).
FD  AUSGABE1 LABEL RECORD STANDARD.
01  A1SATZ PIC X(13).
FD  AUSGABE2 LABEL RECORD STANDARD.
01  A2SATZ PIC X(13).
SD  SORTIER LABEL RECORD STANDARD.
01  SSATZ.
    02  S0          PIC X.
    02  S1          PIC 9(4).
    02  S2          PIC 9(4).
    02  S3          PIC 9(4).
WORKING-STORAGE SECTION.
01  EINGABE-STATUS PIC X VALUE LOW-VALUE.
    88  EINGABE-END VALUE HIGH-VALUE.
01  SORTIER-STATUS PIC X VALUE LOW-VALUE.
    88  SORTIER-ENDE VALUE HIGH-VALUE.
PROCEDURE DIVISION.
HAUPT SECTION.
H1.
    OPEN INPUT EINGABE OUTPUT AUSGABE1 AUSGABE2.
    SORT SORTIER ASCENDING S1 S2 S3                          (1)
                                                              |
         INPUT PROCEDURE EINGANG OUTPUT PROCEDURE AUSGANG.   (1)
    CLOSE EINGABE AUSGABE1 AUSGABE2.
HE.
    STOP RUN.
EINGANG SECTION.
E0.
    PERFORM UNTIL EINGABE-ENDE                               (2)
      READ EINGABE                                            |
      AT END                                                  |
        SET EINGABE-ENDE TO TRUE                              |
      NOT AT END                                              |
```

```
          IF E0 NOT = "C"                                    |
            THEN                                             |
            RELEASE SSATZ FROM ESATZ                         |
          END-IF                                             |
        END-READ                                             |
    END-PERFORM.                                            (2)
AUSGANG SECTION.
A0.
    PERFORM UNTIL SORTIER-ENDE                              (3)
      RETURN SORTIER                                         |
      AT END                                                 |
        SET SORTIER-ENDE TO TRUE                             |
      NOT AT END                                             |
        IF S0 = "A"                                          |
        THEN                                                 |
          WRITE AUSGABE1 FROM SSATZ                          |
        ELSE                                                 |
          WRITE AUSGABE2 FROM SSATZ                          |
        END-IF                                               |
      END-RETURN                                             |
    END-PERFORM.                                            (3)
```

(1)

1. Die Steuerung wird an die Eingabeprozedur abgegeben (EINGANG SECTION).
2. Die Sätze in der Sortierdatei werden aufsteigend nach S1 S2 S3 sortiert.
3. Die Steuerung geht an die Ausgabeprozedur über (AUSGANG SECTION).

(2) Ein SATZ der Eingabedatei wird eingelesen. Nur solche Sätze, die an der ersten Stelle kein „C“ enthalten, sollen verarbeitet werden. Ist der Satz ESATZ gültig, wird er an die Sortierdatei (als SSATZ) übergeben. Dies wird so lange durchgeführt, bis das Ende der Eingabedatei erkannt wird. Dann geht die Steuerung wieder zur SORT-Anweisung.

(3) Ein Satz wird aus der Sortierdatei übernommen. Enthält er als erstes Zeichen ein „A“, wird er in die 1. Ausgabedatei (AUSGABE1) geschrieben. Andere Sätze werden in die 2. Ausgabedatei (AUSGABE2) geschrieben. Wenn alle Sätze der Sortierdatei verarbeitet sind, wird die der SORT-Anweisung folgende Anweisung ausgeführt.

Format 2 Sortieren von Tabellen

```
SORT datenname-2 {ON {ASCENDING | DESCENDING} {KEY | KEY-YY} {datenname-1}... }...
    [WITH DUPLICATES IN ORDER]
    [COLLATING SEQUENCE IS alphabetname]
    [USING datenname-3]
```

Syntaxregeln

1. Die SORT-Anweisung darf überall in der PROCEDURE DIVISION angegeben werden, außer in DECLARATIVES und in Ein- und Ausgabeprozeduren, die zu einer SORT-Anweisung gehören.
2. datenname-2 bzw. datenname-3, falls USING angegeben ist, bezeichnen die zu sortierende Tabelle. datenname-1... bezeichnet ein oder mehrere Datenfelder, die als Sortierbegriffe verwendet werden sollen (Schlüsselfelder),

3. Der mit datenname-2 bezeichnete Datensatz muss in einer Datenerklärung definiert sein und unterliegt dabei folgenden Regeln:
   a. datenname-2 darf gekennzeichnet sein.
   b. Die Datenerklärung von datenname-2 muss eine OCCURS-Klausel enthalten, d.h. als Tabellenelement definiert sein.
   c. Wenn die mit datenname-2 bezeichnete Tabelle einer Tabelle untergeordnet ist (mehrdimensionale Tabelle), dann muss datenname-2 als indizierbare Tabelle definiert sein, d.h. in der Datenerklärung der übergeordneten Tabelle mit der INDEXED-BY-Angabe ein Indexname vereinbart sein. Der Indexname muss vor der Ausführung der SORT-Anweisung mit der gewünschten Elementnummer versorgt werden (siehe „Indizierung").
4. Die mit datenname-1 bezeichneten Schlüsselfelder müssen in der Datenerklärung von datenname-2 definiert sein. Dabei gelten folgenden Regeln:
   a. datenname-1 ist entweder namensgleich mit datenname-2 oder namensgleich mit einem datenname-2 untergeordneten Datenfeld.
   b. datenname-1 darf gekennzeichnet sein.
   c. Bezieht sich datenname-1 auf ein datenname-2 untergeordnetes Datenfeld, so darf die Beschreibung dieses Datenfeldes weder selbst eine OCCURS-Klausel enthalten, noch einem Datenfeld untergeordnet sein, dessen Beschreibung eine
   d. datenname-1 darf sich nicht auf ein Datenfeld beziehen, dessen Beschreibung eine SIGN-Klausel enthält.
   e. Ist das durch datenname-1 bezeichnete Datenfeld als numerisch mit Vorzeichen definiert, darf es nicht länger als 16 Zeichen sein.
   f. Die Sortierschlüssel, die der Angabe KEY-YY folgen, müssen entweder mit PIC 99 USAGE DISPLAY oder USAGE PACKED-DECIMAL definiert sein.
5. Für datenname-3 gelten dieselben Regeln wie für datenname-2.
6. Wird die USING-Angabe verwendet, müssen datenname-2 und datenname-3 entweder beide als nationale oder beide als alphanumerische bzw. alphabetische Datenfelder beschrieben sein.

Allgemeine Regeln

1. Die Schlüsselwörter ASCENDING und DESCENDING wirken über alle nachfolgenden datenname-1-Angaben hinweg bis zum nächsten Schlüsselwort ASCENDING oder DESCENDING.
2. Die mit datenname-1 benannten Datenfelder sind die Sortierschlüssel. Die Sortierschlüssel werden von links nach rechts hierarchisch für den Sortiervorgang herangezogen, unabhängig davon, ob ASCENDING oder DESCENDING angegeben ist. Die erste Angabe von datenname-1 ist somit der Hauptsortierbegriff, die zweite Angabe von datenname-1 der nachrangige Sortierbegriff usf.
3. Ist DUPLICATES angegeben, und zwei oder mehr Tabellenelemente stimmen bezüglich aller Schlüsselfelder überein, dann wird die relative Reihenfolge dieser Tabellenelemente durch das Sortieren nicht verändert.
4. Ist DUPLICATES nicht angegeben, und zwei oder mehr Tabellenelemente stimmen bezüglich aller Schlüsselfelder überein, dann ist die relative Reihenfolge dieser Tabellenelemente nach dem Sortieren unbestimmt.
5. Die Vergleichsfolge (collating sequence), die zum Vergleich der alphanumerischen Schlüsselfelder verwendet wird, ist bei Ausführungsbeginn der SORT-Anweisung wie folgt festgelegt:
   a. Ist COLLATING SEQUENCE in der SORT-Anweisung angegeben, so ist diese Angabe Kriterium für die Sortierfolge.
   b. Ist COLLATING SEQUENCE in der SORT-Anweisung nicht angegeben, wird die programmspezifische Sortierfolge verwendet (siehe "OBJECT-COMPUTER-Paragraf").

   Für Vergleiche von nationalen Sortierfeldern wird die nationale (maschineneigene) Sortierfolge verwendet.

6. Die gemäß den ASCENDING- bzw. DESCENDING KEY-Angaben sortierten Tabellenelemente werden in der mit datenname-2 bezeichneten Tabelle abgelegt.
7. Die Elemente der Tabelle werden sortiert, indem die Inhalte der Datenfelder, die als Sortierschlüssel definiert sind, nach den Regeln für Vergleichsbedingungen verglichen werden:
   a. Wenn die Inhalte der verglichenen Schlüsselfelder nicht gleich sind und die ASCENDING-Angabe wirksam ist, dann hat das Tabellenelement, dessen Schlüsselfeld den niedrigeren Wert enthält, die niedrigere Elementnummer.
   b. Wenn die Inhalte der verglichenen Schlüsselfelder nicht gleich sind und die DESCENDING-Angabe wirksam ist, dann hat das Tabellenelement, dessen Schlüsselfeld den höheren Wert enthält, die niedrigere Elementnummer.
   c. Wenn die Inhalte der verglichenen Schlüsselfelder gleich sind, wird der nächste angegebene Sortierschlüssel zum Vergleich herangezogen.
8. Die in der SORT-Anweisung angegebenen Sortierschlüssel haben Vorrang gegenüber den ggf. in der Datenerklärung von datenname-2 angegebenen Sortierschlüsseln.
9. Mit der Angabe USING datenname-3 kann eine zweite Tabelle für den Sortiervorgang herangezogen werden.
   In diesem Fall werden die Elemente der mit datenname-3 bezeichneten Tabelle wie oben beschrieben sortiert und anschließend in die mit datenname-2 bezeichnete Tabelle gemäß den Regeln für die MOVE-Anweisung übertragen. Hinsichtlich der Übertragung der einzelnen Tabellenelemente ist Folgendes zu beachten:
   Ist das Sende-Element kürzer als das Empfangselement, wird es mit Leerzeichen aufgefüllt; ist es länger, wird es abgeschnitten.
10. Die Übertragung der sortierten Tabellenelemente von datenname-3 in die Tabelle datenname-2 endet mit dem letzten sortierten Tabellenelement (datenname-3) oder mit Erreichen der Anzahl der Tabellenelemente von datenname-2. Das bedeutet: Enthält die Tabelle datenname-3 mehr Elemente als die Tabelle datenname-2, werden die überzähligen Elemente nicht übertragen. Enthält sie weniger, bleiben die überzähligen Elemente der Tabelle datenname-2 unverändert erhalten.
11. Der Sortiervorgang verändert nicht den Inhalt der Tabelle datenname-3.

Beispiel 8-76

Sortieren einer Tabelle

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABSORT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TABELLE.
   02 TAB-ELEM OCCURS 10 TIMES.
      03 VORNAME     PIC X(8).
      03 FILLER      PIC X(3) VALUE SPACES.
      03 NAME        PIC X(10).
PROCEDURE DIVISION.
EINZIGE SECTION.
INITIALISIEREN.
    MOVE "PETER" TO VORNAME (1) MOVE "KRAUS" TO NAME (1).
    MOVE "JANE" TO VORNAME (2) MOVE "FONDA" TO NAME (2).
    MOVE "PETER" TO VORNAME (3) MOVE "FONDA" TO NAME (3).
```

```
    MOVE "KARL" TO VORNAME (4) MOVE "KRAUS" TO NAME (4).
    MOVE "UWE" TO VORNAME (5) MOVE "SEELER" TO NAME (5).
    MOVE "WALT" TO VORNAME (6) MOVE "DISNEY" TO NAME (6).
    MOVE "CLARA" TO VORNAME (7) MOVE "WIECK" TO NAME (7).
    MOVE "LEONID" TO VORNAME (8) MOVE "KOGAN" TO NAME (8).
    MOVE "ERICH" TO VORNAME (9) MOVE "FROMM" TO NAME (9).
    MOVE "ELVIS" TO VORNAME (10) MOVE "PRESLEY" TO NAME (10).
SORTIEREN.
    SORT TAB-ELEM ON ASCENDING KEY NAME VORNAME.
ENDE.
    STOP RUN.
```

Die folgende Tabelle wird ausgegeben mit `AID %DISPLAY TABELLE`

Sortierte Tabelle:

```
Elementnummer
(1)             |WALT     |    |DISNEY
(2)             |JANE     |    |FONDA
(3)             |PETER    |    |FONDA
(4)             |ERICH    |    |FROMM
(5)             |LEONID   |    |KOGAN
(6)             |KARL     |    |KRAUS
(7)             |PETER    |    |KRAUS
(8)             |ELVIS    |    |PRESLEY
(9)             |UWE      |    |SEELER
(10)            |CLARA    |    |WIECK
```

8.10.42 START-Anweisung

Gilt für **relative und indizierte Dateiorganisation**.

Funktion

Die START-Anweisung legt den logischen Ausgangspunkt innerhalb einer Datei für nachfolgende sequenzielle Leseoperationen fest.

Format

```
START  dateiname  [WITH NO LOCK]  [ KEY          { IS EQUAL TO
                                                 | IS =
                                                 | IS GREATER THAN
                                                 | IS >
                                                 | IS NOT LESS THAN
                                                 | IS NOT <
                                                 | IS GREATER THAN OR EQUAL TO
                                                 | IS >=
                                                 | IS LESS THAN
                                                 | IS <
                                                 | IS LESS THAN OR EQUAL TO
                                                 | IS <=
                                                 | IS NOT GREATER THAN
                                                 | IS NOT >
                                                 } datenname]

    [INVALID KEY unbedingte-anweisung-1]

    [NOT INVALID KEY unbedingte-anweisung-2]

    [END-START]
```

Die Formaterweiterung (WITH NO LOCK), die bei der Simultanverarbeitung wirksam wird, ist im Handbuch „COBOL2000 Benutzerhandbuch“ [1] beschrieben.

Syntaxregeln

1. Vergleichsoperatoren sind notwendige Trennsymbole. Im Format sind sie der Übersichtlichkeit wegen nicht unterstrichen.
2. dateiname muss eine Datei im sequenziellen oder dynamischen Zugriffsmodus bezeichnen.
3. datenname kann gekennzeichnet sein.
4. Für **relativ organisierte** Dateien muss datenname der Name des für diese Datei erklärten RELATIVE KEY-Datenfeldes sein.
5. Für **indiziert organisierte** Dateien muss datenname der Name eines für diese Datei erklärten RECORD KEY- oder ALTERNATE RECORD KEY-Datenfeldes oder ein alphanumerisches oder nationales Datenfeld sein, welches dem RECORD KEY- oder ALTERNATE RECORD KEY-Datenfeld untergeordnet ist. In diesem Fall muss die erste Zeichenposition beider Datenfelder identisch sein.
6. Die Angabe INVALID KEY muss vorhanden sein, falls keine entsprechende USE-Prozedur angegeben ist.

Allgemeine Regeln

Für **relative und indizierte Dateiorganisation** gilt:

1. Die durch dateiname angegebene Datei muss zum Zeitpunkt der Ausführung der START-Anweisung durch INPUT oder I-O eröffnet sein (siehe „OPEN-Anweisung").
2. Falls die Angabe KEY in der START-Anweisung fehlt, wird der Vergleichsoperator EQUAL angenommen.

3. Nach Ausführung einer START-Anweisung wird der Inhalt des FILE STATUS-Datenfeldes (falls angegeben) der Datei aktualisiert (siehe „FILE STATUS-Klausel").
4. Die Vergleichsoperatoren haben folgende Wirkung:
   a. Bei den Vergleichsoperatoren EQUAL, GREATER, GREATER OR EQUAL, NOTLESS und ihren Äquivalenten wird der Dateipositionsindikator auf den ersten Datensatz gesetzt, der die Vergleichsbedingung erfüllt.
   b. Bei den Vergleichsoperatoren LESS, LESS OR EQUAL, NOT GREATER und ihren Äquivalenten wird der Dateipositionsindikator auf den letzten Datensatz gesetzt, der die Vergleichsbedingung erfüllt.
   c. Falls für keinen Datensatz der Datei die Vergleichsbedingung erfüllt ist, tritt eine INVALID-KEY-Bedingung auf, der Dateipositionsindikator zeigt „ungültig“ an, und die START-Anweisung ist erfolglos.
5. Der weitere Programmablauf nach der START-Anweisung hängt davon ab, ob sie eine INVALID KEY- bzw. eine NOT INVALID KEY-Angabe enthält (siehe "Schlüsselfehler-Bedingung").

   Nur für **relative Dateiorganisation** gilt:

6. Die Ausführung der START-Anweisung verändert weder den Inhalt des Satzbereichs noch den des RELATIVE KEY der Datei noch den eines Datenfeldes einer DEPENDING ON-Angabe in einer RECORD-Klausel für diese Datei.
7. Die Art des Vergleichs wird durch den Vergleichsoperator in der KEY-Angabe festgelegt. Die Position eines jeden Satzes in der durch dateiname bezeichneten Datei wird mit dem Inhalt des durch datenname bezeichneten Datenfeldes (RELATIVE KEY) verglichen.

   Nur für **indizierte Dateiorganisation** gilt:

8. Die Ausführung der START-Anweisung verändert weder den Inhalt des Satzbereichs noch den des RECORD KEY der Datei noch den eines Datenfeldes einer DEPENDING ON-Angabe in einer RECORD-Klausel für diese Datei.
9. Wird in der KEY-Angabe der START-Anweisung ein Alternativschlüssel verwendet, so gilt dieser als Bezugsschlüssel für die nachfolgenden READ-Anweisungen (Format 2).
10. Die Art des Vergleichs wird durch den Vergleichsoperator in der KEY-Angabe festgelegt. Der logische Schlüssel jedes Satzes in der durch dateiname bezeichneten Datei wird mit dem Inhalt des durch datenname bezeichneten Datenfeldes (RECORD KEY oder ALTERNATE RECORD KEY) verglichen. Ist die Länge von datenname und die des RECORD KEY- bzw. ALTERNATE RECORD KEY-Datenfeldes verschieden, so wird der Vergleich so durchgeführt, als würde das größere Datenfeld von rechts auf die Länge des kleineren Datenfeldes abgeschnitten. Alle Schlüsselvergleiche erfolgen auf der Basis der maschineneigenen Sortierfolge, d.h. als wäre keine PROGRAM COLLATING SEQUENCE IS NATIVE oder gar keine PROGRAM COLLATING SEQUENCE angegeben.

8.10.43 STOP-Anweisung

Funktion

Die STOP-Anweisung beendet den Ablauf des Programms endgültig oder verursacht einen vorübergehenden Halt.

Format

```
STOP {RUN | literal}
```

Syntaxregeln

1. Das Literal kann numerisch, alphanumerisch oder jede figurative Konstante außer ALL literal sein.
2. Falls das Literal numerisch ist, muss es eine Ganzzahl ohne Vorzeichen sein.
3. Falls eine STOP-Anweisung mit RUN-Angabe in einem Programmsatz vorkommt, muss sie die einzige Anweisung in diesem Programmsatz sein oder sie muss die letzte Anweisung in einer Folge von unbedingten Anweisungen sein.
4. Die STOP-Anweisung mit RUN-Angabe beendet die Ausführung des Programms und gibt die Steuerung an das Betriebssystem zurück.
5. Wird STOP literal benutzt, so wird das Literal an dem zugeordneten Haupt- oder Nebenbedienplatz der Datenverarbeitungsanlage (dem Anlagenbediener) ausgegeben. In diesem Fall kann nur der Anlagenbediener das Programm fortsetzen. Die Fortführung des Programmes beginnt mit der nächsten ausführbaren Anweisung.

Allgemeine Regeln

1. Falls die Anzahl der Zeichen des alphanumerischen Literals größer ist als die maximal erlaubte Anzahl von Zeichen zur Ausgabe auf den Haupt- bzw. Nebenbedienplatz, wird mehr als eine physische Ausgabeoperation durchgeführt, um das Literal auszugeben.
2. Während der Ausführung einer STOP RUN-Anweisung wird für jede offene Datei eine implizite CLOSE-Anweisung ohne Wahlangaben ausgeführt. Für diese Dateien vereinbarte USE-Prozeduren werden nicht ausgeführt.

8.10.44 STRING-Anweisung

Funktion

Mit Hilfe der STRING-Anweisung wird der Inhalt von zwei oder mehreren Datenfeldern ganz oder teilweise in ein einzelnes Datenfeld übertragen und zusammengefügt (gekettet).

Format

```
STRING {{bezeichner-1 | literal-1}... [ DELIMITED BY {bezeichner-2 | literal-2 |
SIZE}]}...

    INTO bezeichner-3 [WITH POINTER bezeichner-4]

    [ON OVERFLOW unbedingte-anweisung-1]

    [NOT ON OVERFLOW unbedingte-anweisung-2]

    [END-STRING]
```

Syntaxregeln

1. Jedes Literal kann eine figurative Konstante sein, die jedoch nicht mit ALL beginnen darf.
2. Alle Literale müssen nichtnumerische Literale sein. Alle Bezeichner, mit Ausnahme von bezeichner-4, müssen implizit oder explizit mit USAGE DISPLAY oder USAGE NATIONAL beschrieben sein. Ist einer der literal-1, literal-2, bezeichner-1, bezeichner-2 oder bezeichner-3 von der Klasse national, dann müssen alle von der Klasse national sein.
3. Wenn bezeichner-1... numerische Datenelemente sind, müssen sie als ganzzahlige Datenelemente ohne das Symbol „P“ in der Maskenzeichenfolge beschrieben sein.
4. Ist DELIMITED BY nicht angegeben, so wird DELIMITED BY SIZE angenommen.
5. bezeichner-1..., literal-1... sind die Sendefelder; bezeichner-3 ist das Empfangsfeld.
6. bezeichner-3 darf kein druckaufbereitetes Datenelement sein und darf nicht mit der JUSTIFIED-Klausel beschrieben sein. bezeichner-3 darf keine stark typisierte Datengruppe sein.
7. bezeichner-4 ist ein Zählerfeld und muss ein ganzzahliges, numerisches Datenelement sein, das groß genug ist, einen Wert von der Größe von bezeichner-3 plus 1 Byte aufzunehmen. Das Symbol „P“ darf in der Maskenzeichenfolge von bezeichner-4 nicht benutzt werden.
8. bezeichner-3 darf nicht einer Teilfeldselektion unterzogen werden.

Allgemeine Regeln

1. bezeichner-2, literal-2 stellen Begrenzer dar, d.h. sie markieren eine Zeichenfolge, bis zu der der Inhalt eines Sendefeldes übertragen werden soll.
   Ist SIZE angegeben, wird das durch bezeichner-1, literal-1 definierte Datenfeld vollständig übertragen.
2. Ist als literal-1 oder literal-2 eine figurative Konstante angegeben, so wird sie als ein Datenfeld der Länge 1 betrachtet, mit der gleichen USAGE wie sie bezeichner-3 hat.
3. Beim Ausführen einer STRING-Anweisung läuft die Übertragung von Zeichen nach folgenden Regeln ab:
   a. Die Sendefelder literal-1 bzw. der Inhalt der als bezeichner-1 definierten Datenfelder werden aufeinanderfolgend in das Empfangsfeld von bezeichner-3 übertragen. Hierbei gelten die Regeln für die MOVE-Anweisung von national nach national, wenn bezeichner-3 von der Klasse national ist, anderenfalls von alphanumerisch nach alphanumerisch. In beiden Fällen findet aber kein Auffüllen mit Leerzeichen statt.
      Ist bezeichner-1 ein null-längiges Datenfeld, so wird er nicht übertragen.
   b. Ist DELIMITED BY ohne SIZE angegeben, wird der Inhalt von bezeichner-1... bzw. der Wert von literal-1 zeichenweise von links nach rechts in das Empfangsfeld übertragen. Dabei wird mit dem äußersten

linken Zeichen begonnen und die Übertragung wird solange fortgesetzt, bis entweder das Ende des jeweiligen Sendefeldes erreicht ist oder die Zeichenfolge des Begrenzers literal-2 bzw. der Inhalt von bezeichner-2 erkannt wird. Der Begrenzer wird nicht übertragen.

c. Ist DELIMITED BY SIZE angegeben oder ist bezeichner-2 ein null-längiges Datenfeld, wird der gesamte Inhalt der Sendefelder literal-1 bzw. bezeichner-1 in das Empfangsfeld bezeichner-3 übertragen, bis alle Sendefelder übertragen sind oder das Ende von bezeichner-3 erreicht ist. Die Reihenfolge der Übertragung entspricht der Reihenfolge der Sendefelder in der STRING-Anweisung.

4. Wird POINTER angegeben, ist das Zählerfeld bezeichner-4 explizit für den Benutzer verfügbar, d.h. der Anfangswert muss vom Benutzer gesetzt werden. Er darf nicht kleiner als 1 sein und muss groß genug sein, um einen Wert in der Länge von bezeichner-3 plus 1 Byte aufnehmen zu können.
5. Die Subskripte eines Bezeichners in der POINTER-Angabe werden ausgewertet, bevor die STRING-Anweisung ausgeführt wird.
6. Wird POINTER weggelassen, werden die nachfolgenden Regeln so angewendet, als ob der Benutzer bezeichner-4 mit dem Anfangswert 1 versehen hätte.
7. Bei der Übertragung in das Empfangsfeld bezeichner-3 wird jedes Zeichen einzeln aus dem Sendefeld an die Zeichenstelle von bezeichner-3, die durch den Wert des Zählerfeldes bezeichner-4 bestimmt wird, übertragen. Nach jeder Übertragung eines Zeichens wird bezeichner-4 um 1 erhöht. Der Wert von bezeichner-4 wird während der Ausführung der STRING-Anweisung nur auf diese Weise geändert.
8. Nach Ausführen der STRING-Anweisung hat nur der Teil des Empfangsfeldes bezeichner-3, in den Zeichen übertragen wurden, einen geänderten Inhalt; der Rest von bezeichner-3 bleibt unverändert.
9. Wenn zu irgendeinem Zeitpunkt während oder nach der Initialisierung der STRING-Anweisung, jedoch vor ihrer vollständigen Abarbeitung, der Wert des Zählerfeldes bezeichner-4 entweder kleiner als 1 oder größer als die Anzahl der Zeichenstellen im Empfangsfeld bezeichner-3 ist, werden keine weiteren Daten nach bezeichner-3 übertragen, sondern es wird die unbedingte Anweisung in der ON OVERFLOW-Angabe (sofern angegeben) ausgeführt.
10. Wurde die ON OVERFLOW-Angabe weggelassen und tritt eine der oben beschriebenen Bedingungen ein, geht die Steuerung zur nächsten auszuführenden Anweisung über.
11. Die unbedingte Anweisung in der NOT ON OVERFLOW-Angabe wird ausgeführt, wenn die STRING-Anweisung beendet ist, ohne dass die oben beschriebenen Bedingungen eingetreten sind.

Beispiel 8-77

```
IDENTIFICATION DIVISION.
PROGRAM-ID. STRNG.
ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SPECIAL-NAMES.
         TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  FELD1   PIC X(16)  VALUE "ANFANGSBEDINGUNG".
77  FELD2   PIC X(12)  VALUE "WERTEBEREICH".
77  FELD3   PIC X(25)  VALUE SPACES.
77  FELD4   PIC 99     VALUE 3.
PROCEDURE DIVISION.
PROC SECTION.
HAUPT.
    DISPLAY "Vor STRING: " UPON T.
    PERFORM DISPLAY-FELDER.
    STRING FELD1, FELD2 DELIMITED BY "B",
           " SETZEN" DELIMITED BY SIZE
      INTO FELD3 WITH POINTER FELD4
```

```
    ON OVERFLOW
      DISPLAY "Fehler" UPON T
    END-STRING
    DISPLAY "Nach STRING" UPON T.
    PERFORM DISPLAY-FELDER.
    STOP RUN.
DISPLAY-FELDER.
    DISPLAY "Feld1 = *" FELD1 "*" UPON T.
    DISPLAY "Feld2 = *" FELD2 "*" UPON T.
    DISPLAY "Feld3 = *" FELD3 "*" UPON T.
    DISPLAY "Feld4 = *" FELD4 "*" UPON T.
```

Ergebnis:

Vor STRING	**Nach STRING**
FELD1 = *ANFANGSBEDINGUNG*	FELD1 = *ANFANGSBEDINGUNG*
FELD2 = *WERTEBEREICH*	FELD2 = *WERTEBEREICH*
FELD3 = * *	FELD3 = * ANFANGSWERTE SETZEN *
FELD4 = *03*	FELD4 = *22*

8.10.45 SUBTRACT-Anweisung

Funktion

Die SUBTRACT-Anweisung wird benutzt, um den Wert eines numerischen Datenfeldes oder die Summe von zwei oder mehreren Werten numerischer Datenfelder von einem oder mehreren Operanden zu subtrahieren.

Format 1 der SUBTRACT-Anweisung speichert die Differenz im entsprechenden

Operanden ab. In einer SUBTRACT-Anweisung können mehrere Subtraktionen angegeben werden, indem mehr als ein Ergebnisfeld spezifiziert wird.

Format 2 der SUBTRACT-Anweisung verwendet die GIVING-Angabe.

Format 3 der SUBTRACT-Anweisung subtrahiert die Elemente einer Datengruppe von den entsprechenden Datenelementen einer anderen Gruppe.

Format 1

```
SUBTRACT {bezeichner-1 | literal-1}...
        FROM {bezeichner-n [ROUNDED]}...
        [ON SIZE ERROR unbedingte-anweisung-1]
        [NOT ON SIZE ERROR unbedingte-anweisung-2]
        [END-SUBTRACT]
```

Syntaxregeln

1. Jeder Bezeichner muss sich auf ein numerisches Datenelement beziehen.
2. Die gemeinsame Stellenzahl der Operanden, die in einer gegebenen Anweisung ermittelt wird, darf nicht mehr als 31 Ziffernstellen betragen (siehe "Arithmetische Anweisungen").
3. Alle Literale und Bezeichner, die dem Wort FROM vorangehen, werden addiert, und die Summe wird vom derzeitigen Wert von bezeichner-n... abgezogen. Die Ergebnisse der Subtraktion werden als neue Werte von bezeichner-n abgespeichert.

Für weitere Regeln siehe unter "Angaben in Anweisungen"; die ROUNDED-Angabe und die (NOT) ON SIZE ERROR-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-78

Anweisung	Maskenzeichenfolge desErgebnisfeldes	Rechnung
`SUBTRACT A, B FROM D`	999	D - (A + B) abgespeichert in D als nnn
`SUBTRACT A FROM B,C`	9(3) für B 9(5) für C	B - A abgespeichert in B als nnn C - A abgespeichert in C als nnnnn

Format 2

```
SUBTRACT {bezeichner-1 | literal-1}... FROM {bezeichner-m | literal-m}
         GIVING {bezeichner-n [ROUNDED]}...
        [ON SIZE ERROR unbedingte-anweisung-1]
```

```
[NOT ON SIZE ERROR unbedingte-anweisung-2]

[END-SUBTRACT]
```

Syntaxregeln

1. Jeder Bezeichner, der dem Wort GIVING vorangeht, muss sich auf ein numerisches Datenelement beziehen.
2. bezeichner-n kann sich sowohl auf ein numerisches Datenelement als auch auf ein numerisch druckaufbereitetes Datenelement beziehen.
3. Die gemeinsame Stellenzahl der Operanden, die in einer gegebenen Anweisung ermittelt wird, mit Ausnahme der Datenfelder, die dem Wort GIVING folgen, darf nicht mehr als 31 Ziffernstellen betragen (siehe "Arithmetische Anweisungen").
4. Alle Literale oder Bezeichner, die dem Wort FROM vorangehen, werden addiert, und die Summe wird von literal-m oder bezeichner-m subtrahiert. Das Ergebnis der Subtraktion ergibt den neuen Wert von bezeichner-n... .

Für weitere Regeln siehe unter "Angaben in Anweisungen"; die GIVING-Angabe, ROUNDED-Angabe und (NOT) ON SIZE ERROR-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-79

Anweisung	Maskenzeichenfolge desErgebnisfeldes (C)	Rechnung
`SUBTRACT A,B FROM` `100 GIVING C.`	9(5)	100 - (A + B) abgespeichert in C als nnnnn.

Format 3

```
SUBTRACT { CORR | CORRESPONDING } bezeichner-1

         FROM bezeichner-2 [ROUNDED]

        [ON SIZE ERROR unbedingte-anweisung-1]

        [NOT ON SIZE ERROR unbedingte-anweisung-2]

        [END-SUBTRACT]
```

Syntaxregeln

1. Jeder Bezeichner muss sich auf eine Datengruppe beziehen.
2. Die gemeinsame Stellenzahl der Operanden, die für jedes Paar von entsprechenden Datenfeldern getrennt bestimmt wird, darf nicht mehr als 31 Ziffernstellen betragen (siehe "Arithmetische Anweisungen").
3. Die Datenelemente innerhalb des ersten Operanden (bezeichner-1) werden von den entsprechenden Datenelementen innerhalb des zweiten Operanden (bezeichner-2) subtrahiert. Die Ergebnisse werden in den Feldern des zweiten Operanden abgespeichert.

Für weitere Regeln siehe unter "Angaben in Anweisungen"; die CORRESPONDING-Angabe, ROUNDED-Angabe und (NOT) ON SIZE ERROR-Angabe sind in diesem Abschnitt beschrieben.

Beispiel 8-80

Siehe die Beschreibung der CORRESPONDING-Angabe als Beispiel für die Verwendung dieses Zusatzes ("CORRESPONDING-Angabe").

8.10.46 UNSTRING-Anweisung

Funktion

Durch die UNSTRING-Anweisung werden aufeinanderfolgende Daten aus einem Sendefeld getrennt (entkettet) und in mehrere Empfangsfelder übertragen.

Format

```
UNSTRING bezeichner-1

    [DELIMITED BY [ALL] {bezeichner-1 | literal-1} [OR [ALL] {bezeichner-3 |
    literal-2}]...]

     INTO {bezeichner-4 [DELIMITER IN bezeichner-5] [COUNT IN bezeichner-6]}...

    [WITH POINTER bezeichner-7] [TALLYING IN bezeichner-8]

    [ON OVERFLOW unbedingte-anweisung-1]

    [NOT ON OVERFLOW unbedingte-anweisung-2]

    [END-UNSTRING]
```

Syntaxregeln

1. literal-1 und literal-2 müssen Literale von der Kategorie alphanumerisch oder national sein. Sie dürfen aber keine figurativen Konstanten sein, die mit ALL beginnen.
2. bezeichner-1, bezeichner-2, bezeichner-3, bezeichner-5 müssen Datenfelder von der Kategorie alphanumerisch oder national sein.
3. Ist irgendein literal-1, literal-2, bezeichner-1, bezeichner-2, ..., bezeichner-5 von der Kategorie national, dann müssen alle von der Kategorie national sein.
4. bezeichner-4 darf sein:
   - von der Kategorie national
   - von der Kategorie alphabetisch, alphanumerisch oder numerisch. Er muss dann explizit oder implizit mit USAGE DISPLAY beschrieben sein. Ein numerisches Datenfeld darf nicht das Zeichen ’P’ in der Maskenzeichenfolge haben.
5. bezeichner-6, bezeichner-7, bezeichner-8 müssen als ganzzahlige, numerische Datenelemente beschrieben sein.
   Das Symbol "P“ darf in der PICTURE-Maskenzeichenfolge nicht angegeben sein.
6. bezeichner-1 ist das Sendefeld.
7. bezeichner-4 ist das Empfangsfeld; bezeichner-5 ist das Empfangsfeld für Begrenzer.
8. bezeichner-1 darf nicht einer Teilfeldselektion unterzogen werden.
9. DELIMITER IN und COUNT IN können nur im Zusammenhang mit DELIMITED BY angegeben werden.
10. Kein Bezeichner darf mit der Stufennummer 88 definiert sein.

Allgemeine Regeln

1. Alle Bezugnahmen auf bezeichner-2 und literal-1gelten ebenso für bezeichner-3 und literal-2 sowie für alle ihre Wiederholungen.
2. Ist bezeichner-1 ein null-längiges Datenfeld:
   a. Der Inhalt von bezeichner-4 bis bezeichner-8 bleibt unverändert.
   b. Es wird weder unbedingte-anweisung-1 noch unbedingte-anweisung-2 ausgeführt.
   c. Die Ablaufsteuereung geht sofort zum Ende der UNSTRING-Anweisung über.

3. Ist als literal-1 eine figurative Konstante angegeben, so steht sie für ein 1 Zeichen langes nationales Datenfeld, wenn bezeichner-1 ein nationales Datenfeld ist, andernfalls für ein 1 Zeichen langes alphanumerisches Datenfeld.
4. Ist ALL angegeben, werden aufeinanderfolgende Wiederholungen von literal-1 bzw. bezeichner-2 so gewertet, als wären sie nur einmal aufgetreten, und literal-1 bzw. bezeichner-2 wird nur einmal nach bezeichner-5 übertragen.
5. Folgen zwei Begrenzer unmittelbar aufeinander, wird das aktuelle Empfangsfeld mit Leerzeichen gefüllt, wenn es alphabetisch, alphanumerisch oder national beschrieben ist. Es wird mit Nullen gefüllt, wenn es numerisch beschrieben ist.
6. literal-1, bezeichner-2 stellen Begrenzer dar. Besteht ein Begrenzer aus zwei oder mehr Zeichen, müssen alle Zeichen in der gleichen zusammenhängenden Reihenfolge wie im Sendefeld auftreten, damit sie als Begrenzer erkannt werden.
   Ist bezeichner-2 ein null-längiges Datenfeld, so wird er nicht als Begrenzer erkannt.
7. Sind zwei oder mehr Begrenzer in der DELIMITED BY-Angabe spezifiziert, so wird diese mit OR verknüpft. Jeder Begrenzer wird mit dem Sendefeld verglichen. Ist der Vergleich positiv, d.h. die gesuchten Zeichen sind im Sendefeld gefunden worden, gelten sie als einzelne Begrenzer. Kein Zeichen im Sendefeld kann Bestandteil von mehr als einem Begrenzer sein. Begrenzer können sich nicht überschneiden.
   Die Begrenzer werden mit dem Sendefeld in der Reihenfolge verglichen, in der sie in der UNSTRING-Anweisung angegeben sind.
8. Nach Initialisierung der UNSTRING-Anweisung ist bezeichner-4 der aktuelle Empfangsbereich. Die Daten werden von bezeichner-1 nach bezeichner-4 entsprechend den folgenden Regeln übertragen:
   a. Ist POINTER angegeben, beginnt die Prüfung mit der Zeichenfolge von bezeichner-1 mit der relativen Zeichenposition, die durch bezeichner-7 angezeigt wird.

      Ist POINTER nicht angegeben, beginnt die Prüfung mit dem äußersten linken Zeichen von bezeichner-1.
   b. Ist DELIMITED BY angegeben, wird die Prüfung von links nach rechts fortgesetzt, bis ein Begrenzer entweder spezifiziert durch den Wert von literal-1 oder das Datenfeld von bezeichner-2 angetroffen wird.

      Ist DELIMITED BY nicht angegeben, ist die Anzahl der geprüften Zeichen genauso groß wie die Größe des aktuellen Empfangsbereichs.

      Ist jedoch das Vorzeichen des Empfangsfeldes so spezifiziert, dass es eine eigene Zeichenposition belegt, ist die Anzahl der Zeichen um eins geringer als die Größe des aktuellen Empfangsbereichs.

      Ist das Ende von bezeichner-1 erreicht, ehe ein Begrenzer gefunden wurde, endet die Prüfung mit dem zuletzt geprüften Zeichen.
   c. Die Zeichenfolge, die bis zum Begrenzer ermittelt wurde, wird wie ein nationales Datenelement behandelt, wenn bezeichner-1 von der Kategorie national ist.
      Andernfalls wird sie wie ein alphanumerisches Datenelement behandelt. Entsprechend den Regeln der MOVE-Anweisung (siehe "MOVE-Anweisung" "MOVE-Anweisung") wird dieses in den aktuellen Empfangsbereich übertragen, ein Dezimalpunkt wird jedoch ignoriert.
   d. Ist DELIMITER IN angegeben, werden die Begrenzungszeichen wie ein nationales Datenelement behandelt, wenn bezeichner-1 von der Kategorie national ist.
      Andernfalls werden sie wie ein alphanumerisches Datenelement behandelt. Entsprechend den Regeln der MOVE-Anweisung (siehe "MOVE-Anweisung" "MOVE-Anweisung") wird dieses in den aktuellen Empfangsbereich übertragen.
      Wenn das Ende von bezeichner-1 erreicht wurde, ohne einen Begrenzer zu finden, werden leerzeichen in bezeichner-5 übertragen.
   e. Ist COUNT IN angegeben, wird die Anzahl der geprüften Zeichen (ausgenommen evtl. vorhandene Begrenzungszeichen) in den Bereich von bezeichner-6 nach den Regeln der Elementübertragung übertragen.

f. Ist DELIMITED BY angegeben, wird die Überprüfung von bezeichner-1 mit dem ersten Zeichen rechts vom Begrenzer fortgesetzt.

   Ist DELIMITED BY nicht angegeben, wird die Überprüfung mit dem nächsten Zeichen nach dem letzten übertragenen Zeichen fortgesetzt.

g. Nachdem die Daten nach bezeichner-4 übertragen worden sind, ist die Wiederholung von bezeichner-4 der aktuelle Empfangsbereich.
   Die beschriebene Prozedur wird solange wiederholt, bis alle Zeichen im bezeichner-1 verarbeitet oder keine Empfangsbereiche mehr vorhanden sind.

9. Die Anfangswerte für die Datenfelder mit der POINTER- oder TALLYING-Angabe müssen vom Benutzer gesetzt werden.
10. Der Inhalt von bezeichner-7 wird bei jedem im Datenfeld von bezeichner-1 geprüften Zeichen um 1 erhöht.

    Wenn die UNSTRING-Anweisung mit POINTER-Angabe ausgeführt ist, enthält bezeichner-7 einen Wert, der dem Anfangswert, zuzüglich der Anzahl der im Datenfeld von bezeichner-1 geprüften Zeichen entspricht.
11. Wenn eine UNSTRING-Anweisung mit TALLYING-Angabe ausgeführt ist, enthält bezeichner-8 einen Wert, der dem Anfangswert, zuzüglich der Anzahl der Empfangsfelder, die angesprochen wurden, entspricht.
12. Durch jede der folgenden Situationen wird eine OVERFLOW-Bedingung verursacht:
    a. Bei der Ausführung von UNSTRING ist der Wert im Datenfeld von bezeichner-7 kleiner als 1 oder größer als die Länge von bezeichner-1.
    b. Während der Ausführung der UNSTRING-Anweisung sind alle Datenempfangsbereiche angesprochen worden, das Datenfeld von bezeichner-1 enthält jedoch noch Zeichen, die nicht geprüft worden sind.
13. Wenn eine OVERFLOW-Bedingung vorliegt, wird die UNSTRING-Anweisung beendet.
    Ist ON OVERFLOW angegeben, wird die in dieser Angabe enthaltene unbedingte Anweisung ausgeführt. Wurde die ON OVERFLOW-Angabe weggelassen, geht die Steuerung zur nächsten auszuführenden Anweisung über.
14. Die unbedingte Anweisung in der NOT ON OVERFLOW-Angabe wird ausgeführt, wenn die UNSTRING-Anweisung beendet und keine der oben beschriebenen Bedingungen eingetreten ist.
15. Die Auswertung der Subskribierung und Indizierung für die Bezeichner geschieht wie folgt:
    a. Sind die Felder bezeichner-1, bezeichner-7, bezeichner-8 subskribiert oder indiziert, wird der Indexwert für diese Felder nur einmal berechnet und zwar unmittelbar vor der Ausführung der UNSTRING-Anweisung.
    b. Jede Subskribierung von Bezeichnern in der DELIMITED BY-, INTO-, DELIMITER IN- und COUNT-Angabe wird ausgewertet, bevor die Datenübertragung in das jeweilige Datenfeld erfolgt.

Beispiel 8-81

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. UNSTRING.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     TERMINAL IS T.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  FELD      PIC  X(12)  VALUE  "ABCDEFGHIJKL".
 01  BEREICH.
     02  TEIL1 PIC X       VALUE SPACES.
     02  TEIL2 PIC XX      VALUE SPACES.
     02  TEIL3 PIC XXX     VALUE SPACES.
 01  ZAHL      PIC 99      VALUE ZERO.
```

```
PROCEDURE DIVISION.
PROC SECTION.
HAUPT.
    DISPLAY "Vor UNSTRING: " UPON T.
    PERFORM DISPLAY-FELDER.
*
    UNSTRING FELD DELIMITED BY "E" OR "H" OR "K" OR "L"
        INTO TEIL3, TEIL2, TEIL1
        TALLYING IN ZAHL.
    END-UNSTRING
*
DISPLAY "Nach UNSTRING: " UPON T.
    PERFORM DISPLAY-FELDER.
    STOP RUN.
DISPLAY-FELDER.
    DISPLAY "Feld  = *" FELD  "*" UPON T.
    DISPLAY "Teil1 = *" TEIL1 "*" UPON T.
    DISPLAY "Teil2 = *" TEIL2 "*" UPON T.
    DISPLAY "Teil3 = *" TEIL3 "*" UPON T.
    DISPLAY "Zahl  = *" ZAHL  "*" UPON T.
```

Ergebnis:

Vor UNSTRING	**Nach UNSTRING**
FELD = *ABCDEFGHIJKL*	FELD = *ABCDEFGHIJKL*
TEIL1 = * *	TEIL1 = *I*
TEIL2 = * *	TEIL2 = *FG*
TEIL3 = * *	TEIL3 = *ABC*
ZAHL = *00*	ZAHL = *03*

8.10.47 USE-Anweisung

Funktion

Die USE-Anweisung leitet Prozedurvereinbarungen ein und legt die Bedingungen für deren Ausführung fest. Die USE-Anweisung selbst wird jedoch nicht ausgeführt.

Format 1 vereinbart Benutzerkennsatzroutinen.

Format 2 vereinbart Prozeduren, die durchlaufen werden sollen, falls für eine Datei ein Ein-/Ausgabe-Fehler auftritt.

Format 3 wie Format 2, aber mit GLOBAL-Anweisung. Gilt nur für geschachtelte Programme.

Format 4 vereinbart Prozeduren, die durchlaufen werden sollen, wenn ein Ausnahmezustand ausgelöst worden ist.

Format 5 vereinbart Prozeduren, die vom Listenprogrammsteuersystem (Report Writer) vor der Listenausgabe durchlaufen werden sollen (siehe Kapitel „Listenprogramm (Report-Writer)").

Format 1 für sequenzielle Dateiorganisation

```
USE {BEFORE | AFTER} STANDARD [ENDING | BEGINNING] [REEL | UNIT | FILE] LABEL
PROCEDURE ON {dateiname-1}...
```

Syntaxregeln

1. dateiname-1 muss in einer Dateierklärung (FD) der DATA DIVISION des Programms enthalten sein.
2. dateiname-1 darf nicht der Name einer Sortierdatei sein.
3. Mit dateiname-1 wird diejenige Dateierklärung bezeichnet, für die die angegebenen Kennsatzprozeduren durchgeführt werden sollen.
4. Die angegebenen Prozeduren werden im Zusammenhang mit OPEN- und CLOSE-Anweisungen für die Datei durchlaufen.
5. Nach Durchlaufen einer USE-Prozedur wird das Programm bei der aufrufenden Routine fortgesetzt.
6. BEFORE oder AFTER bezeichnen die Art der zu verarbeitenden Kennsätze.

   BEFORE bedeutet, dass nicht standardisierte Kennsätze verarbeitet werden sollen (solche Kennsätze können nur für Magnetbanddateien angegeben werden).

   AFTER bedeutet, dass auf Standard-Dateikennsätze standardisierte Benutzerkennsätze folgen und diese verarbeitet werden sollen.
7. BEGINNING oder ENDING zeigt an, dass entweder Anfangs- oder Endekennsätze verarbeitet werden sollen.

   Falls die Angaben BEGINNING und ENDING fehlen, werden die vereinbarten Prozeduren sowohl für Anfangs- als auch Endekennsätze durchlaufen.
8. Die Angaben REEL, UNIT oder FILE bedeuten, dass die vereinbarten Prozeduren durchlaufen werden sollen, wenn Datenträger-, Spulen- oder Dateikennsätze vorhanden sind.

   Die Angabe REEL gilt nicht für Plattenspeicherdateien. Die Angabe UNIT ist nicht anwendbar für Dateien im wahlfreien Zugriffsmodus, da in diesem Fall nur Dateikennsätze verarbeitet werden. Vom Compiler werden die Angaben REEL und UNIT gleich behandelt.
9. Fehlen die obigen Angaben, werden die vereinbarten Prozeduren, je nach Art des Datenträgers, entweder für Spulen- und Dateikennsätze oder für Datenträger- und Dateikennsätze durchlaufen.
10. Für zeilensequenzielle Dateien ist die USE-Anweisung vom Format 1 nicht zulässig.

Allgemeine Regeln

1. Die für eine Datei zu bearbeitenden Kennsätze müssen innerhalb der Dateierklärung einer Datei als Datennamen in der LABEL RECORDS-Klausel aufgeführt werden und als 01 Datenfelder innerhalb der Dateierklärung oder der LINKAGE SECTION definiert werden.
2. Der gleiche Dateiname kann in verschiedenen Variationen der USE-Anweisung von Format 1 auftreten. Jedoch darf dies nicht den gleichzeitigen Aufruf mehrerer Prozedurvereinbarungen bewirken.
3. Falls die Angabe dateiname-1 benutzt wird, darf innerhalb der Dateierklärung für diese Datei nicht die LABEL RECORDS-Klausel mit OMITTED-Angabe vorhanden sein.
4. Für *externe* Dateien können keine Benutzerkennsatzroutinen vereinbart werden.
5. Die Standardsystemprozeduren werden für alle Standardkennsätze durchlaufen, die aus System- und Benutzerkennsätzen bestehen.
   a. Kennsätze von Eingabe- oder Ein-/Ausgabe-Dateien werden in der folgenden Reihenfolge überprüft:
      i. Die Standard-Anfangskennsätze werden vom Ein-/Ausgabe-System überprüft.
      ii. Benutzer-Anfangskennsätze werden von eventuell vorhandenen USE-Prozeduren überprüft.
      iii. Die Standard-Endekennsätze werden vom Ein-/Ausgabe-System überprüft.
      iv. Benutzer-Endekennsätze werden von eventuell vorhandenen USE-Prozeduren überprüft.
   b. Kennsätze für Ausgabedateien werden in folgender Reihenfolge erzeugt:
      i. Die Standard-Anfangskennsätze werden vom Ein-/Ausgabe-System erzeugt.
      ii. Benutzer-Anfangskennsätze werden von eventuell vorhandenen USE-Prozeduren erzeugt.
      iii. Bevor die Benutzer-Anfangskennsätze ausgegeben werden, wird überprüft, ob sie mit den Zeichen UHL beginnen.
      iv. Die Standard-Endekennsätze werden vom Ein-/Ausgabe-System erzeugt.
      v. Benutzer-Endekennsätze werden von eventuell vorhandenen USE-Prozeduren erstellt.
      vi. Bevor die Benutzer-Endekennsätze ausgegeben werden, wird überprüft, ob sie mit den Zeichen UTL beginnen.
6. Innerhalb einer USE-Prozedur darf keine Bezugnahme auf Prozeduren außerhalb der Prozedurvereinbarungen (DECLARATIVES) enthalten sein mit Ausnahme der PERFORM-Anweisung.

   Auf Prozedurnamen, die einer USE-Anweisung untergeordnet sind, darf aus einer anderen Prozedur oder außerhalb der Prozedurvereinbarungen (DECLARATIVES) nur mit einer PERFORM-Anweisung Bezug genommen werden.
7. Der Ausgang für eine Prozedurvereinbarung vom Format 1 wird durch den Compiler hinter der letzten Anweisung des entsprechenden Kapitels erzeugt. Alle logischen Programmabläufe innerhalb des Kapitels müssen auf diesen Ausgang führen.
8. Zur Allgemeinen Regel 7. gibt es eine Ausnahme:

   Die GO TO-Anweisung mit MORE-LABELS-Angabe kann als spezieller Ausgang benutzt werden. Nach Ausführung dieser Anweisung werden vom Laufzeitsystem die folgenden Schritte durchlaufen:
   a. Falls Kennsätze erzeugt werden, werden die aktuellen Anfangs- oder Endekennsätze geschrieben; danach wird das Programm am Anfang der Prozedurvereinba
      rung fortgesetzt, um weitere Kennsätze erzeugen zu können. Der Benutzer ist dafür verantwortlich, dass die letzte Anweisung des Kapitels durchlaufen wird, nachdem die Kennsätze erzeugt worden sind. Hierbei ist zu beachten, dass nach Ausführung der GO TO-Anweisung mit MORE-LABELS-Angabe auf alle Fälle ein Kennsatz geschrieben wird; falls vom Benutzer kein neuer Kennsatz zur Verfügung gestellt wurde, wird vom Laufzeitsystem ein Hilfskennsatz analog Allgemeine Regel 7. erzeugt.
   b. Falls Kennsätze überprüft werden, wird vom System ein zusätzlicher Anfangs- oder Endekennsatz gelesen; danach wird das Programm am Anfang der Prozedurvereinbarung fortgesetzt, um weitere Kennsätze überprüfen zu können. Während der Verarbeitung von Benutzerkennsätzen wird jedoch die

Prozedurvereinbarung nur dann angesprungen, wenn weitere Benutzerkennsätze vorliegen. In diesem Falle muss der Programmierer also keinen Programmablauf durch die letzte Anweisung des Kapitels vorsehen. Wenn jedoch nicht standardisierte Kennsätze verarbeitet werden, ist dem System deren Anzahl nicht bekannt. Daher muss in diesem Fall die letzte Anweisung des Kapitels durchlaufen werden, um die Kennsatzverarbeitung abzuschließen.

Beispiel 8-82

In diesem Beispiel behandelt eine Prozedurvereinbarung (ALPHA) Anfangskennsätze und eine andere (BETA) Endekennsätze. Die Prozedurverbarungen werden sowohl im Eingabe- als auch Ausgabemodus durchlaufen.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. USESEQ.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT BEISPIEL-DATEI ASSIGN TO "TESTBAND".
DATA DIVISION.
FILE SECTION.
FD  BEISPIEL-DATEI,
    RECORD CONTAINS 100 CHARACTERS,
    LABEL RECORD IS BEISPIEL-KENNSATZ,
    DATA RECORD IS BEISPIEL-SATZ.
01  BEISPIEL-KENNSATZ.
    02  KENNSATZ-KENNUNG         PICTURE X(4).
    02  KENNSATZ-INHALT          PICTURE X(76).
01  BEISPIEL-SATZ                PICTURE X(100).
WORKING-STORAGE SECTION.
77  EIN-AUSGABE-INDIKATOR PICTURE X.
    88  EINGABEMODUS VALUE "I".
    88  AUSGABEMODUS VALUE "O".
77  KENNSATZ-ZAEHLER PICTURE 9.
PROCEDURE DIVISION.
DECLARATIVES.
ALPHA SECTION.
    USE AFTER STANDARD BEGINNING FILE
        LABEL PROCEDURE ON BEISPIEL-DATEI.
ALPHA-1.
    IF EINGABEMODUS
    THEN
      DISPLAY "512010 KENNSATZ GELESEN:-" BEISPIEL-KENNSATZ
              UPON T
      GO TO MORE-LABELS
    ELSE
      IF KENNSATZ-ZAEHLER = 0
      THEN
        MOVE "UHL1" TO KENNSATZ-KENNUNG
        MOVE "1. KENNSATZ ERZEUGT DURCH ALPHA."
             TO KENNSATZ-INHALT
        DISPLAY "511030 KENNSATZ ERZEUGT:-" BEISPIEL-KENNSATZ
                UPON T
        MOVE 1 TO KENNSATZ-ZAEHLER
        GO TO MORE-LABELS
      ELSE
```

```
        MOVE "UHL2" TO KENNSATZ-KENNUNG
        MOVE "2. KENNSATZ ERZEUGT DURCH ALPHA" TO KENNSATZ-INHALT
        DISPLAY "511530 WEITERER KENNSATZ ERZEUGT:-"
                BEISPIEL-KENNSATZ UPON T
        MOVE 2 TO KENNSATZ-ZAEHLER
      END-IF
    END-IF.
ALPHA-END.
    EXIT.
BETA SECTION.
    USE AFTER STANDARD ENDING FILE
        LABEL PROCEDURE ON BEISPIEL-DATEI.
BETA-1.
    IF EINGABEMODUS
    THEN
      DISPLAY "522010 KENNSATZ GELESEN: " BEISPIEL-KENNSATZ
               UPON T
      GO TO MORE-LABELS
    ELSE
      IF KENNSATZ-ZAEHLER = 0
      THEN
        MOVE "UTL1" TO KENNSATZ-KENNUNG
        MOVE "1. KENNSATZ ERZEUGT" TO KENNSATZ-INHALT
        DISPLAY "521030 KENNSATZ ERZEUGT:-" BEISPIEL-KENNSATZ
                UPON T
        MOVE 1 TO KENNSATZ-ZAEHLER
        GO TO MORE-LABELS
      ELSE
        MOVE "UTL2" TO KENNSATZ-KENNUNG
        MOVE "2. KENNSATZ ERZEUGT DURCH BETA" TO KENNSATZ-INHALT
        DISPLAY "521530 WEITERER KENNSATZ ERZEUGT: "
                 BEISPIEL-KENNSATZ UPON T
        MOVE 2 TO KENNSATZ-ZAEHLER
      END-IF
    END-IF.
BETA-END.
    EXIT.
END DECLARATIVES.
GAMMA SECTION.
AUF-GEHTS.
    MOVE "O" TO EIN-AUSGABE-INDIKATOR.
    OPEN OUTPUT BEISPIEL-DATEI
    CLOSE BEISPIEL-DATEI
    STOP RUN.
```

Beispiel 8-83

In diesem Beispiel sind UHL-FELD, KENNSATZ-NR und BENUTZER-INFO gemeinsame Kennsatzfelder. Daher ist es erlaubt, das Feld UHL-FELD im Paragrafen L-1 ohne Kennzeichnung zu verwenden.

Angaben in der DATA DIVISION:

```
FD  DATEI-1
    ...
    LABEL RECORD IS KENNSATZ-1
    ...
```

```
01  KENNSATZ-1.
    02 UHL-FELD PIC X(3).
    02 KENNSATZ-NR PIC 9.
    02 BENUTZER-INFO PIC X(76).
01  SATZ-1.
    ...
FD  DATEI-2
    ...
    LABEL RECORD IS KENNSATZ-2
    ...
01  KENNSATZ-2.
    02 UHL-FELD PIC X(3).
    02 KENNSATZ-NR PIC 9.
    02 BENUTZER-INFO PIC X(76).
01  SATZ-2.
```

Anweisungen in der PROCEDURE DIVISION:

```
L SECTION.
    USE AFTER STANDARD LABEL PROCEDURE ON INPUT.
L-1.
    IF UHL-FELD NOT = "UHL" THEN STOP RUN.
    ...
    GO TO MORE-LABELS.
L-9.
    EXIT.
M SECTION.
    ...
EROEFFNEN SECTION.
OEFFNEN.
    OPEN INPUT DATEI-1, DATEI-2.
```

Format 2 für alle Dateiorganisationsformen

```
USE AFTER STANDARD {ERROR | EXCEPTION} PROCEDURE ON { {dateiname-1}... | INPUT |
OUTPUT | I-O | EXTEND}
```

Syntaxregeln

1. Die Angaben ERROR und EXCEPTION sind gleichbedeutend und können wahlweise verwendet werden.
2. dateiname-1 darf nur in einer USE-Anweisung angegeben werden. INPUT, OUTPUT, I-O und EXTEND dürfen höchstens einmal angegeben werden.
3. Die Dateien, auf die in der USE-Anweisung implizit (INPUT, OUTPUT, I-O und EXTEND) oder explizit (dateiname-1, dateiname-2,...) Bezug genommen wird, brauchen nicht dieselbe Organisation und denselben Zugriff zu haben.

Allgemeine Regeln

1. Die USE-Prozeduren werden durchlaufen:

a. bei Auftreten einer Ende-Bedingung, wenn die Ein-/Ausgabe-Anweisung, bei der diese Bedingung auftrat, keine AT END-Angabe enthält, oder bei Auftreten einer Schlüsselfehler- Bedingung, wenn die Ein-/Ausgabe-Anweisung, bei der diese Bedingung auftrat, keine INVALID KEY-Angabe enthält.

b. bei Auftreten eines schwerwiegenden Fehlers - FILE STATUS CODE 30.

Treffen a) oder b) zu und fehlt eine entsprechende USE-Prozedur für die Datei, führt das zu Programmabbruch.

2. Bei Angabe von dateiname-1 werden die Fehlerbehandlungsprozeduren nur für die genannten Dateien durchlaufen. Für diese Dateien werden keine anderen USE-Prozeduren durchlaufen.
3. Vor Ausführung der Benutzerfehlerroutine werden die Standardsystemroutinen zur Behandlung von Ein-/Ausgabe-Fehlern durchlaufen.
4. Nach Durchlaufen einer USE-Prozedur wird das Programm bei der laufenden Routine fortgesetzt.
5. INPUT bedeutet, dass die angegebenen Prozeduren nur für Dateien durchlaufen werden, die sich im Eingabemodus befinden (OPEN-Anweisung mit INPUT-Angabe).
6. OUTPUT bedeutet, dass die angegebenen Prozeduren nur für Dateien durchlaufen werden, die sich im Ausgabemodus befinden (OPEN-Anweisung mit OUTPUT-Angabe).
7. I-O bedeutet, dass die angegebenen Prozeduren nur für Dateien durchlaufen werden, die sich im Aktualisierungsmodus befinden (OPEN-Anweisung mit I-O-Angabe).
8. EXTEND bedeutet, dass die angegebenden Prozeduren nur für Dateien durchlaufen werden, die sich im Erweiterungsmodus befinden (OPEN-Anweisung mit EXTEND-Angabe).
9. Innerhalb einer USE-Prozedur darf keine Bezugnahme auf Prozeduren außerhalb der Prozedurvereinbarungen (DECLARATIVES) enthalten sein, mit Ausnahme der PERFORM- und der RESUME-Anweisung.
10. Auf Prozedurnamen, die einer USE-Anweisung untergeordnet sind, darf aus einer anderen Prozedur oder außerhalb der Prozedurvereinbarungen (DECLARATIVES) nur mit einer PERFORM-Anweisung Bezug genommen werden.
11. Die Anweisungen innerhalb einer USE-Prozedur dürfen nicht dazu führen, dass eine noch aktive USE-Prozedur erneut aktiviert wird.

Beispiel 8-84

```
IDENTIFICATION DIVISION.
PROGRAM-ID. USESEQ2.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OPTIONAL EINGABE ASSIGN "EINGABE"
           FILE STATUS ANZEIGE.
DATA DIVISION.
FILE SECTION.
FD  EINGABE.
01  SATZ                    PIC X(80).
WORKING-STORAGE SECTION.
01  ANZEIGE PIC XX.
01  CLOSE-INDIKATOR         PIC X VALUE "0".
    88  DATEI-ZU            VALUE "0".
    88  DATEI-OFFEN         VALUE "1".
PROCEDURE DIVISION.
```

```
DECLARATIVES.
EINGABE-FEHLER SECTION.
    USE AFTER ERROR PROCEDURE ON EINGABE.
STATUS-ABFRAGE.
    IF ANZEIGE = "10"
       DISPLAY "Dateiende von Eingabe erreicht" UPON T
    ELSE
       DISPLAY "Nicht behebbarer Fehler (" ANZEIGE
               ") FUER DATEI EINGABE" UPON T
       IF DATEI-OFFEN
          CLOSE EINGABE
       END-IF
       DISPLAY "Programm abnormal beendet" UPON T
       STOP RUN
    END-IF.
END DECLARATIVES.
ANFANG SECTION.
ARBEIT.
    OPEN INPUT EINGABE
    SET DATEI-OFFEN TO TRUE
    READ EINGABE
    CLOSE EINGABE WITH LOCK
    SET DATEI-ZU TO TRUE
    OPEN INPUT EINGABE
    CLOSE EINGABE
    STOP RUN.
```

Beispiel 8-85

```
IDENTIFICATION DIVISION.
PROGRAM-ID. USEREL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OPTIONAL WORKFILE ASSIGN TO "WORKFILE"
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS DYNAMIC
    RELATIVE KEY IS RELSCHL
    FILE STATUS ANZEIGE.
DATA DIVISION.
FILE SECTION.
FD  WORKFILE.
01  SATZ.
    03  INHALT                PIC X(100).
WORKING-STORAGE SECTION.
01  ANZEIGE                   PIC XX.
01  RELSCHL                   PIC 9(4).
01  CLOSE-INDIKATOR           PIC X VALUE "0".
    88  DATEI-ZU              VALUE "0".
    88  DATEI-OFFEN           VALUE "1".
PROCEDURE DIVISION.
DECLARATIVES.
```

```
EINGABE-FEHLER SECTION.
    USE AFTER ERROR PROCEDURE ON WORKFILE.
STATUS-ABFRAGE.
    EVALUATE ANZEIGE
    WHEN    "10"
      DISPLAY "Dateiende von WORKFILE erreicht" UPON T
    WHEN    "22"
      DISPLAY "Satz mit Schluessel" RELSCHL "SCHON VORHANDEN" UPON T
    WHEN    "23"
      DISPLAY "Satz mit Schluessel" RELSCHL "NICHT VORHANDEN" UPON T
    WHEN OTHER
      DISPLAY "Nicht behebbarer Fehler "(" ANZEIGE ")"
              "FUER DATEIEINGABE" UPON T
      IF DATEI-OFFEN
      THEN
        CLOSE WORKFILE
      END-IF
      DISPLAY "Programm abnormal beendet" UPON T
      STOP RUN
    END-EVALUATE.
END-DECLARATIVES.
HAUPT SECTION.
AUF-ZU.
    ...
    STOP RUN.
```

Beispiel 8-86

```
IDENTIFICATION DIVISION.
PROGRAM-ID. USEIND.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OPTIONAL DATEI1 ASSIGN TO "EIN-AUS"
          ORGANIZATION IS INDEXED
          RECORD KEY ISAMSCHL
          ACCESS MODE IS DYNAMIC
          FILE STATUS ANZEIGE.
DATA DIVISION.
FILE SECTION.
FD  DATEI1.
01  SATZ.
    03  ISAMSCHL              PIC 9(8).
    03  INHALT                PIC X(72).
WORKING-STORAGE SECTION.
01  ANZEIGE                   PIC XX.
01  CLOSE-INDIKATOR           PIC X VALUE "0".
    88  DATEI-ZU              VALUE "0".
    88  DATEI-OFFEN           VALUE "1".
PROCEDURE DIVISION.
DECLARATIVES.
DATEI-FEHLER SECTION.
```

```
    USE AFTER ERROR PROCEDURE ON DATEI1.
STATUS-ABFRAGE.
    EVALUATE ANZEIGE
    WHEN    "10"
      DISPLAY "Dateiende von DATEI1 erreicht" UPON T
    WHEN    "21"
      DISPLAY "Satz mit Schluessel" ISAMSCHL
          "SCHON VORHANDEN ODER NICHT IN"
          "AUFSTEIGENDER REIHENFOLGE" UPON T
    WHEN    "22"
      DISPLAY "Satz mit Schluessel" ISAMSCHL
          "SCHON VORHANDEN" UPON T
    WHEN    "23"
      DISPLAY "Satz mit Schluessel" ISAMSCHL
              "NICHT VORHANDEN" UPON T
    WHEN OTHER
      DISPLAY "Nicht behebbarer Fehler"
              "(" ANZEIGE ") FUER DATEIEINGABE" UPON T
      IF DATEI-OFFEN
      THEN
        CLOSE DATEI1
      END-IF
      DISPLAY "Programm abnormal beendet" UPON T
      STOP RUN
    END-EVALUATE.
END DECLARATIVES.
HAUPT SECTION.
AUF-ZU.
    ...
    STOP RUN.
```

Format 3 USE Anweisung mit GLOBAL-Angabe

```
USE GLOBAL AFTER STANDARD {EXCEPTION | ERROR} PROCEDURE ON {{dateiname-1}... |
INPUT | OUTPUT | I-O | EXTEND}
```

Die USE-Anweisung mit GLOBAL-Angabe kann in geschachtelten Programmen angegeben werden, um Prozedurvereinbarungen bzw. Kennsatzroutinen als global zu definieren.

Für die Vereinbarung von Benutzerkennsatzroutinen darf keine GLOBAL-Klausel angegeben wersen.

Syntaxregeln

1. Eine USE-Prozedur benötigt keine FD-Erklärung mit GLOBAL-Klausel.
2. Weitere Syntaxregeln zur USE-Anweisung siehe Formate vorher.

Allgemeine Regeln

1. Die GLOBAL-Angabe für eine USE-Prozedur hat die gleiche Wirkung wie die GLOBAL-Klausel in Daten- und Dateibeschreibungen.
2. Beansprucht eine Ein-/Ausgabeanweisung eine USE-Prozedur, so wird die gültige USE-Prozedur aus der Menge aller in dem geschachtelten Programm vereinbarten USE-Prozeduren nach folgenden Präzedenzregeln ausgewählt:
   a. Wenn im selben Programm eine passende (siehe Format 2) USE-Prozedur definiert ist, so wird diese ausgewählt.
   b. Wenn a) nicht zutrifft, gilt eine passende USE-Prozedur, die im *direkt* übergeordneten (nächstäußeren) Programm als global vereinbart ist.
   c. Wenn weder a) noch b) zutreffen, gilt eine passende USE-Prozedur, die im *indirekt* übergeordneten Programm als global vereinbart ist.

      Regel c) wird solange angewendet, bis alle indirekt übergeordneten Programme nach der gültigen globalen USE-Prozedur durchsucht sind.

   Falls keine passende USE-Prozedur gefunden wird, wird auch keine ausgeführt.
3. Als Erweiterung zum ANS85 erlaubt der hier beschriebene Compiler auch PERFORM-Anweisungen, die sich auf Programmteile außerhalb der Prozedurvereinbarungen (DECLARATIVES) beziehen. Diese Programmteile können auch CALL-, GO TO-, GOBACK- und EXIT PROGRAM-Anweisungen enthalten. Bei Anwendung einer globalen USE-Prozedur kann es zu einem rekursiven Aufruf desjenigen Programms kommen, das die USE-Prozedur aktiviert hat. Ein - stets unerlaubter - rekursiver Aufruf wird erst beim Ablauf des Programms festgestellt und führt zum Programmabbruch.
   Eine globale USE-Prozedur sollte deshalb keine EXIT PROGRAM-Anweisung ausführen.

Beispiel 8-87

für die Gültigkeitsbereiche globaler USE-Prozeduren

```
PROGRAM-ID. A-PROG.
...
FD  DATEI1  GLOBAL.
...
   PROGRAM-ID. B-PROG.
   ...
       USE GLOBAL ... ON DATEI1.          (1) globale USE-Prozedur
   ...
      PROGRAM-ID. C-PROG.
      ...
          USE GLOBAL ... ON INPUT.        (2) globale USE-Prozedur
          USE        ... ON OUTPUT.       (3) lokale USE-Prozedur
      ...
          OPEN OUTPUT DATEI1.             USE-Prozedur (3)
          OPEN EXTEND DATEI1.             USE-Prozedur (1)
      ...
         PROGRAM-ID. D-PROG.
         ...
             OPEN INPUT DATEI1.           USE-Prozedur (2)
         ...
            PROGRAM-ID. E-PROG.
            ...
            FD  DATEI1.
            ...
            OPEN INPUT DATEI1.            USE-Prozedur (2)
            OPEN OUTPUT DATEI1.           USE-Prozedur (1)
            ...
            END PROGRAM E-PROG.
         END PROGRAM D-PROG.
      END PROGRAM C-PROG.
   END PROGRAM B-PROG.

   PROGRAM-ID. F-PROG.
   ...
       OPEN I-O DATEI1.                   keine USE-Prozedur
   END PROGRAM F-PROG.
END PROGRAM A-PROG.
```

Format 4 für Ausnahmezustände

```
USE AFTER {EXCEPTION CONDITION | EC } {ausnahmesituationsname | EC-ALL} ...
```

Syntaxregeln

1. ausnahmesituationsname muss einer der in der Tabelle 45 aufgeführten Namen sein.
2. Innerhalb einer USE-Prozedur darf keine Bezugnahme auf Prozeduren außerhalb der Prozedurvereinbarungen (DECLARATIVES) enthalten sein, mit Ausnahme der PERFORM- und RESUME-Anweisung.
3. Auf Prozedurnamen, die einer USE-Anweisung untergeordnet sind, darf aus einer anderen Prozedur oder außerhalb der Prozedurvereinbarungen (DECLARATIVES) nur mit einer PERFORM-Anweisung Bezug genommen werden.
4. EC ist gleichbedeutend mit EXCEPTION CONDITION.

Allgemeine Regeln

1. Die Anweisungen innerhalb einer USE-Prozedur dürfen nicht dazu führen, dass eine noch aktive USE-Prozedur erneut aktiviert wird.
2. Wird ein Ausnahmezustand ausgelöst, so wird die erste USE-Prozedur, in der der zugehörige Ausnahmesituationsname angegeben ist, ausgewählt.
   Ist dieser Ausnahmesituationsname in keiner USE-Prozedur angegeben, so wird die erste USE-Prozedur, in der EC-ALL angegeben ist, ausgewählt.
   Ist auch EC-ALL in keiner USE-Prozedur angegeben, so wird der Programmlauf abhängig von der Kategorie der Ausnahmesituation (*fatal/non-fatal*) abgebrochen oder bei der nächsten ausführbaren Anweisung fortgesetzt.
3. Wird eine USE-Prozedur durch eine Ausnahemsituation aktiviert, so wird der Programmlauf abgebrochen, wenn die USE-Prozedur nicht durch die Ausführung einer expliziten Anweisung (z.B. GOBACK, RESUME) verlassen wurde.

8.10.48 WRITE-Anweisung

Funktion

Die WRITE-Anweisung veranlasst die Ausgabe eines Datensatzes in eine Ausgabedatei. Ferner kann mit Format 1 der Vorschub einer Druckerdatei gesteuert werden.

Format 1 für sequenzielle Dateiorganisation

```
WRITE datensatzname [FROM bezeichner-1]

    [{AFTER | BEFORE} ADVANCING{{bezeichner-2 | ganzzahl} [LINES | LINE] |
    {merkname | PAGE}}]

    [AT {END-OF-PAGE | EOP} unbedingte-anweisung-1]

    [NOT AT {END-OF-PAGE | EOP} unbedingte-anweisung-2]

    [END-WRITE]
```

Syntaxregeln

1. datensatzname und bezeichner-1 dürfen nicht den gleichen Speicherbereich belegen.
2. datensatzname muss einer Dateierklärung (FD) der DATA DIVISION des Programms zugeordnet sein und darf gekennzeichnet werden.
3. datensatzname darf nicht Bestandteil einer Sortierdatei sein.
4. Falls bezeichner-2 in der ADVANCING-Angabe verwendet wird, muss er der Name eines ganzzahligen numerischen Datenelementes sein.
5. Der Wert von ganzzahl oder des durch bezeichner-2 angesprochenen Datenfeldes muss größer oder gleich Null und darf nicht größer als 15 sein.

   Für Dateien mit dem ASSIGN-Eintrag PRINTER literal-1 darf ganzzahl einen Wert von > 15 und < 100 haben.
6. Falls END-OF-PAGE oder NOT END-OF-PAGE verwendet ist, muss innerhalb der Dateierklärung der entsprechenden Datei eine LINAGE-Klausel vorhanden sein (siehe „LINAGE-Klausel").
7. Die Angaben EOP und END-OF-PAGE sind gleichbedeutend.
8. Falls merkname verwendet wird, muss dafür innerhalb des SPECIAL-NAMES-Paragrafen eine Eintragung vorliegen.
9. Die Verwendung von merkname innerhalb der ADVANCING-Angabe ist nicht zulässig, falls für die in der WRITE-Anweisung angesprochene Datei eine LINAGE-Klausel angegeben wurde.
10. ADVANCING PAGE und END-OF-PAGE dürfen nicht zusammen in einer einzelnen WRITE-Anweisung angegeben werden.

Allgemeine Regeln

1. Der durch eine WRITE-Anweisung ausgegebene Datensatz steht im Satzbereich nicht mehr zur Verfügung, es sei denn, die zum Datensatz gehörende Datei wurde in einer SAME RECORD AREA-Klausel aufgeführt oder die Ausführung der WRITE-Anweisung war erfolglos. Der Datensatz steht auch allen anderen Dateien zur Verfügung, die in einer etwaigen SAME RECORD AREA-Klausel zusammen mit der angespochenen Datei aufgeführt wurden.
2. Die Ausführung einer WRITE-Anweisung mit der Angabe FROM ist gleichbedeutend mit:

   ```
   MOVE bezeichner-1 TO datensatzname
   der gleichen WRITE-Anweisung ohne FROM Angabe
   ```

   Die Übertragung findet entsprechend den Regeln für die MOVE-Anweisung ohne CORRESPONDING-Angabe statt.

Der Inhalt des Datensatzbereiches vor Ausführung der impliziten MOVE-Anweisung hat keinen Einfluss auf den Ablauf der WRITE-Anweisung.

Nach erfolgreicher Ausführung der WRITE-Anweisung steht die Information in dem durch bezeichner-1 angesprochenen Bereich nach wie vor zur Verfügung; dies gilt jedoch, wie in Allgemeine Regel 1 erläutert, nicht unbedingt für den Datensatzbereich.

3. Wird bei einer Mehrdatenträger-Ausgabedatei für Band oder Plattenspeicher im sequenziellen Zugriffsmodus das Ende des Datenträgers erkannt, werden bei Ausführung der WRITE-Anweisung die folgenden Schritte durchlaufen:
   a. Es werden die Standard-Datenträger-Anfangskennsatzprozeduren und die durch eine USE-Prozedur angegebenen Benutzer-Datenträger-Endekennsatzprozeduren durchlaufen. Dabei hängt die Reihenfolge dieser Prozeduren von den Angaben BEFORE/AFTER der eventuell vorhandenen USE-Prozedur ab.
   b. Es wird ein Datenträgerwechsel durchgeführt.
   c. Es werden die Standard-Datenträger-Anfangskennsatzprozeduren und die durch eine USE-Prozedur angegebenen Benutzer-Datenträger-Anfangskennsatzprozeduren durchlaufen. Dabei hängt die Reihenfolge dieser Prozeduren von den Angaben BEFORE/AFTER der eventuell vorhandenen USE-Prozedur ab.
4. Nach Ausführung einer WRITE-Anweisung wird der Wert des Datenfeldes einer für diese Datei vorhandenen FILE STATUS-Klausel aktualisiert (siehe auch „FILE STATUS-Klausel").
5. Ist eine Datei im Erweiterungsmodus geöffnet, bewirkt die WRITE-Anweisung, dass am Ende der Datei Datensätze hinzugefügt werden, als ob die Datei im Ausgabemodus geöffnet wäre. Der erste Datensatz, der nach der Ausführung einer OPEN EXTEND-Anweisung geschrieben wird, ist der Nachfolger des letzten in der Datei befindlichen Satzes.
6. Falls für die zu datensatzname gehörende Datei die RECORD-Klausel mit VARYING-Angabe angegeben wurde, darf die Anzahl der Zeichenpositionen des Datensatzes, der durch datensatzname bezeichnet wurde, nicht größer sein als ganzzahl-3 und nicht kleiner als ganzzahl-2 (siehe Format 2 im Abschnitt „RECORD-Klausel"). Wurde keine RECORD-Klausel mit VARYING-Angabe angegeben, darf die Anzahl der Zeichenpositionen nicht größer sein als der gemäß Datensatzerklärung größte Satz der zugehörigen Datei. Tritt einer dieser Fälle ein, ist die WRITE-Anweisung erfolglos. Der Schreibvorgang findet nicht statt. Der Inhalt des Datensatzbereiches wird nicht verändert. Der Ein-/Ausgabe-Zustand der mit datensatzname zusammenhängenden Datei wird auf einen Wert gesetzt, der die Ursache der Bedingung anzeigt (siehe „Ein-/Ausgabe-Zustand").
7. Sowohl die ADVANCING- als auch die END-OF-PAGE-Angabe bieten die Möglichkeit, jede einzelne Zeile innerhalb einer zu druckenden Seite zu positionieren.
8. Bei Verwendung der Angabe BEFORE/AFTER ADVANCING, wird der entsprechende Datensatz gedruckt, bevor/nachdem die zu druckende Seite entsprechend den nachfolgenden Regeln vorgeschoben wird.
9. Wenn während der Ausführung einer WRITE-Anweisung mit der NOT END-OF-PAGE-Angabe die Seitenende-Bedingung nicht auftritt, wird bei erfolgreicher Ausführung der WRITE-Anweisung mit unbedingte-anweisung-2 fortgesetzt, nachdem der Satz geschrieben ist und nachdem der Ein-/Ausgabe-Zustand der Datei, aus der der Satz stammt, aktualisiert worden ist. Bei erfolgloser Ausführung der WRITE-Anweisung wird der Ein-/Ausgabe-Zustand der Datei aktualisiert, dann die für diese Datei angegebene USE-Prozedur ausgeführt und schließlich mit unbedingte-anweisung-2 fortgesetzt.
10. Der Vorschub der Druckseite erfolgt bei Vorhandensein der ADVANCING-Angabe nach folgenden Regeln:
    a. Bei Vorhandensein von bezeichner-2, ganzzahl oder merkname erfolgt der Vorschub entsprechend den in Tabelle 34 angegebenen Regeln.
    b. Ist PAGE angegeben, wird der entsprechende Datensatz gedruckt, bevor oder nachdem auf den Beginn der ersten Zeile einer neuen Seite positioniert wird. Der Beginn einer neuen Seite ist hierbei

durch die physischen Eigenschaften des Druckes (Vorschub nach Kanal 1) oder, falls eine LINAGE-Klausel vorhanden ist, durch den Beginn der nächsten logischen Seite definiert (siehe auch Abschnitt „LINAGE-Klausel").

11. Fehlt die ADVANCING-Angabe, wird für Dateien mit ASSIGN-Angabe PRINTER oder PRINTER literal-1 ein WRITE...AFTER ADVANCING 1 LINE durchgeführt (siehe auch Tabelle 35).
12. Wird bei Ausführung einer WRITE-Anweisung mit der END-OF-PAGE-Angabe das Ende einer logischen Seite erreicht, wird der Programmablauf mit unbedingte-anweisung-1 fortgesetzt. Das logische Ende einer Seite ist durch die LINAGE-Klausel in der Dateierklärung der Datei definiert.
13. Eine END-OF-PAGE-Bedingung tritt auf, wenn während der Ausführung einer WRITE-Anweisung mit der END-OF-PAGE-Angabe innerhalb des Seitenfußes gedruckt oder vorgeschoben wird. Dies ist dann der Fall, wenn während der Ausführung einer solchen WRITE-Anweisung der Wert des LINAGE-COUNTER-Sonderregisters gleich dem oder größer als der Wert von ganzzahl-2 oder des durch datenname-2 angesprochenen Datenfeldes einer LINAGE-Klausel wird. In diesem Fall wird die WRITE-Anweisung ausgeführt und danach der Programmablauf bei der in der END-OF-PAGE-Angabe genannten unbedingten Anweisung fortgesetzt.
14. Eine automatische Seitenüberlaufbedingung tritt immer dann auf, wenn eine gegebene WRITE-Anweisung (mit oder ohne END-OF-PAGE-Angabe) nicht mehr innerhalb der aktuellen Seitengrenzen ausgeführt werden kann.

    Dies ist dann der Fall, wenn während der Ausführung einer solchen WRITE-Anweisung der Wert des LINAGE-COUNTER-Sonderregisters größer als der Wert von ganzzahl-1 oder des durch datenname-1 angesprochenen Datenfeldes einer LINAGE-Klausel wird. In diesem Fall wird der entsprechende Datensatz gedruckt, bevor oder nachdem (abhängig von der Angabe BEFORE/AFTER) auf die erste Zeile der nächsten logischen Seite positioniert worden ist. Diese ist durch die Angabe der LINAGE-Klausel festgelegt. Die in der END-OF-PAGE-Angabe genannte unbedingte Anweisung wird durchlaufen, nachdem der Datensatz geschrieben und das Gerät neu positioniert
    wurde.

    Falls die Angaben ganzzahl-2 oder datenname-2 der LINAGE-Klausel nicht vorhanden sind, tritt die END-OF-PAGE-Bedingung dann auf, wenn während der Ausführung der Ausführung einer WRITE-Anweisung mit EOP-Angabe der Wert des LINAGE-COUNTER-Sonderregisters gleich dem oder größer als der Wert von ganzzahl-1 oder des durch datenname-1 angesprochenen Datenfeldes einer LINAGE-Klausel wird. Wenn ganzzahl-2 oder datenname-2 in der LINAGE-Klausel angegeben sind, die Ausführung einer WRITE-Anweisung jedoch dazu führen würde, dass der Wert des LINAGE-COUNTER sowohl

    a. gleich dem oder größer als der Wert von datenname-2 bzw. ganzzahl-2, als auch
    b. gleich dem oder größer als der Wert von datenname-1 bzw. ganzzahl-1

    ist, läuft die Verarbeitung so ab, als ob ganzzahl-2 oder datenname-2 nicht vorhanden wären.
15. Wird die ADVANCING-Angabe zusammen mit der LINAGE-Klausel für ein Gerät verwendet, das keinen physischen Drucker darstellt, werden sowohl Leerzeilen am oberen und unteren Seitenrand als auch innerhalb der Seite durch Datensätze mit Inhalt Leerzeichen auf dem entsprechenden Speichermedium dargestellt.
16. Wird bei Ausführung einer WRITE-Anweisung versucht, außerhalb der physischen Grenzen einer Datei zu schreiben, tritt eine Ausnahmebedingung auf und die folgenden Schritte laufen ab:
    a. Der Wert des FILE STATUS-Datenfeldes der entsprechenden Datei wird gesetzt, um eine Verletzung der Dateigrenzen anzuzeigen (siehe „FILE STATUS-Klausel").
    b. Ist für diese Datei explizit oder implizit eine USE ERROR/EXCEPTION-Prozedur vorhanden, wird sie durchlaufen.
    c. Ist für diese Datei weder explizit noch implizit eine USE-Prozedur vorhanden, führt das zum Programmabbruch.
17. Bei Ausführung einer WRITE-Anweisung muss die Datei mit den Angaben OUTPUT, oder EXTEND in der OPEN-Anweisung eröffnet worden sein.

18. Tabelle 36 enthält genaue Angaben über die Verwendung des ersten Zeichens des Datensatzes in Abhängigkeit vom symbolischen Gerätenamen der ASSIGN-Klausel und den Angaben der WRITE-Anweisung.
19. Bei einer WRITE-Anweisung mit AFTER-ADVANCING-Angabe wird auf die angeforderte Zeile positioniert und nach dem Drucken um eine weitere Zeile vorgeschoben. Bei ASSIGN-Angabe PRINTER literal-1 wird eine WRITE AFTER-Angabe wie folgt ausgeführt:
    a. Schreiben eines Leersatzes mit entsprechender Vorschubangabe (WRITE BEFORE).
    b. Schreiben des Datensatzes.

    Vorschubunterdrückung ist für jedes Gerät nur bei Angabe von WRITE BEFORE möglich.

Beispiel 8-88

Die Datei LADEDAT wird eingelesen und in die Druckdatei ADATEI geschrieben. Der jeweils aktuelle Datensatz wird vor dem Einfügen von 5 Leerzeilen geschrieben.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WP1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ADATEI ASSIGN TO PRINTER "AUSGABE".
    SELECT LADEDAT ASSIGN TO "EINGABE".
DATA DIVISION.
FILE SECTION.
FD   ADATEI
     LINAGE IS 24 LINES
     LINES AT TOP 24
     LINES AT BOTTOM 24.
01  KUNDENSATZ  PIC X(95).
FD   LADEDAT
     LABEL RECORD IS STANDARD.
01  KUNSATZ.
    05 KNA PIC X(30).
    05 STR PIC X(20).
    05 ORT PIC X(20).
    05 PLZ PIC X(5).
    05 ART PIC X(20).
WORKING-STORAGE SECTION.
01  DATEI-SCHALTER PIC X VALUE LOW-VALUE.
   88  EOF VALUE HIGH-VALUE.
PROCEDURE DIVISION.
HAUPT SECTION.
OEFFNEN.
    OPEN OUTPUT ADATEI.
    OPEN INPUT LADEDAT.
EINLESEN.
    PERFORM UNTIL EOF
      READ LADEDAT INTO KUNDENSATZ
      AT END
        CLOSE ADATEI LADEDAT
        SET EOF TO TRUE
      NOT AT END
        WRITE KUNDENSATZ BEFORE ADVANCING 5
      END-READ
```

```
        END-PERFORM
        STOP RUN.
```

Vorschubangabe	Inhalt/Bedeutung	Ergebnis	
bezeichner-2	ganzzahlig numerisch mit einem Wert von01 bis 15	n-zeiliger Vorschub	
bezeichner-2	alphanumerisches Datenfeld der Länge 1(PIC X). Folgende Werte sind zulässig:		
	Leerzeichen	einzeiliger Vorschub	
	0	zweizeiliger Vorschub	
	-	dreizeiliger Vorschub	
	+	Vorschubunterdrückung	
	1-8	Vorschub Kanal 1-8	
	A	Vorschub Kanal 10	
	B	Vorschub Kanal 11	
ganzzahl	numerisches Literal 01 bis 15	n-zeiliger Vorschub	
merkname	Ein Name, mit dem innerhalb des SPECIAL-NAMES-Paragrafen ein Herstellername verknüpft worden ist.	herstellername	Ergebnis
		C01 bis C08 C10 oder C11	Vorschub Kanal 1-8 Vorschub Kanal 10 bzw. 11

Tabelle 34: Bedeutung und Ergebnis der Druckervorschubzeichen in einer WRITE-Anweisung mit ADVANCING

symbolischer Gerätename	WRITE-Anweisung ohne ADVANCING-Angabe	WRITE-Anweisung mit ADVANCING-Angabe	Kommentar
PRINTER literal	Standardvorschub bei fehlender ADVANCING-Angabe entspricht einer Angabe AFTER 1 LINE; das erste Zeichen des Datensatzes steht für Benutzerdaten zur Verfügung.	Das erste Zeichen des Datensatzes steht für Benutzerdaten zur Verfügung.	Der Platz für das Vorschubzeichen wird vom Compiler reserviert und ist dem Benutzer nicht zugänglich. Für diesen Druckertyp ist die Angabe der LINAGE-Klausel in der Dateierklärung möglich. Es sind sowohl WRITE-Anweisungen mit als auch ohne ADVANCING-Angabe für eine Datei zulässig.
PRINTER PRINTER01 - PRINTER99	wie oben	wie oben	Der Platz für das Vorschubzeichen wird vom Compiler reserviert und ist dem Benutzer nicht zugänglich. Die LINAGE-Klausel ist für diese Datei nicht erlaubt. Die Verwendung einer WRITE-Anweisung mit

			und ohne ADVANCING-Angabe für ein und dieselbe Datei ist nicht zulässig. Sollte dennoch dieser Fall eintreten, wird für die Sätze ohne ADVANCING-Angabe ein WRITE AFTER ADVANCING 1 LINE implizit durchgeführt.
literal	Der Vorschub wird durch das erste Zeichen in jedem logischen Datensatz kontrolliert; der Benutzer muss daher vor der Ausführung jeder solchen WRITE-Anweisung das geeignete Steuerzeichen dort zur Verfügung stellen.	Der Benutzer muss das erste Zeichen eines logischen Datensatzes reservieren; an diese Stelle wird vom Laufzeitsystem zum Programmablauf das Vorschubzeichen eingetragen. Eventuell enthaltene Benutzerdaten werden überschrieben.	Es dürfen WRITE-Anweisungen mit und ohne ADVANCING-Angabe gemischt verwendet werden. In beiden Fällen beginnt jedoch die Benutzerinformation des Druckersatzes erst ab dem zweiten Zeichen des Datensatzes.

Tabelle 35: Verwendung von symbolischen Gerätenamen für Drucker in Verbindung mit der WRITE-Anweisung

Format 2 für relative und indizierte Dateiorganisation

```
WRITE datensatzname [FROM bezeichner]
    [INVALID KEY unbedingte-anweisung-1]
    [NOT INVALID KEY unbedingte-anweisung-2]
    [END-WRITE]
```

Syntaxregeln

1. datensatzname und bezeichner dürfen nicht den gleichen Speicherbereich belegen.
2. datensatzname muss einer Dateierklärung (FD) der DATA DIVISION des Programms zugeordnet sein und darf gekennzeichnet werden.
3. Die INVALID KEY-Angabe muss für eine Datei vorhanden sein, falls nicht eine entsprechende USE-Prozedur vereinbart wurde.

Allgemeine Regeln

Für **relative und indizierte Dateiorganisation** gilt:

1. Die Datei, deren Datensatz in der WRITE-Anweisung angesprochen wird, muss im Eröffnungsmodus Ausgabe (OUTPUT), Aktualisieren (I-O) oder Erweitern (EXTEND) eröffnet worden sein.
2. Der durch eine WRITE-Anweisung ausgegebene Datensatz steht im Satzbereich nicht mehr zur Verfügung, es sei denn, die zum Datensatz gehörende Datei wurde in einer SAME RECORD AREA-Klausel aufgeführt, oder die Ausführung der WRITE-Anweisung wurde wegen des Auftretens einer Schlüsselfehler-Bedingung erfolglos abgebrochen. Der Datensatz steht auch den Dateien zur Verfügung, die in einer SAME RECORD AREA-Klausel zusammen mit der angesprochenen Datei aufgeführt wurden.
3. Die Ausführung einer WRITE-Anweisung mit der Angabe FROM ist gleichbedeutend mit:

   ```
   MOVE bezeichner-1 TO datensatzname
   der gleichen WRITE-Anweisung ohne FROM Angabe
   ```

   Die Übertragung findet entsprechend den Regeln für die MOVE-Anweisung ohne CORRESPONDING-Angabe statt.

Der Inhalt des Datensatzbereiches vor Ausführung der impliziten MOVE-Anweisung hat keinen Einfluss auf den Ablauf der WRITE-Anweisung.

4. Die Ausführung einer WRITE-Anweisung bewirkt, dass der Inhalt des Datenfeldes aktualisiert wird, das in der FILE STATUS-Klausel der zugehörigen Dateierklärung angegeben wurde (siehe auch „FILE STATUS-Klausel").
5. Wenn während der Ausführung einer WRITE-Anweisung mit der NOT INVALID KEY-Angabe die Schlüsselfehler-Bedingung nicht auftritt, wird mit unbedingte-anweisung-2 wie folgt fortgesetzt:
   a. Bei erfolgreicher Ausführung der WRITE-Anweisung: nachdem der Satz geschrieben ist und nachdem der Ein-/Ausgabe-Zustand der Datei, aus der der Satz stammt, aktualisiert worden ist.
   b. Bei erfolgloser Ausführung der WRITE-Anweisung: nachdem der Ein-/Ausgabe Zustand der Datei aktualisiert wurde und nach Ausführung der USE-Prozedur, die für die Datei, aus der der Satz stammt, angegeben wurde.
6. Falls eine Schlüsselfehler-Bedingung auftrat, ist die WRITE-Anweisung ohne Erfolg ausgeführt worden; der Inhalt des Satzbereiches steht weiterhin zur Verfügung, und ein eventuell vorhandenes FILE STATUS-Datenfeld dieser Datei wird auf einen Wert gesetzt, der den Grund der Schlüsselfehler-Bedingung anzeigt. Das Programm wird entsprechend den Regeln der Schlüsselfehler-Bedingung fortgesetzt.
7. Die Anzahl der Zeichenpositionen in dem durch datensatzname bezeichneten Datensatz darf nicht größer als die größte und nicht kleiner als die kleinste Anzahl von Zeichenpositionen sein, die laut RECORD IS VARYING-Klausel für den Datensatz erlaubt ist. Andernfalls ist die Ausführung der WRITE-Anweisung erfolglos, die Schreiboperation findet nicht statt, der Inhalt des Satzbereichs bleibt unbeeinflusst und der Ein-/Ausgabe-Zustand für die entsprechende Datei wird auf den Wert gesetzt, der das Überschreiten der Satzlängengrenzen anzeigt (siehe „Ein-/Ausgabe-Zustand").

   Nur für **relative Dateiorganisation** gilt:
8. Bei Ausführung einer WRITE-Anweisung wird zur Bestimmung der Position des auszugebenden Datensatzes innerhalb der Datei der Inhalt des RELATIVE KEY-Datenfeldes verwendet. Vor Ausführung der WRITE-Anweisung muss der Inhalt des zugehörenden Schlüsselfeldes entsprechend gesetzt sein (siehe RELATIVE KEY-Angabe im Abschnitt „ACCESSMODE-Klausel").
9. Befindet sich eine Datei im Ausgabemodus (OPEN-Anweisung mit OUTPUT-Angabe) ist Folgendes zu beachten:
   a. Im sequenziellen Zugriffsmodus veranlasst die WRITE-Anweisung die Ausgabe eines Datensatzes zum Erstellen einer neuen Datei. Der erste Datensatz bekommt die relative Satznummer 1 (eins) während die danach folgenden Datensätze die Nummern 2, 3, 4 ... erhalten. Wurde RELATIVE KEY angegeben, wird während der Ausführung der WRITE-Anweisung vom Ein-/Ausgabe-System die relative Satznummer im Datenfeld, dessen Name in der RELATIVE KEY-Angabe auftritt, eingetragen.
   b. Im wahlfreien oder dynamischen Zugriffsmodus (die im Falle OUTPUT gleichbe deutend sind) muss der Inhalt des Datenfeldes der hier obligatorischen RELATIVE KEY-Angabe vom Benutzer auf den Wert der relativen Satznummer gesetzt werden, die der im Satzbereich befindliche Datensatz erhalten soll. Der entsprechende Datensatz wird dann als n-ter Datensatz der Datei ausgegeben, wobei n der Wert der relativen Satznummer ist.
10. Befindet sich eine Datei im Erweiterungsmodus (OPEN-Anweisung mit EXTEND-Angabe), veranlasst die WRITE-Anweisung das Hinzufügen eines Datensatzes an die Datei. Der so übergebene erste Datensatz erhält dabei eine relative Satznummer, die um 1 höher ist als die höchste relative Satznummer der bereits existierenden Sätze in der Datei. Jeder nachfolgende Datensatz hat dann eine um 1 (gegenüber dem letzten Satz) erhöhte Satznummer. Wenn die RELATIVE KEY-Angabe vorliegt, wird bei jedem WRITE die relative Satznummer der MOVE-Anweisung entsprechend in das Satzschlüsselfeld übertragen.
11. Befindet sich eine Datei im Aktualisierungsmodus (OPEN-Anweisung mit I-O-Angabe) und im wahlfreien oder dynamischen Zugriffsmodus (was in diesem Falle gleichbedeutend ist), werden durch die WRITE-Anweisung Datensätze in eine bereits existierende Datei eingefügt. Der Inhalt des Datenfeldes der hier obligatorischen RELATIVE KEY-Angabe muss vom Benutzer auf den Wert der relativen Satznummer

gesetzt werden, die der im Satzbereich befindliche Datensatz erhalten soll. Mit Ausführung der WRITE-Anweisung wird der Satzinhalt mit entsprechender Satznummer dem Ein-/AusgabeSystem übergeben.

12. Die Schlüsselfehler-Bedingung wird durch die in Tabelle 36 angeführten Gründe verursacht.

ACCESS MODE-Klausel	**OPEN-Angabe und Ergebnis**	**Grund der Schlüsselfehler-Bedingung**
SEQUENTIAL	OUTPUT oder EXTEND Es wird ein Datensatz einer existierenden oder neu zu erstellenden Datei hinzugefügt.	Es steht kein Platz für den Datensatz in der Datei zur Verfügung.
RANDOM oder DYNAMIC	OUTPUT oder I-O Es wird ein Datensatz einer bestehenden Datei hinzugefügt.	Der Inhalt des RELATIVE KEY-Datenfeldes bezeichnet einen Datensatz, der bereits in der Datei existiert, oder es steht kein Platz für den Datensatz in der Datei zur Verfügung.

Tabelle 36: WRITE-Anweisung, Gründe für das Auftreten einer Schlüsselfehler-Bedingung

Nur für **indizierte Dateiorganisation** gilt:

13. Bei Ausführung einer WRITE-Anweisung wird zur Bestimmung der Position des auszugebenden Datensatzes innerhalb der Datei der Inhalt des RECORD KEY-Datenfeldes verwendet. Vor Ausführung der WRITE-Anweisung muss der Inhalt des zugehörenden Schlüsselfeldes entsprechend gesetzt sein (siehe „RECORD KEY-Klausel").
14. Befindet sich eine Datei im Ausgabemodus (OPEN-Anweisung mit OUTPUT-Angabe) ist Folgendes zu beachten:
    a. Im sequenziellen Zugriffsmodus veranlasst die WRITE-Anweisung die Ausgabe eines Datensatzes zum Erstellen einer neuen Datei. Die Datensätze müssen dabei in aufsteigender Reihenfolge des RECORD KEY übergeben werden. Vor Ausführung der WRITE-Anweisung muss der Inhalt des RECORD KEY-Datenfeldes auf den gewünschten Wert gesetzt werden.
    b. Im wahlfreien oder dynamischen Zugriffsmodus (die im Falle OUTPUT gleichbedeutend sind) dürfen die Datensätze in einer im Programm festgelegten Reihenfolge an das Ein-/Ausgabe-System übergeben werden.
15. Die Schlüsselfehler-Bedingung wird durch die in Tabelle 37 angeführten Gründe verursacht.

ACCESS MODEKlausel	**OPEN-Angabe und Ergebnis**	**Grund der Schlüsselfehler-Bedingung**
SEQUENTIAL	OUTPUT / EXTEND Einer neu zu erstellenden Datei wird ein Datensatz hinzugefügt („Lademodus “).	Der Primärschlüsselwert ist nicht größer als der des vorhergehenden Datensatzes (die Primärschlüssel müssen in aufsteigender Reihenfolge alphanumerisch sortiert sein). In der Datei steht kein Platz mehr zur Aufnahme des Datensatzes zur Verfügung.
RANDOM oder DYNAMIC	OUTPUT / I-O / EXTEND Einer bestehenden Datei wird ein Datensatz hinzugefügt.	Der Datensatz hat den gleichen Primärschlüsselwert wie ein bereits in der Datei vorhandener Datensatz.

		In der Datei steht kein Platz mehr zur Aufnahme des Datensatzes zur Verfügung. Ein Alternativschlüsselwert, für den keine Duplikate zulässig sind, ist in einem der Datensätze der Datei schon vorhanden.

Tabelle 37: WRITE-Anweisung, Gründe für das Auftreten einer Schlüsselfehler-Bedingung

16. Wurde die Datei im Erweiterungsmodus eröffnet, muss der erste an das DVS übergebene Satz einen Primärschlüssel haben, dessen Wert höher ist als der höchste in der Datei vorhandene Primärschlüsselwert.
17. Ist für die Datei ALTERNATE RECORD KEY WITH DUPLICATES angegeben, braucht der Alternativschlüsselwert nicht eindeutig zu sein.

9 Interne Standard-Funktionen

Eine interne Standard-Funktion ermöglicht es, ein temporäres Datenfeld zu referenzieren, das vom COBOL-Programm zur Verfügung gestellt wird und dessen Wert automatisch berechnet wird, wenn die Funktion angesprochen wird.

9.1 Allgemeines

Funktionsname

Jede Standard-Funktion hat einen Namen, mit dem der Programmierer sie ansprechen kann. Ein Funktionsname ist ein Schlüsselwort aus einer besonderen Liste von COBOL-Wörtern und ist notwendiger Bestandteil des Funktionsbezeichners. Außerhalb eines Funktionsbezeichners kann ein solches Schlüsselwort auch als benutzerdefinierter Name verwendet werden.

Returnwert einer Funktion

Jede Funktion, die erfolgreich durchlaufen wurde, liefert ein Funktionsresultat, den *Returnwert*. Um den Returnwert zu bestimmen, verarbeitet die Funktion die Datenwerte, die von den im Funktionsbezeichner genannten Argumenten geliefert werden.
Das Funktionsresultat ist definiert

1. durch die Länge des Returnwertes bei alphanumerischen und nationalen Funktionen,
2. durch das Vorzeichen des Returnwertes bzw. die Ganzzahligkeit bei numerischen und ganzzahligen Funktionen,
3. im Übrigen durch den Returnwert selbst.

Für die Argumente gelten bestimmte Vorschriften: Datentyp, Anzahl, Länge und Wertebereich der Argumente sind durch die Definition der Funktion festgelegt. Nur wenn diese Vorschriften eingehalten werden, liefert die Funktion einen definierten Returnwert.

Fehler-Returnwert

Ist eine Funktion mit einem ungültigen Argument versehen, ist das Resultat undefiniert. Mit einer Compileroption kann die Überprüfung der Argumentwerte veranlasst und der Funktion der *Fehler-Returnwert* zugewiesen werden, der anzeigt, dass die Funktion nicht erfolgreich abgelaufen ist.
Mit einer weiteren Compileroption kann bewirkt werden, dass der Fehlerfall zur Ablaufzeit gemeldet wird (Fehlermeldungen COB9123 - COB9128). Näheres hierzu ist im Handbuch „COBOL2000 Benutzerhandbuch“ [1] dargestellt.

Datumskonversion

In den Funktionen zur Datumskonversion wird der Gregorianische Kalender verwendet. Das Startdatum Montag, 1. Januar 1601 wurde ausgewählt, um eine einfache Beziehung zwischen dem Standarddatum und DAY-OF-WEEK herzustellen; d.h. das ganzzahlige Datum 1 war ein Montag, DAY-OF-WEEK 1.

Argumente

Argumente bezeichnen die für den Ablauf einer Funktion verwendeten Werte. Argumente werden im Funktionsbezeichner (siehe "Funktionsbezeichner") angegeben. Sie können als Bezeichner, als arithmetische Ausdrücke oder als Literale angegeben werden. Die Formatbeschreibung einer Funktion enthält die Anzahl der erforderlichen Argumente, die null, eins oder mehr betragen kann. Für manche Funktionen kann die Anzahl der angebbaren Argumente variabel sein.

Argumente gehören einer bestimmten Datenklasse oder einer Teilmenge einer Datenklasse an. Es gibt folgende Argumenttypen:

a. Numerisch. Es muss ein arithmetischer Ausdruck angegeben werden. Der Wert des arithmetischen Ausdrucks wird, einschließlich des Vorzeichens, zur Bestimmung des Funktionswertes herangezogen.
b. Alphabetisch. Ein Datenfeld der Klasse alphabetisch oder ein alphanumerisches Literal, das nur aus alphabetischen Zeichen besteht, muss angegeben sein. Die Länge des Arguments kann für die Bestimmung des Funktionswertes verwendet werden.

c. Alphanumerisch. Ein Datenfeld der Klasse alphabetisch oder alphanumerisch oder ein alphanumerisches Literal muss angegeben sein. Die Länge des Arguments kann für die Bestimmung des Funktionswertes verwendet werden.

d. National. Ein nationales Datenfeld oder ein nationales Literal muss angegeben werden. Die Länge des Arguments kann für die Bestimmung des Funktionswertes verwendet werden.

e. Ganzzahlig. Es muss ein arithmetischer Ausdruck, der immer in einem ganzzahligen Wert resultiert, angegeben sein. Der Wert des arithmetischen Ausdrucks wird einschließlich des Vorzeichens zur Bestimmung des Funktionswertes verwendet.

f. Von der Klasse objekt.

g. Von der Klasse zeiger.

h. Typdeklaration.

i. Index. Ein Datenfeld mit USAGE INDEX muss angegeben werden.

Wenn das Format einer Funktion erlaubt, dass ein Argument beliebig oft wiederholt wird, kann eine Tabelle referenziert werden. Die Referenz erfolgt durch Angabe des Datennamens und beliebiger Kennzeichner für die Tabelle, unmittelbar gefolgt von einer Subskribierung, in der ein oder mehrere Subskripte in dem Wort ALL bestehen.

Wenn ALL als Subskript angegeben ist, hat dies die gleiche Wirkung, als ob jedes Tabellenelement zu dieser Subskriptposition angegeben wäre.

Beispiel:

ALL-Subskripte in einer dreidimensionalen Tabelle mit zehn Elementen in jeder Dimension:

```
FUNCTION MAX (TAB(ALL 2 ALL))
```

ist identisch mit

```
FUNCTION MAX (TAB(1 2 1) TAB(1 2 2) ... TAB(1 2 10)
              TAB(2 2 1) TAB(2 2 2) ... TAB(2 2 10)
              ...
              TAB(10 2 1) TAB(10 2 2) ... TAB(10 2 10))
```

Wenn das ALL-Subskript mit einer OCCURS DEPENDING ON-Klausel verknüpft ist, wird der Wertebereich vom Objekt der OCCURS DEPENDING ON-Klausel bestimmt. Dieses Objekt muss bei der Auswertung der Argumente größer als 0 sein. Wenn ein mit ALL subskribiertes Argument teilfeldselektiert ist, so bezieht sich der Teilfeldselektor auf jedes implizit referenzierte Tabellenelement.

Funktionstypen

Funktionen sind Datenelemente. Sie liefern alphanumerische, nationale, numerische oder index Werte und können keine Empfangsoperanden sein. Es gibt folgende Typen von Funktionen:

a. Alphanumerische Funktionen. Sie gehören zur Klasse und Kategorie alphanumerisch. Alphanumerische Funktionen haben die implizite USAGE DISPLAY.

b. Nationale Funktionen. Sie gehören zur Klasse und Kategorie national. Nationale Funktionen haben die implizite USAGE NATIONAL.

c. Numerische Funktionen. Sie gehören zur Klasse und Kategorie numerisch.
Eine numerische Funktion wird stets als vorzeichenbehaftet behandelt.
Eine numerische Funktion kann nur in einem arithmetischen Ausdruck verwendet werden.
Eine numerische Funktion darf nicht referenziert werden, wo ein ganzzahliger Operand verlangt ist, selbst wenn die Auswertung der Funktion einen ganzzahligen Wert ergibt.

d. Ganzzahl-Funktionen. Sie gehören zur Klasse und Kategorie numerisch.
   Eine Ganzzahl-Funktion wird stets als vorzeichenbehaftet behandelt.
   Eine Ganzzahl-Funktion kann nur in einem arithmetischen Ausdruck verwendet werden.
e. Index-Funktionen. Sie gehören zur Klasse und Kategorie Index.

9.2 Übersicht über die Standard-Funktionen

Die folgende Tabelle fasst die verfügbaren Funktionen zusammen.

Die Spalte „Argumente“ definiert Typ und Argumentanzahl wie folgt:

A	bedeutet	alphabetisch
I	bedeutet	ganzzahlig
Ind	bedeutet	index
Nat	bedeutet	national
Num	bedeutet	numerisch
O	bedeutet	Objektreferenz
P	bedeutet	Zeiger
T	bedeutet	Typdeklaration
X	bedeutet	alphanumerisch[1]

[1] X in der Argumentenspalte beinhaltet stark typisierte Datengruppen. Bestimmt der Typ des Arguments den Typ der Funktion, dann ist die Funktion alphanumerisch, sobald ein einziges ihrer Argumente stark typisiert ist. Dies gilt auch wenn alle Argumente der Funktion vom gleichen Typ sind. Bei den Funktionen ADDR, MAX, MIN, ORD-MAX und ORD-MIN sind stark typisierte Datengruppen als Argumente ausgeschlossen.

Die Spalte „Typ“ definiert den Typ der Funktion wie folgt:

I	bedeutet	ganzzahlig
Ind	bedeutet	index
Nat	bedeutet	national
Num	bedeutet	numerisch
X	bedeutet	alphanumerisch

Funktionsname	Argumente	Typ	Returnwert
ACOS	Num1	Num	Arcuscosinus von Num1
ADDR	A1 oder Ind1 Nat1 oder Num1 oder X1	I	Adresse des Arguments
ANNUITY	Num1, I2	Num	jährliche Zahlung im Zeitraum I2 bei Zinssatz Num1 bezogen auf das Anfangskapital 1
ASIN	Num1	Num	Arcussinus von Num1
ATAN	Num1	Num	Arcustangens von Num1
BYTE-LENGTH	A1 oder Ind1 oder N1 oder	I	Länge des Arguments in Bytes

	Nat1 oder O1 oder P1 oder T1 oder X1		
CHAR	I1	X	Zeichen auf Position I1 der alphanumerischen Sortierfolge
CHAR-NATIONAL	I1	Nat	Zeichen auf Position I1 der nationalen Sortierfolge
COS	Num1	Num	Cosinus von Num1
CURRENT-DATE	keine	X	aktuelles Datum und aktuelle Uhrzeit
DATE-OF-INTEGER	I1		Umwandlung der angegebenen Anzahl Tage in das Standard-Datum der Form JJJJMMTT
DATE-TO-YYYYDDD	I1, I2	I	Umwandlung von I1 aus einer Standard-Datumsangabe mit zweistelliger Jahreszahl (JJDDD) in die Form JJJJDDD, abhängig vom Wert von I2
DAY-OF-INTEGER	I1	I	Umwandlung der angegebenen Anzahl Tage in das julianische Datum der Form JJJJTTT
DAY-TO-YYYYDDD	I1, I2	I	Umwandlung von I1 aus einer Datumsangabe nach julianischer Form (JJDDD) in die Form JJJJDDD, abhängig vom Wert von I2
DISPLAY-OF	Nat1, X2	X	alphanumerische Darstellung der nationalen Zeichenfolge Nat1
EXCEPTION-STATUS	keine	X	Name des letzten Ausnahmezustands
FACTORIAL	I1	I	Fakultät von I1
INTEGER	Num1	I	Größten Ganzzahl, die nicht größer ist als Num1
INTEGER-OF-DATE	I1	I	Umwandlung einer Standard-Datumsangabe (JJJJMMDD)
INTEGER-OF-DAY	I1	I	Umwandlung einer Datumsangabe nach julianischer Form (JJJJTTT)
INTEGER-PART	Num1	I	ganzzahliger Teil von Num1
LENGTH	A1 oder Ind1 oder Nat1 oder Num1 oder X1 oder T1	I	Länge des Arguments in Zeichen
LOG	Num1	Num	natürlicher Logarithmus von Num1
LOG10	Num1	Num	Logarithmus von Num1 zur Basis 10
LOWER-CASE	A1 oder	Nat X	alle Buchstaben im Argument werden in Kleinbuchstaben umgesetzt

	Nat1 oder X1		
MAX	A1... oder I1... oder Ind1 oder Nat1... oder Num1... oder X1...	X I Ind Nat Num X	höchster Argumentwert
MEAN	Num1...	Num	arithmetisches Mittel der Argumentwerte
MEDIAN	Num1...	Num	mittlerer Argumentwert
MIDRANGE	Num1...	Num	Mittel aus dem niedrigsten und höchsten Argument
MIN	A1... oder I1... oder Ind1 oder Nat1... oder Num1... oder X1...	X I Ind Nat Num X	niedrigster Argumentwert
MOD	I1, I2	I	I1 modulo I2
NATIONAL-OF	X1, Nat2	Nat	nationale Darstellung der alphanumerischen Zeichenfolge X1
NUMVAL	Nat1 oder X1	Num	Numerischer Wert einer Zeichenkette
NUMVAL-C	Nat1 oder Nat2 oder X1, X2	Num	Numerischer Wert einer Zeichenkette mit wahlweisen Kommas und Währungszeichen
ORD	A1 oder Nat1 oder X1	I	Ordnungsposition von A1 / X1 in der Sortierfolge
ORD-MAX	A1... oder Ind1 oder Nat1... oder Num1... oderX1...	I	Ordnungsposition des höchsten Argumentwertes
ORD-MIN	A1... oder Ind1 oder Nat1... oder Num1... oder X1...	I	Ordnungsposition des niedrigsten Argumentwer- tes
PRESENT-VALUE	Num1 Num2...	Num	Tilgungsanteil einer Summe von Ratenzahlungen, Num2, bei einem Zinssatz Num1
RANDOM	I1	Num	Zufallszahl
RANGE	I1... oder Num1...	I Num	Differenz zwischen höchstem und niedrigstem Argumentwert
REM		Num	Rest der Division Num1 durch Num2

	Num1, Num2		
REVERSE	A1 oder Nat1 oder X1	Nat X	umgekehrte Reihenfolge der Zeichen des Arguments
SIN	Num1	Num	Sinus von Num1
SQRT	Num1	Num	Quadratwurzel
STANDARD-DEVIATION	Num1...	Num	Standardabweichung der Argumentwerte
SUM	I1... oder Num1...	I Num	Summe der Argumentwerte
TAN	Num1	Num	Tangens von Num1
UPPER-CASE	A1 oder Nat1 oder X1	Nat X	alle Buchstaben im Argument werden in Großbuchstaben umgesetzt
VARIANCE	Num1...	Num	Varianz der Argumentwerte
WHEN-COMPILED	keine	X	Datum und Uhrzeit der Programmübersetzung
YEAR-TO-YYYY	I1, I2	I	Umwandlung der 2-stelligen Jahreszahl I1 in eine 4-stellige Jahreszahl, abhängig vom Wert von I2

9.2.1 ACOS - Arcuscosinus

Die ACOS-Funktion liefert einen numerischen Wert im Bogenmaß, der den Arcuscosinus von argument-1 darstellt.
Funktionstyp: numerisch.

Format

FUNCTION ACOS (argument-1)

Argumente

1. argument-1 muss der Klasse numerisch angehören.
2. Der Wert von argument-1 muss größer oder gleich -1 und kleiner oder gleich +1 sein.

Returnwerte

1. Der Returnwert ist der Arcuscosinus von argument-1 und ist größer oder gleich null und kleiner oder gleich pi.
2. Der Fehler-Returnwert ist -2.

Siehe auch: COS, SIN, ASIN, TAN, ATAN

Beispiel 9-1

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  AS PIC S9V999 VALUE -0.25.
01  R PIC 9V99.
01  RES PIC +9.99.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION ACOS (AS).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: +1.82

9.2.2 ADDR - Adresse eines Bezeichners

Die ADDR-Funktion liefert eine ganze Zahl, die die Adresse von argument-1 darstellt.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION ADDR (argument-1)
```

Argument

1. argument-1 kann ein Literal oder ein Datenelement jeder Klasse (ausgenommen die Klassen objekt, zeiger und typisiert) oder Kategorie sein.

Returnwert

1. Der Returnwert ist eine ganze Zahl, die die Adresse von argument-1 zur Ablaufzeit darstellt.

9.2.3 ANNUITY - Annuität

Die ANNUITY-Funktion liefert einen numerischen Wert, der die gleichbleibende Jahresrate (Annuität), ausgehend von einem Kreditbetrag 1, darstellt. Die Laufzeit des Kredits wird mit argument-2 angegeben. Der Zinssatz wird mit argument-1 angegeben und am Ende jeden Jahres der angegebenen Laufzeit berechnet. Funktionstyp: numerisch.

Format

```
FUNCTION ANNUITY (argument-1 argument-2)
```

Argumente

1. argument-1 muss zur Klasse numerisch gehören.
2. Der Wert von argument-1 muss größer oder gleich null sein.
3. argument-2 muss positiv ganzzahlig sein.

Returnwerte

1. Wenn der Wert von argument-1 gleich null ist, errechnet sich der Wert der Funktion aus: 1 / argument-2
2. Wenn der Wert von argument-1 ungleich null ist, errechnet sich der Wert der Funktion aus: argument-1 / (1 - (1 + argument-1)**(- argument-2))
3. Der Fehler-Returnwert ist -2.

Siehe auch: PRESENT-VALUE

Beispiel 9-2

Das folgende Programm berechnet die jährlichen Raten für einen Kredit in der Höhe von 100000 zu drei verschiedenen Zinssätzen bei einer Laufzeit von 1 bis 10 Jahren.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ZINSTAB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS FENSTER
    DECIMAL-POINT IS COMMA.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 KAPITAL             PIC 9(9).
01 LZ                  PIC 99.
01 RECHEN-TABELLE.
    02 ZINSSATZ PIC V9(7) OCCURS 3 INDEXED BY R-IND-S.
 01 TITELZEILE.
    02 PIC XX VALUE SPACE.
    02 OCCURS 3 INDEXED BY T-IND-S.
      10 ZINS-ED PIC BBBZZ9,999999B.
      10 PIC X VALUE FROM (1) "%" REPEATED TO END.
 01 AUSGABE-TABELLE.
    02 ZEILE OCCURS 10 INDEXED BY A-IND-Z.
      10 LAUFZEIT  PIC Z9.
      10 RATE PIC BZZZBZZZBZZ9,99 OCCURS 3 INDEXED BY A-IND-S.
 PROCEDURE DIVISION.
 ONLY SECTION.
```

```
 PARA.
     MOVE 100000 TO KAPITAL
*** Zinssatz 5,75 % ***
     MOVE 0,0575 TO ZINSSATZ (1)
*** Zinssatz 8,90 % ***
     MOVE 0,0890 TO ZINSSATZ (2)
*** Zinssatz 12,10 % ***
     MOVE 0,1210 TO ZINSSATZ (3)
     PERFORM VARYING R-IND-S FROM 1 BY 1 UNTIL R-IND-S > 3
       SET T-IND-S TO R-IND-S
       MULTIPLY ZINSSATZ (R-IND-S) BY 100 GIVING ZINS-ED (T-IND-S)
     END-PERFORM
     PERFORM VARYING A-IND-Z FROM 1 BY 1 UNTIL A-IND-Z > 10
       PERFORM VARYING A-IND-S FROM 1 BY 1 UNTIL A-IND-S > 3
          SET R-IND-S TO A-IND-S
          SET LZ TO A-IND-Z
          MOVE LZ TO LAUFZEIT (A-IND-Z)
          COMPUTE RATE (A-IND-Z  A-IND-S) = KAPITAL *
              FUNCTION ANNUITY (ZINSSATZ (R-IND-S) LZ)
       END-PERFORM
     END-PERFORM
     DISPLAY TITELZEILE UPON FENSTER
     PERFORM VARYING A-IND-Z FROM 1 BY 1 UNTIL A-IND-Z > 10
       DISPLAY ZEILE (A-IND-Z) UPON FENSTER
     END-PERFORM
     STOP RUN.
```

Ergebnis:

	5,750000 %	8,900000 %	12,100000 %
1	105 750,00	108 900,00	112 100,00
2	54 352,67	56 769,79	59 247,57
3	37 238,06	39 435,08	41 706,466
4	28 694,12	30 799,14	32 992,81
5	23 578,41	25 642,57	27 809,72
6	20 176,80	22 225,55	24 391,49
7	17 754,64	19 802,43	21 981,30
8	15 944,62	18 000,34	20 200,66
9	14 542,66	16 612,13	18 839,28
10	13 426,32	15 513,49	17 770,92

9.2.4 ASIN - Arcussinus

Die ASIN-Funktion liefert einen numerischen Wert im Bogenmaß, der den Arcussinus von argument-1 darstellt. Funktionstyp: numerisch.

Format

```
FUNCTION ASIN (argument-1)
```

Argumente

1. argument-1 muss der Klasse numerisch angehören.
2. Der Wert von argument-1 muss größer oder gleich -1 und kleiner oder gleich +1 sein.

Returnwerte

1. Der Returnwert ist der Arcussinus von argument-1 und ist größer oder gleich -pi/2 und kleiner oder gleich +pi/2.
2. Der Fehler-Returnwert ist -2.

Siehe auch: SIN, COS, ACOS, TAN, ATAN

Beispiel 9-3

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  AN PIC S9V999 VALUE -0.45.
01  R PIC 9V99.
01  RES PIC -9.99.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION ASIN (AN).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 0.46

9.2.5 ATAN - Arcustangens

Die ATAN-Funktion liefert einen numerischen Wert im Bogenmaß, der den Arcustangens von argument-1 darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION ATAN (argument-1)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwert

1. Der Returnwert ist der Arcustangens von argument-1 und ist größer als -pi/2 und kleiner als +pi/2.

Siehe auch: TAN, SIN, ASIN, COS, ACOS

Beispiel 9-4

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 9V9999.
01  RES PIC -9.9999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION ATAN (-0.45).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 0.4228

9.2.6 BYTE-LENGTH - Anzahl Bytes

Die BYTE-LENGTH-Funktion liefert eine ganze Zahl, die die Länge von argument-1 in Bytes darstellt. Funktionstyp: ganzzahlig.

Format

```
FUNCTION BYTE-LENGTH (argument-1)
```

Argumente

1. argument-1 muss ein alphanumerisches oder nationales Literal sein oder ein Datenelement jeder Klasse oder Kategorie oder ein Typname.
2. Ist irgendein argument-1 untergeordnetes Datenfeld mit der DEPENDING-Angabe der OCCURS-Klausel beschrieben, so wird der Inhalt des DEPENDING-Datenfeldes zum Zeitpunkt der Auswertung der LENGTH-Funktion verwendet.

Returnwerte

1. Wenn argument-1 ein nichtnumerisches Literal, ein Datenelement oder ein Gruppenfeld ist, das kein variabel langes Datenfeld enthält, dann ist der Returnwert die Länge von argument-1 in Bytes.

   > ℹ Handelt es sich bei argument-1 um eine Objektreferenz, so wird die Länge der Objektreferenz und nicht die Größe des Objektes selbst geliefert.

2. Ist argument-1 ein Gruppenfeld, dem ein variabel langes Datenfeld untergeordnet ist, so ist der Returnwert die Länge von argument-1 in Bytes. Sie berechnet sich entsprechend den Regeln für ein Sendefeld mit OCCURS-Klausel.
3. Der Returnwert zählt vorkommende implizite FILLER-Zeichen in argument-1 mit.

Siehe auch: LENGTH

Beispiel 9-5

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
MAIN.
    COMPUTE RES = FUNCTION LENGTH (N"anita beiss sie, bat ina").
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 048

9.2.7 CHAR-NATIONAL - Zeichen in der nationalen Sortierfolge

Die CHAR-NATIONAL-Funktion liefert dasjenige Zeichen, dessen Ordnungsposition innerhalb der nationalen Sortierfolge mit argument-1 bestimmt ist.
Funktionstyp: national.

Format

```
FUNCTION CHAR-NATIONAL (argument-1)
```

Argumente

1. argument-1 muss ganzzahlig sein.
2. Der Wert von argument-1 muss größer als null und kleiner oder gleich 65536 sein.

Returnwerte

1. Es wird als Returnwert dasjenige Zeichen geliefert, dessen Ordnungsposition in UTF-16 mit argument-1 bestimmt ist.
2. Der Fehler-Returnwert ist ein Leerzeichen.

Siehe auch: ORD

Beispiel 9-7

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LETTER PIC 999 VALUE 64.
01  R PIC N.
PROCEDURE DIVISION.
MAIN.
    MOVE FUNCTION CHAR-NATIONAL (LETTER) TO R.
    STOP RUN.
```

Ergebnis: R enthält das Zeichen N“@“ in UTF-16 Darstellung.

9.2.8 CHAR - Zeichen in der alphanumerischen Sortierfolge

Die CHAR-Funktion liefert dasjenige Zeichen, dessen Ordnungsposition innerhalb der alphanumerischen Sortierfolge mit argument-1 bestimmt ist.
Funktionstyp: alphanumerisch.

Format

```
FUNCTION CHAR (argument-1)
```

Argumente

1. argument-1 muss ganzzahlig sein.
2. Der Wert von argument-1 muss größer als null und kleiner oder gleich der Anzahl Positionen in der Sortierfolge sein.

Returnwerte

1. Wenn mehrere Zeichen in der Sortierfolge die gleiche Position haben, wird als Returnwert dasjenige Zeichen des ersten Literals geliefert, das für diese Zeichenposition in der ALPHABET-Klausel angegeben ist.
2. Der Fehler-Returnwert ist ein Leerzeichen.

Siehe auch: ORD

Example 9-6

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LETTER PIC 999 VALUE 194.
01  R PIC X.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION CHAR (LETTER) TO R.
    DISPLAY R UPON T.
    STOP RUN.
```

Ergebnis: A

„A“ hat im EBCDI-Code die Ordnungsposition 194.

9.2.9 COS - Cosinus

Die COS-Funktion liefert den Cosinus des Winkels oder Bogens, der in argument-1 im Bogenmaß angegeben ist.
Funktionstyp: numerisch.

Format

```
FUNCTION COS (argument-1)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwert

1. Der Returnwert ist der Cosinus von argument-1 und ist größer oder gleich -1 und kleiner oder gleich +1.

Siehe auch: ACOS, SIN, ASIN, TAN, ATAN

Beispiel 9-8

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC S9V9(10).
01  RES PIC -9.9(10).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION COS (3.1425).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: -0.9999995883

9.2.10 CURRENT-DATE - Aktuelles Datum

Die CURRENT-DATE-Funktion liefert einen 21 Zeichen langen alphanumerischen Wert, der das aktuelle Kalenderdatum und die aktuelle Uhrzeit darstellt.
Funktionstyp: alphanumerisch.

Format

```
FUNCTION CURRENT-DATE
```

Returnwert

1. Die Zeichenpositionen des Returnwertes enthalten (von links nach rechts gezählt):

Character position	Contents
1-4	Vier Ziffern für die Jahresangabe (Gregorianischer Kalender)
5-6	Zwei Ziffern für die Monatsangabe; Wertebereich: 01 bis 12
7-8	Zwei Ziffern für die Tagesangabe; Wertebereich: 01 bis 31
9-10	Zwei Ziffern für die Stundenangabe; Wertebereich: 00 bis 23
11-12	Zwei Ziffern für die Minutenangabe; Wertebereich: 00 bis 59
13-14	Zwei Ziffern für die Sekundenangabe; Wertebereich: 00 bis 59
15-21	0000000

Siehe auch: DATE-OF-INTEGER, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, WHEN-COMPILED

Beispiel 9-9

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DATUM PIC XXXX/XX/XXBBXXBXXBXXBXXBBX(5).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION CURRENT-DATE TO DATUM.
    DISPLAY DATUM UPON T.
```

Ergebnis: 1995/08/10 14 36 19 00 00000

9.2.11 DATE-OF-INTEGER - Datumskonversion

Die DATE-OF-INTEGER-Funktion wandelt eine angegebene Anzahl Tage um in das entsprechende Standard-Datumsformat (JJJJMMDD).
Funktionstyp: ganzzahlig.

Format

```
FUNCTION DATE-OF-INTEGER (argument-1)
```

Argument

1. argument-1 ist eine positive Ganzzahl, die eine Anzahl von seit dem 31.12.1600 (gemäß Gregorianischem Kalender) vergangenen Tagen darstellt.

Returnwerte

1. Der Returnwert ist das ISO-Standarddatum, das der in argument-1 angegebenen Anzahl Tage entspricht.
2. JJJJ ist das Jahr gemäß Gregorianischem Kalender, MM ist der Monat dieses Jahres, DD ist der Tag dieses Monats.
3. Der Fehler-Returnwert ist 0.

Siehe auch: DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, CURRENT-DATE, WHEN-COMPILED

Beispiel 9-10

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DATUM PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE DATUM = FUNCTION DATE-OF-INTEGER (50000).
    DISPLAY DATUM UPON T.
    STOP RUN.
```

Ergebnis: 17371123
Der 50000.Tag seit dem 31.12.1600 war der 23.11.1737.

9.2.12 DATE-TO-YYYYMMDD - Jahreszahlkonversion

Die DATE-TO-YYYYMMDD-Funktion wandelt ein durch argument-1 angegebenes Datum vom Standard-Datumsformat mit 2-stelliger Jahreszahl in ein Datum vom StandardDatumsformat mit 4-stelliger Jahreszahl um. Das Ende des 100-Jahr Intervalls, in welches das in argument-1angegebene Jahr fällt, wird bestimmt, indem argument-2 zum aktuellen Jahr (das Jahr, in dem die Funktion ausgeführt wird) addiert wird („gleitendes Fenster “).
Funktionstyp: ganzzahlig.

Format

```
FUNCTION DATE-TO-YYYYMMD (argument-1 [argument-2])
```

Argumente

1. argument-1 muss eine positive ganze Zahl kleiner als 1000000 sein.
2. argument-2, wenn angegeben, muss eine ganze Zahl sein.
3. Falls argument-2 nicht angegeben ist, wird als zweites Argument der Wert 50 angenommen.
4. Die Summe aus dem aktuellen Jahr und argument-2 muss kleiner als 10000 und größer als 1699 sein.
5. Es wird nicht überprüft, ob argument-1 ein gültiges Datum ist. Das heißt, die Werte 0 und 999999 sind auch dann gültige Argumente für die Funktion
   DATE-TO-YYYYMMDD, wenn durch eine der Optionen
   CHECK-FUNKTION-ARGUMENTS = YES oder
   SET-FUNCTION-ERROR-DEFAULT = YES eine Überprüfung der Argumente verlangt wird.

Returnwerte

1. Der Returnwert ist das in argument-1 angegebene Datum mit 4-stelliger Jahreszahl.
   Für ein Argument der Form JJMMDD ist der Returnwert definiert durch:FUNCTION YEAR-TO-YYYY (JJ, argument-2) * 10000 + MMTT
2. Der Fehler-Returnwert ist 0.

Siehe auch: DAY-TO-YYYYDDD, YEAR-TO-YYYY

Beispiel 9-11

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DATUM         PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE DATUM = FUNCTION DATE-TO-YYYYMMDD (590123).
    DISPLAY DATUM UPON T.
(1)
    COMPUTE DATUM = FUNCTION DATE-TO-YYYYMMDD (470101 -50).
    DISPLAY DATUM UPON T.
(2)
    STOP RUN.
```

Ein ausführlicheres Beispiel ist bei der Funktion YEAR-TO-YYYY zu finden.

Ergebnis:

Im Jahr 1996 liefert das Programm folgende Ergebnisse:
(1) 19590123
(2) 18470101

Im Jahr 2009 liefert das Programm folgende Ergebnisse:
(1) 20590123
(2) 19470101

9.2.13 DAY-OF-INTEGER - Datumskonversion

Die DAY-OF-INTEGER-Funktion wandelt eine angegebene Anzahl Tage um in das julianische Datumsformat (JJJJDDD).
Funktionstyp: ganzzahlig.

Format

```
FUNCTION DAY-OF-INTEGER (argument-1)
```

Argument

1. argument-1 ist eine positive Ganzzahl, die eine Anzahl von seit dem 31.12.1600 (gemäß Gregorianischem Kalender) vergangenen Tagen darstellt.

Returnwerte

1. Der Returnwert ist das julianische Datum, das der in argument-1 angegebenen Anzahl Tage entspricht.
2. JJJJ ist das Jahr gemäß Gregorianischem Kalender, DDD bezeichnet den Tag dieses Jahres.
3. Der Fehler-Returnwert ist 0.

Siehe auch: DATE-OF-INTEGER, INTEGER-OF-DAY, INTEGER-OF-DATE, CURRENT-DATE, WHEN-COMPILED

Beispiel 9-12

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DAYS PIC 9999 VALUE 5000.
01  DATUM PIC X(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE DATUM = FUNCTION DAY-OF-INTEGER (DAYS)
    DISPLAY DATUM UPON T.
    STOP RUN.
```

Ergebnis: 1614252
Der 5000.Tag seit dem 31.12.1600 war der 252.Tag des Jahres 1614.

9.2.14 DAY-TO-YYYYDDD - Jahreszahlkonversion

Die DAY-TO-YYYYDDD-Funktion wandelt ein durch argument-1 angegebenes Datum vom julianischen Datumsformat mit 2-stelliger Jahreszahl in ein Datum vom julianischen Datumsformat mit 4-stelliger Jahreszahl um. Das Ende des 100-Jahr Intervalls, in welches das in argument-1 angegebene Jahr fällt, wird bestimmt, indem argument-2 zum aktuellen Jahr (das Jahr, in dem die Funktion ausgeführt wird) addiert wird („gleitendes Fenster“).
Funktionstyp: ganzzahlig.

Format

```
FUNCTION DAY-TO-YYYYDDD (argument-1 [argument-2])
```

Argumente

1. argument-1 muss eine positive ganze Zahl kleiner als 100000 sein.
2. argument-2, wenn angegeben, muss eine ganze Zahl sein.
3. Falls argument-2 nicht angegeben ist, wird als zweites Argument der Wert 50 angenommen.
4. Die Summe aus dem aktuellen Jahr und argument-2 muss kleiner als 10000 und größer als 1699 sein.
5. Es wird nicht überprüft, ob argument-1 ein gültiges Datum ist. Das heißt, die Werte 0 und 99999 sind auch dann gültige Argumente für die Funktion DAY-TO-YYYYDDD, wenn durch eine der Optionen CHECK-FUNKTION-ARGUMENTS = YES oder SET-FUNCTION-ERROR-DEFAULT = YES eine Überprüfung der Argumente verlangt wird.

Returnwerte

1. Der Returnwert ist das in argument-1 angegebene Datum mit w4-stelliger Jahreszahl. Für ein Argument der Form JJTTT ist der Returnwert definiert durch:
   FUNCTION YEAR-TO-YYYY (JJ, argument-2) * 1000 + TTT
2. Der Fehler-Returnwert ist 0.

Siehe auch: DATE-TO-YYYYMMDD, YEAR-TO-YYYY

Beispiel 9-13

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DATUM PIC 9(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE DATUM = FUNCTION DAY-TO-YYYYDDD (59001).
    DISPLAY DATUM UPON T.                                    (1)
    COMPUTE DATUM = FUNCTION DAY-TO-YYYYDDD (47365 -50).
    DISPLAY DATUM UPON T.                                    (2)
    STOP RUN.
```

Ein ausführlicheres Beispiel ist bei der Funktion YEAR-TO-YYYY zu finden.

Ergebnis:

Im Jahr 1996 liefert das Programm folgende Ergebnisse:
(1) 1959001
(2) 1847365

Im Jahr 2009 liefert das Programm folgende Ergebnisse:
(1) 2059001
(2) 1947365

9.2.15 DISPLAY-OF - alphanumerische Zeichendarstellung

Die DISPLAY-OF-Funktion liefert eine Zeichenfolge, die die Zeichen des Arguments konvertiert zu alphanumerischer Darstellung enthält.
Funktionstyp: alphanumerisch.

Format

```
FUNCTION DISPLAY-OF (argument-1 [argument-2])
```

Argumente

1. argument-1 muss von der Klasse national und mindestens ein Zeichen lang sein.
2. argument-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.
3. argument-2 muss von der Klasse alphabetisch oder alphanumerisch und genau ein Zeichen lang sein. Bei der Konvertierung im Ergebnis gibt argument-2 ein Ersatzzeichen an Stelle derjenigen nationalen Zeichen an, für die es keine entsprechenden alphanumerischen Zeichen gibt.

Returnwerte

1. Es wird als Returnwert eine Zeichenfolge geliefert, in der jedes nationale Zeichen aus argument-1 zu seinem entsprechenden Zeichen in alphanumerischer Darstellung konvertiert ist.
2. Ist argument-2 angegeben, dann wird jedes Zeichen aus argument-1, für das es kein entsprechendes Zeichen in alphanumerischer Darstellung gibt, zu dem durch argument-2 angegebenen Ersatzzeichen konvertiert.
3. Ist argument-2 nicht angegeben und enthält argument-1 Zeichen, für die es kein entsprechendes Zeichen in alphanumerischer Darstellung gibt, werden diese zu einem von XHCS festgelegten Ersatzzeichen (Punkt '.') konvertiert. Die Ausnahmesituation EC-DATA-CONVERSION tritt auf.
4. Der Returnwert enthält genauso viele Zeichen wie argument-1.
5. Der Fehler-Returnwert ist ein Leerzeichen.

> ℹ Steht FUNCTION DISPLAY-OF als Sendefeld in einer MOVE-Anweisung, für die die Überprüfung der Ausnahmesituation EC-DATA-CONVERSION eingeschaltetet ist und auch eine USE-Prozedur existiert, so bleibt das Empfangsfeld unverändert, wenn die Ausnahmesituation auftritt.

Siehe auch: NATIONAL-OF

Beispiel 9-14

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 nat PIC XXX VALUE NX"E23A005A0040".
01 ersatz PIC X VALUE "z".
01 R PIC xxx.
PROCEDURE DIVISION.
MAIN.
     DISPLAY FUNCTION DISPLAY-OF (nat, ersatz).
     STOP RUN.
```

Ergebnis: zZ@

9.2.16 EXCEPTION-STATUS - Ausnahmezustand

Die EXCEPTION-STATUS-Funktion liefert den Namen des letzten Ausnahmezustands.
Funktionstyp: alphanumerisch.

Format

```
FUNCTION EXCEPTION-STATUS
```

Returnwert

1. Der Returnwert ist eine 31 Zeichen lange alphanumerische Zeichenkette, die linksbündig den Namen des letzten Ausnahmezustands enthält. Zeigt der letzte Ausnahmezustand an, dass kein Ausnahmezustand ausgelöst ist, enthält die Zeichenkette 31 Leerzeichen.

9.2.17 FACTORIAL - Fakultät

Die FACTORIAL-Funktion liefert einen ganzzahligen Wert, die die Fakultät von argument-1 darstellt.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION FACTORIAL (argument-1)
```

Argument

1. argument-1 muss eine Ganzzahl größer oder gleich null und kleiner oder gleich 29 sein.

Returnwerte

1. Enthält argument-1 den Wert 0, ist der Returnwert 1.
2. Ist der Wert von argument-1 positiv, wird die Fakultät dieses Wertes zurückgeliefert.
3. Der Fehler-Returnwert ist -2.

Siehe auch: LOG, LOG10, SQRT

Beispiel 9-15

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION FACTORIAL (07).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 00005040

9.2.18 INTEGER - Nächstkleinere Ganzzahl

Die INTEGER-Funktion liefert den größten ganzzahligen Wert, der kleiner oder gleich ist dem in argument-1 angegebenen Wert.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION INTEGER (argument-1)
```

Argument

1. argument-1 muss der Klasse numerisch angehören und muss einen Wert enthalten, der größer oder gleich - 10^{31}+1 und kleiner 10^{31} ist.

Returnwerte

1. Der Returnwert ist die größte Ganzzahl, die kleiner oder gleich dem Wert von argument-1 ist. Ist z.B. der Wert von argument-1 -1,5, dann wird -2 zurückgeliefert; ist der Wert von argument-1 +1,5, dann wird +1 zurückgeliefert.
2. Der Fehler-Returnwert ist - 9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: INTEGER-PART

Beispiel 9-16

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION INTEGER (-3.3).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 00000004

9.2.19 INTEGER-OF-DATE - Datumskonversion

Die INTEGER-OF-DATE-Funktion wandelt ein Datum vom Standard-Datumsformat um in die entsprechende Anzahl Tage, die seit dem 31.12.1600 vergangen sind.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION INTEGER-OF-DATE (argument-1)
```

Argument

1. argument-1 muss eine Ganzzahl der Form JJJJMMTT sein, deren Wert sich folgendermaßen errechnet: (JJJJ * 10000) + (MM * 100) + TT
   a. JJJJ bezeichnet das Jahr im Gregorianischen Kalender. Es muss eine Ganzzahl größer 1600 sein.
   b. MM bezeichnet einen Monat und muss eine positive Ganzzahl kleiner 13 sein.
   c. TT bezeichnet einen Tag und muss eine Ganzzahl kleiner 32 sein, die für den angegebenen Monat und das angegebene Jahr gültig sein muss. sein muss.

Returnwerte

1. Der Returnwert ist eine Ganzzahl, die die Anzahl von Tagen darstellt, die seit dem 31.12.1600 (einem Sonntag) bis zum in argument-1 angegebenen Datum vergangen sind.
2. Der Fehler-Returnwert ist 0.

Siehe auch: INTEGER-OF-DAY, DATE-OF-INTEGER, DAY-OF-INTEGER, CURRENT-DATE, WHEN-COMPILED

Beispiel 9-17

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TAGE PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE TAGE = FUNCTION INTEGER-OF-DATE (19530410).
    DISPLAY TAGE UPON T.
    STOP RUN.
```

Ergebnis: 00128665

Der 10.4.1953 war der 128665.Tag seit dem 31.12.1600.

9.2.20 INTEGER-OF-DAY - Datumskonversion

Die INTEGER-OF-DAY-Funktion wandelt ein Datum vom julianischen Datumsformat um in die entsprechende Anzahl Tage, die seit dem 31.12.1600 vergangen sind.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION INTEGER-OF-DAY (argument-1)
```

Argument

1. argument-1 muss eine Ganzzahl der Form JJJJTTT sein, deren Wert sich folgendermaßen errechnet: (JJJJ * 1000) + TTT
   a. JJJJ bezeichnet das Jahr im Gregorianischen Kalender. Es muss eine Ganzzahl größer 1600 sein.
   b. TTT bezeichnet den Tag des Jahres und muss eine Ganzzahl kleiner 367 sein, wobei 366 nur angegeben werden darf, wenn das angegebene Jahr ein Schaltjahr ist.

Returnwerte

1. Der Returnwert ist eine Ganzzahl, die die Anzahl von Tagen darstellt, die seit dem 31.12.1600 bis zu dem in argument-1 angegebenen Datum vergangen sind.
2. Der Fehler-Returnwert ist 0.

Siehe auch: INTEGER-OF-DATE, DAY-OF-INTEGER, DATE-OF-INTEGER, CURRENT-DATE, WHEN-COMPILED

Beispiel 9-18

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TAGE PIC 9(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE TAGE = FUNCTION INTEGER-OF-DAY (1993299).
    DISPLAY TAGE UPON T.
    STOP RUN.
```

Ergebnis: 0143474
Der 299.Tag des Jahres 1993 ist der 143474.Tag seit dem 31.12.1600.

9.2.21 INTEGER-PART - Ganzzahliger Teil eines Gleitpunktwertes

Die INTEGER-PART-Funktion liefert den ganzzahligen Teil von argument-1.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION INTEGER-PART (argument-1)
```

Argumente

1. argument-1 muss der Klasse numerisch angehören. Für den Wert w von argument-1 muss gelten:

   $-10^{31} < w < 10^{31}$

Returnwerte

1. Ist der Wert von argument-1 null, ist der Returnwert null.
2. Ist der Wert von argument-1 positiv, ist der Returnwert die größte ganze Zahl, die kleiner oder gleich dem Wert von argument-1 ist. Ist z.B. der Wert von argument-1 +1,5, dann ist der Returnwert +1.
3. Ist der Wert von argument-1 negativ, ist der Returnwert die größte ganze Zahl, die größer oder gleich dem Wert von argument-1 ist. Ist z.B. der Wert von argument-1 -1,5, dann ist der Returnwert -1.
4. Der Fehler-Returnwert ist -9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: INTEGER

Beispiel 9-19

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION INTEGER-PART (-3.3).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: -00000003

9.2.22 LENGTH - Anzahl Zeichen

Die LENGTH-Funktion liefert eine ganze Zahl, die die Länge von argument-1 in Zeichenpositionen darstellt. Funktionstyp: ganzzahlig.

Format

```
FUNCTION LENGTH (argument-1)
```

Argumente

1. argument-1 kann ein nichtnumerisches Literal oder ein Datenelement jeder Klasse oder Kategorie oder ein Typname sein.
2. Ist irgendein argument-1 untergeordnetes Datenfeld mit der DEPENDING-Angabe der OCCURS-Klausel beschrieben, so wird der Inhalt des DEPENDING-Datenfeldes zum Zeitpunkt der Auswertung der LENGTH-Funktion verwendet.

Returnwerte

1. Wenn argument-1 ein nichtnumerisches Literal, ein Datenelement oder ein Gruppenfeld ist, das kein variabel langes Datenfeld enthält, dann ist der Returnwert eine ganze Zahl, die die Länge von argument-1 in Zeichenpositionen darstellt.

   > ¡ Handelt es sich bei argument-1 um eine Objektreferenz, so wird die Länge der Objektreferenz und **nicht** die Größe des Objektes selbst geliefert.

2. Ist argument-1 ein Gruppenfeld, dem ein variabel langes Datenfeld untergeordnet ist, so ist der Returnwert die Länge von argument-1 in Bytes. Sie berechnet sich entsprechend den Regeln für ein Sendefeld mit OCCURS-Klausel.
3. Der Returnwert zählt vorkommende implizite FILLER-Zeichen in argument-1 mit.

Beispiel 9-20

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
MAIN.
    COMPUTE RES = FUNCTION LENGTH ("anita beiss sie, bat ina").
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 024

9.2.23 LOG10 - Logarithmus zur Basis 10

Die LOG10-Funktion liefert einen numerischen Wert, der den Logarithmus zur Basis 10 von argument-1 darstellt. Funktionstyp: numerisch.

Format

```
FUNCTION LOG10 (argument-1)
```

Argumente

1. argument-1 muss der Klasse numerisch angehören.
2. Der Wert von argument-1 muss größer als null sein.

Returnwerte

1. Der Returnwert ist der Logarithmus zur Basis 10 von argument-1.
2. Der Fehler-Returnwert ist -9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: LOG, FACTORIAL, SQRT

Beispiel 9-22

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  L PIC S99V999 VALUE 3.
01  R PIC S99V999999.
01  RES PIC 99.999999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION LOG10 (L).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 00.477121

9.2.24 LOG - Logarithmus

Die LOG-Funktion liefert einen numerischen Wert, der den Logarithmus zur Basis e (natürlicher Logarithmus) von argument-1 darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION LOG (argument-1)
```

Argumente

1. argument-1 muss der Klasse numerisch angehören.
2. Der Wert von argument-1 muss größer als null sein.

Returnwerte

1. Der Returnwert ist der Logarithmus zur Basis e von argument-1.
2. Der Fehler-Returnwert ist - 9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: LOG10, FACTORIAL, SQRT

Example 9-21

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  L PIC S99V999 VALUE 3.
01  R PIC S99V999999.
01  RES PIC 99.999999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION LOG (L).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 01.098612

9.2.25 LOWER-CASE - Kleinbuchstaben

Die LOWER-CASE-Funktion wandelt alle Großbuchstaben in einer Zeichenkette um in die entsprechenden Kleinbuchstaben.
Der Funktionstyp ist abhängig vom angegebenen Argumenttyp:

Argumenttyp	Funktionstyp
alphabetisch alphanumerisch national	alphanumerisch alphanumerisch national

Format

```
FUNCTION LOWER-CASE (argument-1)
```

Argumente

1. argument-1 muss von der Klasse alphabetisch, alphanumerisch oder national sein und muss mindestens ein Zeichen lang sein.
2. argument-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.

Returnwerte

1. Der Returnwert ist die gleiche Zeichenkette wie argument-1, außer dass jeder Großbuchstabe in den entsprechenden Kleinbuchstaben umgewandelt ist.
2. Die zurückgelieferte Zeichenkette hat dieselbe Länge wie argument-1.
3. Umlaute werden nicht umgesetzt.
4. Der Fehler-Returnwert ist ein Leerzeichen.

Siehe auch: UPPER-CASE

Beispiel 9-23

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC X(20).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION LOWER-CASE ("GROSS klein") TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: gross klein

9.2.26 MAX - Maximalwert

Die MAX-Funktion liefert den höchsten Wert aus einer Reihe von Argumentwerten.
Der Funktionstyp ist abhängig vom angegebenen Argumenttyp:

Argumenttyp	Funktionstyp
alphabetisch	alphanumerisch
alphanumerisch	alphanumerisch
alle Argumente ganzzahlig	ganzzahlig
index	index
national	national
numerisch	numerisch
numerisch/ganzzahlig	numerisch

Format

```
FUNCTION MAX ({argument-1}...)
```

Argumente

1. Sind mehrere Argumente angegeben, müssen alle Argumente derselben Klasse angehören.
2. Die einzelnen Argumente dürfen nicht mit der ANY LENGTH-Klausel definiert sein.

Returnwerte

1. Der Returnwert ist der Inhalt desjenigen Arguments, das den höchsten Wert enthält. Der höchste Wert wird entsprechend den Vergleichsregeln für einfache Bedingungen bestimmt.
2. Bei mehreren Argumenten mit dem gleichen (Höchst-)Wert gilt der Wert des am weitesten links stehenden Arguments als Returnwert.
3. Ist der Funktionstyp alphanumerisch oder national, ist die Länge des Returnwerts identisch mit der Länge des entsprechenden Arguments.
4. Der Fehler-Returnwert ist 0.

Siehe auch: MIN, ORD-MAX, ORD-MIN, RANGE, MEAN, MEDIAN, MIDRANGE, SUM

Example 9-24

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
01  RES1 PIC X(4).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION MAX (12 32 5 8 17 9).
    MOVE FUNCTION MAX ("HUGO" "EGON" "THEO" "OTTO") TO RES1.
    DISPLAY "Höchster Argumentwert RES: " RES UPON T.
    DISPLAY "Höchster Argumentwert RES1: " RES1 UPON T.
    STOP RUN.
```

Ergebnis: Höchster Argumentwert RES: 032
Höchster Argumentwert RES1: THEO

9.2.27 MEAN - Arithmetischer Mittelwert

Die MEAN-Funktion liefert einen numerischen Wert, der das arithmetische Mittel (Durchschnitt) der angegebenen Argumentwerte darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION MEAN ({argument-1}...)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist das arithmetische Mittel der angegebenen Argumentwerte.
2. Der Returnwert errechnet sich aus der Summe der Argumentwerte geteilt durch die Anzahl der angegebenen Argumente.
3. Der Fehler-Returnwert ist 0.

Siehe auch: MIN, MAX, RANGE, MEDIAN, MIDRANGE, SUM

Beispiel 9-25

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 999V999.
01  RES PIC 999.999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION MEAN (12 32 5 8 17 9).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 013.833

9.2.28 MEDIAN - Mittlerer Argumentwert

Die MEDIAN-Funktion liefert den Wert desjenigen Arguments, das in der Mitte der aufsteigend oder absteigend sortierten Argumente steht.
Funktionstyp: numerisch.

Format

```
FUNCTION MEDIAN ({argument-1}...)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist der Wert des mittleren Arguments aus der Reihe der sortierten Argumente.
2. Ist die Anzahl der Argumente ungerade, ist mindestens die Hälfte der Argumentwerte größer oder gleich bzw. kleiner oder gleich dem Returnwert. Ist die Anzahl der Argumente gerade, wird der Returnwert aus dem arithmetischen Mittel der beiden mittleren Argumentwerte gebildet.
3. Der Vergleich zum Sortieren der Argumentwerte erfolgt nach den Regeln für einfache Bedingungen (siehe Abschnitt „Bedingungen").
4. Der Fehler-Returnwert ist 0.

Siehe auch: MIN, MAX, RANGE, MEAN, MIDRANGE, SUM

Beispiel 9-26

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION MEDIAN (2 32 8 128 16 64 256).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 032

9.2.29 MIDRANGE - Mittelwert

Die MIDRANGE-Funktion liefert das arithmetische Mittel (Durchschnittswert) aus dem niedrigsten und dem höchsten angegebenen Argumentwert.
Funktionstyp: numerisch.

Format

```
FUNCTION MIDRANGE ({argument-1}...)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist der arithmetische Mittelwert aus dem höchsten und dem niedrigsten der angegebenen Argumentwerte. Der Wertevergleich zur Bestimmung des höchsten und niedrigsten Wertes erfolgt nach den Regeln für einfache Bedingungen (siehe Abschnitt „Bedingungen").
2. Der Fehler-Returnwert ist 0.

Siehe auch: MIN, MAX, RANGE, MEAN, MEDIAN, SUM

Beispiel 9-27

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 999V999.
01  RES PIC 999.999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION MIDRANGE (12 32 5 8 17 9).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 018.500

9.2.30 MIN - Minimalwert

Die MIN-Funktion liefert den niedrigsten Wert aus einer Reihe von Argumentwerten.
Der Funktionstyp ist abhängig vom angegebenen Argumenttyp:

Argumenttyp	Funktionstyp
alphabetisch	alphanumerisch
alphanumerisch	alphanumerisch
alle Argumente ganzzahlig	ganzzahlig
index	index
national	national
numerisch	numerisch
numerisch/ganzzahlig	numerisch

Format

```
FUNCTION MIN ({argument-1}...)
```

Argumente

1. Sind mehrere Argumente angegeben, müssen alle Argumente derselben Klasse angehören.
2. Die einzelnen Argumente dürfen nicht mit der ANY LENGTH-Klausel definiert sein.

Returnwerte

1. Der Returnwert ist der Inhalt desjenigen Arguments, das den niedrigsten Wert enthält. Der niedrigste Wert wird entsprechend den Vergleichsregeln für einfache Bedingungen bestimmt.
2. Bei mehreren Argumenten mit dem gleichen (niedrigsten) Wert gilt der Wert des am weitesten links stehenden Arguments als Returnwert.
3. Ist der Funktionstyp alphanumerisch oder national, ist die Länge des Returnwerts identisch mit der Länge des entsprechenden Arguments.
4. Der Fehler-Returnwert ist 0.

Siehe auch: MAX, RANGE, MEAN, MEDIAN, MIDRANGE, SUM

Beispiel 9-28

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION MIN (12 32 5 8 17 9).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 005

9.2.31 MOD - Modulo

Die MOD-Funktion liefert einen ganzzahligen Wert, der argument-1 modulo argument-2 darstellt.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION MOD (argument-1 argument-2)
```

Argumente

1. argument-1 und argument-2 müssen ganzzahlig sein.
2. Für den Wert w von argument-1 muss gelten:

   $-10^{31} < w < 10^{31}$

   Liegt der Wert von argument-1 außerhalb dieses Bereichs, so ist eine Berechnung des Funktionswertes zwar möglich, seine Korrektheit aber nicht gewährleistet.
3. Der Wert von argument-2 darf nicht 0 sein.

Returnwerte

1. Der Returnwert ist argument-1 modulo argument-2. Der Returnwert ist definiert als:

   argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))
2. Der Fehler-Returnwert ist - 9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: REM

Beispiel 9-29

argument-1	argument-2	Returned value
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

9.2.32 NATIONAL-OF - nationale Zeichendarstellung

Die NATIONAL-OF-Funktion liefert eine Zeichenfolge, die die Zeichen des Arguments konvertiert zu nationaler Darstellung enthält.
Funktionstyp: alphanumerisch.

Format

```
FUNCTION NATIONAL-OF (argument-1 [argument-2])
```

Argumente

1. argument-1 muss von der Klasse alphabetisch oder alphanumerisch und mindestens ein Zeichen lang sein.
2. argument-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.
3. argument-2 muss von der Klasse national und genau ein Zeichen lang sein. Bei der Konvertierung im Ergebnis gibt argument-2 ein Ersatzzeichen an Stelle derjenigen alphanumerischen Zeichen an, für die es keine entsprechenden nationalen Zeichen gibt.

Returnwerte

1. Es wird als Returnwert eine Zeichenfolge geliefert, in der jedes alphanumerische Zeichen aus argument-1 zu seinem entsprechenden Zeichen in nationaler Darstellung konvertiert ist.
2. Ist argument-2 angegeben, dann wird jedes Zeichen aus argument-1, für das es kein entsprechendes Zeichen in nationaler Darstellung gibt, zu dem durch argument-2 angegebenen Ersatzzeichen konvertiert.
3. Ist argument-2 nicht angegeben und enthält argument-1 Zeichen, für die es kein entsprechendes Zeichen in nationaler Darstellung gibt, werden diese zu einem von XHCS festgelegten Ersatzzeichen konvertiert (Punkt '.'). Die Ausnahmesituation EC-DATA-CONVERSION tritt auf.
4. Der Returnwert enthält genauso viele Zeichen wie argument-1.
5. Der Fehler-Returnwert ist ein Leerzeichen.

> i Steht FUNCTION NATIONAL-OF als Sendefeld in einer MOVE-Anweisung, für die die Überprüfung der Ausnahmesituation EC-DATA-CONVERSION eingeschaltetet ist und auch eine USE-Prozedur existiert, so bleibt das Empfangsfeld unverändert, wenn die Ausnahmesituation auftritt. Den Gegensatz dazu bildet eine entsprechende MOVE-Anweisung mit impliziter Konvertierung.

Siehe auch: DISPLAY-OF

Beispiel 9-30

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 alfa PIC XXX VALUE "ABC".
01 R PIC NNN.
PROCEDURE DIVISION.
MAIN.
    MOVE FUNCTION NATIONAL-OF (alfa) TO R.
    STOP RUN.
```

Ergebnis: R enthält die Zeichen ABC in UTF-16-Darstellung.

9.2.33 NUMVAL-C - Numerischer Wert einer Zeichenkette mit optionalem Währungszeichen

Die NUMVAL-C-Funktion liefert den numerischen Wert, den die mit argument-1 angegebene Zeichenkette darstellt. Das wahlweise mit argument-2 angegebene Währungszeichen bleibt ebenso unberücksichtigt wie jedes wahlweise dem Dezimalpunkt vorausgehende Komma.
Funktionstyp: numerisch.

Format

```
FUNCTION NUMVAL-C (argument-1 [argument-2])
```

Argumente

1. argument-1 muss von der Kategorie alphanumerisch oder national sein und eines der folgenden zwei Formate besitzen:

   ```
   ['BLANK'] [+ | -] ['BLANK'] [wz] ['BLANK'] {digit[,digit]...[.[digit]] | .
   digit} ['BLANK']
   oder
   ['BLANK'] [wz] ['BLANK'] {digit[,digit]...[.[digit]] | .digit} ['BLANK'] [+ |
   - | CR | DB ] ['BLANK']
   ```

'BLANK'	ein oder mehr Leerzeichen
wz	Währungszeichen: Zeichenkette mit einem oder mehreren Zeichen (argument-2)
digit	ein oder mehrere Ziffern aus der Menge 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

2. Für CR und DB ist die Buchstabenfolge „CR“ bzw. „DB“ in Großbuchstaben anzugeben.
3. Ist die DECIMAL-POINT IS COMMA-Klausel im SPECIAL-NAMES-Paragrafen angegeben, werden die Funktionen von Komma und Dezimalpunkt in argument-1 vertauscht.
4. Die Gesamtzahl der Ziffern in argument-1 darf 31 nicht überschreiten.
5. argument-2 muss, wenn angegeben, von derselben Klasse wie argument-1 sein.
6. Ist argument-2 nicht angegeben, wird als Währungszeichen das programmspezifische Währungszeichen-Symbol verwendet.

Returnwerte

1. Der Returnwert ist der numerische Wert von argument-1.
2. Hat argument-1 eine feste Länge mit bis zu 14 Zeichen, so ist der Fehler-Returnwert -999’999’999’999’999’999.
   Hat argument-1 eine variable Länge oder eine Länge mit mehr als 14 Zeichen, so ist der Fehler-Returnwert -9’999’999’999’999’999’999’999’999’999’999’999.

Siehe auch: NUMVAL

Beispiel 9-32

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  V PIC X(8) VALUE "- $15.00".
01  R PIC 99V99.
01  RES PIC 99.99.
```

```
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION NUMVAL-C (V).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 15.00

9.2.34 NUMVAL - Numerischer Wert einer Zeichenkette

Die NUMVAL-Funktion liefert den numerischen Wert, den die mit argument-1 angegebene Zeichenkette darstellt. Vorausgehende und nachfolgende Leerzeichen bleiben unberücksichtigt.
Funktionstyp: numerisch.

Format

```
FUNCTION NUMVAL (argument-1)
```

Argumente

1. argument-1 muss von der Kategorie alphanumerisch oder national sein und eines der folgenden zwei Formate besitzen:

   ```
   ['BLANK'] [+ | -] ['BLANK'] {digit[.[digit]] | .digit} ['BLANK']
   oder
   ['BLANK'] {digit[.[digit]] | .digit} ['BLANK'] [+ | - | CR | DB]
   ```

'BLANK'	ein oder mehr Leerzeichen
digit	ein oder mehrere Ziffern aus der Menge 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

2. Für CR und DB ist die Buchstabenfolge „CR“ bzw. „DB“ in Großbuchstaben anzugeben.
3. Die Gesamtzahl der Ziffern in argument-1 darf 31 nicht überschreiten.
4. Ist die DECIMAL-POINT IS COMMA-Klausel im SPECIAL-NAMES-Paragrafen angegeben, muss ein Komma in argument-1 als Dezimalpunkt angegeben werden.

Returnwerte

1. Der Returnwert ist der numerische Wert von argument-1.
2. Hat argument-1 eine feste Länge mit bis zu 14 Zeichen, so ist der Fehler-Returnwert -9’999’999’999’999.
   Hat argument-1 eine variable Länge oder eine Länge mit mehr als 14 Zeichen, so ist der Fehler-Returnwert -999’999’999’999’999’999.

Siehe auch: NUMVAL-C

Beispiel 9-31

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  V PIC X(8) VALUE "+ 15.00".
01  R PIC 99V99.
01  RES PIC 99.99.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION NUMVAL (V).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 15.00

9.2.35 ORD-MAX - Position des höchstwertigen Arguments

Die ORD-MAX-Funktion liefert einen ganzzahligen Wert, der angibt, welches der angegebenen Argumente - von links nach rechts gezählt - den höchsten Wert enthält.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION ORD-MAX ({argument-1}...)
```

Argumente

1. Ist mehr als ein argument-1 angegeben, müssen alle Argumente derselben Klasse angehören.
2. argument-1 darf nicht von der Klasse objekt oder der Klasse zeiger sein.

Returnwerte

1. Der Returnwert gibt die Position an, die das höchstwertige Argument in der Reihe der angegebenen Argumente einnimmt.
2. Die Bestimmung des höchstwertigen Arguments erfolgt nach den Regeln für einfache Bedingungen (siehe Abschnitt „Bedingungen").
3. Bei mehreren Argumenten mit dem gleichen (Höchst-)Wert gilt der Wert des am weitesten links stehenden Arguments als Returnwert.
4. Der Fehler-Returnwert ist 0.

Siehe auch: ORD-MIN, MAX, MIN

Beispiel 9-34

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION ORD-MAX (13 4 9 18 5 7).
    DISPLAY R UPON T.
    STOP RUN.
```

Ergebnis: 004
Das vierte Argument (18) hat den höchsten Wert.

9.2.36 ORD-MIN - Position des niedrigstwertigen Arguments

Die ORD-MIN-Funktion liefert einen ganzzahligen Wert, der angibt, welches der Argumente - von links nach rechts gezählt - den niedrigsten Wert enthält.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION ORD-MIN ({argument-1}...)
```

Argumente

1. Ist mehr als ein argument-1 angegeben, müssen alle Argumente derselben Klasse angehören.
2. argument-1 darf nicht von der Klasse objekt oder der Klasse zeiger sein.

Returnwerte

1. Der Returnwert gibt die Position an, die das niedrigstwertige Argument in der Reihe der angegebenen Argumente einnimmt.
2. Die Bestimmung des niedrigstwertigen Arguments erfolgt nach den Regeln für einfache Bedingungen (siehe Abschnitt „Bedingungen").
3. Bei mehreren Argumenten mit dem gleichen (niedrigsten) Wert gilt der Wert des am weitesten links stehenden Arguments als Returnwert.
4. Der Fehler-Returnwert ist 0.

Siehe auch: ORD-MAX, MAX, MIN

Example 9-35

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION ORD-MIN ("Z" "3" "B" "?" "a").
    DISPLAY R UPON T.
    STOP RUN.
```

Ergebnis: 004

Das vierte Argument ("?") hat den niedrigsten Wert.

9.2.37 ORD - Ordnungsposition in der Sortierfolge

Die ORD-Funktion liefert einen ganzzahligen Wert, der die Ordnungsposition von argument-1 in der programmspezifischen Sortierfolge angibt. Die niedrigste Ordnungsposition ist 1.
Funktionstyp: ganzzahlig.

Format

```
FUNCTION ORD (argument-1)
```

Argument

1. argument-1 muss ein Zeichen lang sein und der Kategorie alphabetisch, alphanumerisch oder national angehören.

Returnwert

1. Ist argument-1 von der Klasse alphabetisch oder alphanumerisch, dann ist der Returnwert die Ordnungsposition von argument-1 in der alphanumerischen Sortierfolge.
2. Ist argument-1 von der Klasse national, dann ist der Returnwert die Ordnungsposition von argument-1 in der nationalen Sortierfolge.

Siehe auch: CHAR

Beispiel 9-33

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  L PIC X VALUE "Z".
01  R PIC X(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION ORD (L).
    DISPLAY R UPON T.
    STOP RUN.
```

Ergebnis: 234
Der Buchstabe Z hat im EBCDI-Code die Ordnungsposition 234.

9.2.38 PRESENT-VALUE - Zeitwert (Tilgungsbetrag)

Die PRESENT-VALUE-Funktion liefert aus der Summe einer Reihe von Ratenzahlungen den Tilgungsanteil. Die Ratenzahlungen werden mit argument-2 angegeben. Der Zinssatz, mit dem die Raten berechnet wurden, wird mit argument-1 angegeben.
Funktionstyp: numerisch.

Format

```
FUNCTION PRESENT-VALUE (argument-1 {argument-2}...)
```

Argumente

1. argument-1 und argument-2 müssen der Klasse numerisch angehören.
2. Der Wert von argument-1 muss größer als -1 sein.

Returnwerte

1. Der Returnwert ist die Summe einer Reihe von Ergebnissen, von denen jedes einzelne nach folgender Berechnungsformel zustandekommt:

   argument-2 / (1 + argument-1) ** n

   Der Exponent n wird für jede Berechnung schrittweise um 1 erhöht.
2. Der Fehler-Returnwert ist -9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: ANNUITY

Beispiel 9-36

```
 ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*** Zinssatz 10% **********
 01  ZINS PIC 9V99 VALUE 0.10.
*** Vier Kreditraten *********
 01  B-1 PIC 9(4) VALUE 1000.
 01  B-2 PIC 9(4) VALUE 2000.
 01  B-3 PIC 9(4) VALUE 1000.
 01  B-4 PIC 9(4) VALUE 1000.
*** Zeitwert = Tilgungsbetrag ***
 01  ZW PIC 9(6).
 PROCEDURE DIVISION.
 P1 SECTION.
 MAIN.
     COMPUTE ZW = FUNCTION PRESENT-VALUE (ZINS B-1 B-2 B-3 B-4).
     DISPLAY "getilgte Summe: " ZW UPON T.
     STOP RUN.
```

Ergebnis: getilgte Summe: 003996
Dies ist der nach Zahlung der vier Kreditarten getilgte Betrag.

9.2.39 RANDOM - Zufallszahl

Die RANDOM-Funktion liefert einen numerischen Wert, der eine Zufallszahl aus einer Gleichverteilungsmenge darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION RANDOM [(argument-1)]
```

Argumente

1. argument-1 muss, wenn angegeben, null oder eine positive Ganzzahl sein. argument-1 wird als Ausgangswert verwendet, um eine Folge von Zufallszahlen zu erzeugen.
2. Bei jeder weiteren Referenz auf argument-1 wird eine neue Folge von Zufallszahlen begonnen.
3. Ist in der ersten Referenz innerhalb der Ablaufeinheit argument-1 nicht angegeben, ist der Ausgangswert 0.
4. Jede weitere Referenz ohne Angabe von argument-1 liefert die nächste Zahl in der laufenden Folge.

Returnwerte

1. Der Returnwert ist größer oder gleich null und kleiner als eins.
2. Für einen vorgegebenen Ausgangswert ist die Folge der Zufallszahlen immer die gleiche.
3. Der Wertebereich von argument-1, der verschiedene Folgen von Zufallszahlen liefert, liegt zwischen 0 und $2^{31}-1$.
4. Der Fehler-Returnwert ist -1.

Beispiel 9-37

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOTTO.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS VIDEO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LISTE.
 05 ELEM        OCCURS 6 INDEXED BY SI, LI.
  10 Z         PIC Z9 VALUE FROM (1) IS ZERO REPEATED TO END.
  10 T         PIC XX VALUE FROM (1) IS ", " REPEATED TO END.
01 Z-0         PIC Z9.
01 RAN-VAL     PIC V9(18) BINARY.
01 INIT-ARG    PIC 9(5) BINARY.
01 ID-1.
 02            PIC X(5).
 02 D1         PIC 9.
 02            PIC X.
 02 D2         PIC 9.
 02            PIC X.
 02 D3         PIC 9.
 02            PIC X.
 02 D4         PIC 9.
 02            PIC X.
 02 D5         PIC 9.
 02            PIC X(7).
```

```
01 D6           PIC 9.
PROCEDURE DIVISION.
M SECTION.
M1.
*
* Ermittlung eines von Uhrzeit, Datum und Wochentag
* abhaengigen Start-Arguments fuer die Random-Funktion
*
    MOVE FUNCTION CURRENT-DATE TO ID-1
    ACCEPT D6 FROM DAY-OF-WEEK
    COMPUTE INIT-ARG =
        (10 * D1 + 1000 * D2 + 100 * D3 + D4 + 10000 * D5) * D6
*
* Ermittlung der ersten Zufallszahl
*
    COMPUTE RAN-VAL = FUNCTION RANDOM (INIT-ARG)
*
* Durchlaufen der Schleife, bis 6 Elemente in die Liste
* eingetragen wurden
*
    PERFORM VARYING LI FROM 1 BY 1 UNTIL LI > 6
*
* Durchlaufen der Schleife, bis eine eindeutige Zahl ermittelt
* und in das aktuelle Listenelement eingetragen wurde
*
    PERFORM UNTIL Z (LI) NOT ZERO
*
* Abbildung des Returnwerts der RANDOM-Funktion
* auf eine Ganzzahl zwischen 1 und 49
    COMPUTE Z-0 = FUNCTION INTEGER (49 * RAN-VAL) + 1
    SET SI TO 1
*
* Pruefung des Ergebnisses auf Eindeutigkeit
*
    SEARCH ELEM
*
* Zahl wurde in der Liste nicht gefunden -> Zahl wird eingetragen
*
    AT END MOVE Z-0 TO Z (LI)
*
* Zahl ist in der Liste bereits enthalten -> neue Zufallszahl
*
    WHEN Z (SI) = Z-0 CONTINUE
  END-SEARCH
*
* Ermittlung einer weiteren Zufallszahl
*
    COMPUTE RAN-VAL = FUNCTION RANDOM
  END-PERFORM
END-PERFORM
SORT ELEM ASCENDING Z
MOVE "." TO T (6)
DISPLAY LISTE UPON VIDEO
STOP RUN.
```

Ergebnis: 6 Zahlen zwischen 1 und 49.

9.2.40 RANGE - Differenzwert

Die RANGE-Funktion liefert einen numerischen Wert, der die Differenz zwischen dem höchstwertigen und dem niedrigstwertigen Argument darstellt. Der Funktionstyp ist abhängig von den Argumenttypen:

Argumenttyp	Funktionstyp
alle Argumente ganzzahlig alle Argumente numerisch gemischt ganzzahlig/numerisch	ganzzahlig numerisch numerisch

Format

```
FUNCTION RANGE ({argument-1}...)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist die Differenz aus höchstwertigem argument-1 und niedrigstwertigem argument-1. Die Bestimmung des höchsten und niedrigsten Wertes erfolgt nach den Regeln für einfache Bedingungen (siehe Abschnitt „Vergleichsbedingung").
2. Der Fehler-Returnwert ist -1.

Siehe auch: MIN, MAX, MEAN, MEDIAN, MIDRANGE, ORD-MAX, ORD-MIN, SUM

Beispiel 9-38

```
 ...
DATA DIVISION.
 WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
 MAIN.
    COMPUTE RES = FUNCTION RANGE (12 32 5 8 17 9).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 027

9.2.41 REM - Divisionsrest

Die REM-Funktion liefert einen numerischen Wert, der den Restbetrag der Division von argument-1 durch argument-2 darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION REM (argument-1 argument-2)
```

Argumente

1. argument-1 und argument-2 müssen der Klasse numerisch angehören.
2. Der Wert von argument-2 darf nicht 0 sein.
3. Für den Wert w des Quotienten argument-1 / argument-2 muss gelten:

   $-10^{31} < w < 10^{31}$

Returnwerte

1. Der Returnwert ist der Restbetrag der Division von argument-1 geteilt durch argument-2. Er ist definiert durch:

   argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1 / argument-2))
2. Der Fehler-Returnwert ist - 9'999'999'999'999'999'999'999'999'999'999.

Siehe auch: MOD

Beispiel 9-39

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION REM (928 14).
    DISPLAY R UPON T.
    STOP RUN.
```

Ergebnis: 004

9.2.42 REVERSE - Umgekehrte Zeichenreihenfolge

Die REVERSE-Funktion liefert eine Zeichenkette mit der gleichen Länge und den gleichen Zeichen, wie in argument-1 angegeben, nur dass die Reihenfolge der Zeichen gegenüber argument-1 umgekehrt ist.
Der Funktionstyp ist abhängig vom angegebenen Argumenttyp:

Argumenttyp	Funktionstyp
alphabetisch alphanumerisch national	alphanumerisch alphanumerisch national

Format

```
FUNCTION REVERSE (argument-1)
```

Argumente

1. argument-1 muss der Klasse alphabetisch, alphanumerisch oder national angehören und muss mindestens 1 Zeichen lang sein.
2. argument-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.

Returnwerte

1. Ist argument-1 eine Zeichenkette der Länge n, so ist der Returnwert eine Zeichenkette der Länge n, wobei für 1 <= j <= n das Zeichen auf Position j des Returnwertes dem Zeichen auf Position n - j + 1 von argument-1 entspricht.
2. Der Fehler-Returnwert ist ein Leerzeichen.

Beispiel 9-40

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  REV PIC X(43) VALUE "Vitaler Nebel mit Sinn ist im Leben relativ".
01  RES PIC X(43).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION REVERSE (REV) TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: vitaler nebeL mi tsi nniS tim lebeN relatiV

9.2.43 SIN - Sinus

Die SIN-Funktion liefert den Sinus des Winkels oder Bogens, der in argument-1 im Bogenmaß angegeben ist. Funktionstyp: numerisch.

Format

```
FUNCTION SIN (argument-1)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwert

1. Der Returnwert ist der Sinus von argument-1 und ist größer oder gleich -1 und kleiner oder gleich +1.

Siehe auch: ASIN, COS, ACOS, TAN, ATAN

Beispiel 9-41

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 R PIC S9V9(10).
01 RES PIC -9.9(10).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION SIN (3.1425).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: -0.0009073462

9.2.44 SQRT - Quadratwurzel

Die SQRT-Funktion liefert einen numerischen Wert, der die Quadratwurzel von argument-1 darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION SQRT (argument-1)
```

Argumente

1. argument-1 muss der Klasse numerisch angehören.
2. Der Wert von argument-1 muss null oder positiv sein.

Returnwerte

1. Der Returnwert ist der Absolutwert der Quadratwurzel von argument-1.
2. Der Fehler-Returnwert ist -2.

Siehe auch: FACTORIAL, LOG, LOG10

Beispiel 9-42

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 R PIC 99V999999.
01 RES PIC 99.999999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION SQRT (33).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 05.744562

9.2.45 STANDARD-DEVIATION - Standardabweichung

Die STANDARD-DEVIATION-Funktion liefert einen numerischen Wert, der die Standardabweichung der angegebenen Argumente darstellt.
Funktionstyp: numerisch.

Format

```
FUNCTION STANDARD-DEVIATION ({argument-1}...)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist die Standardabweichung der angegebenen Argumente.
2. Der Returnwert errechnet sich wie folgt:
   a. Die Differenz zwischen jedem Argument und dem arithmetischen Mittel aller Argumente wird berechnet und quadriert.
   b. Die so berechneten Werte werden addiert. Der Additionsbetrag wird dividiert durch die Anzahl der angegebenen Argumente.
   c. Aus dem so gewonnenen Quotienten wird die Quadratwurzel gebildet. Der Returnwert ist der Absolutbetrag dieser Quadratwurzel.
3. Besteht die Argumentfolge nur aus einem Wert oder besteht sie aus variabel vielen Datenelementen und die Gesamtzahl dieser Datenelemente beträgt 1, so ist der Returnwert 0.
4. Der Fehler-Returnwert ist -1.

Siehe auch: VARIANCE

Beispiel 9-43

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC 99V9999.
01  RES PIC 99.9999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION STANDARD-DEVIATION (2 4 6).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 01.6329

9.2.46 SUM - Summe der Argumentwerte

Die SUM-Funktion liefert einen Wert, der die Summe aller angegebenen Argumente darstellt.
Der Funktionstyp ist abhängig von den Argumenttypen:

Argumenttyp	Funktionstyp
alle Argumente ganzzahlig alle Argumente numerisch gemischt ganzzahlig/numerisch	ganzzahlig numerisch numerisch

Format

```
FUNCTION SUM ({argument-1}...)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist die Summe aller Argumentwerte.
2. Der Fehler-Returnwert ist 0.

Siehe auch: MAX, MIN, RANGE, MEAN, MEDIAN, MIDRANGE

Beispiel 9-44

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION SUM (12 32 5 8 17 9).
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 00000083

9.2.47 TAN - Tangens

Die TAN-Funktion liefert den Tangens des Winkels oder Bogens, der in argument-1 im Bogenmaß angegeben ist.
Funktionstyp: numerisch.

Format

```
FUNCTION TAN (argument-1)
```

Argument

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist der Tangens von argument-1.
2. Der Fehler-Returnwert ist -9’999’999’999’999’999’999’999’999’999’999.

Siehe auch: ATAN, SIN, ASIN, COS, ACOS

Beispiel 9-45

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 R PIC S9V9(10).
01 RES PIC -9.9(10).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION TAN (3.1425).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 0.0009073466

9.2.48 UPPER-CASE - Großbuchstaben

Die UPPER-CASE-Funktion wandelt alle Kleinbuchstaben in einer Zeichenkette um in die entsprechenden Großbuchstaben.
Der Funktionstyp ist abhängig vom angegebenen Argumenttyp:

Argumenttyp	Funktionstyp
alphabetisch alphanumerisch national	alphanumerisch alphanumerisch national

Format

```
FUNCTION UPPER-CASE (argument-1)
```

Argumente

1. argument-1 muss der Klasse alphabetisch, alphanumerisch oder national angehören und muss mindestens ein Zeichen lang sein.
2. argument-1 darf nicht mit der ANY LENGTH-Klausel definiert sein.

Returnwerte

1. Der Returnwert ist die gleiche Zeichenkette wie argument-1, außer dass jeder Kleinbuchstabe in den entsprechenden Großbuchstaben umgewandelt ist.
2. Die zurückgelieferte Zeichenkette hat dieselbe Länge wie argument-1.
3. Umlaute werden nicht umgesetzt.
4. Der Fehler-Returnwert ist ein Leerzeichen.

Siehe auch: LOWER-CASE

Beispiel 9-46

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC X(20).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION UPPER-CASE ("frauenlob") TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: FRAUENLOB

9.2.49 VARIANCE - Varianz

Die VARIANCE-Funktion liefert einen numerischen Wert, der die Varianz der angegebenen Argumente darstellt. Funktionstyp: numerisch.

Format

```
FUNCTION VARIANCE ({argument-1}...)
```

Argumente

1. argument-1 muss der Klasse numerisch angehören.

Returnwerte

1. Der Returnwert ist die Varianz der angegebenen Argumente.
2. Der Returnwert ist definiert als Quadrat der Standardabweichung der Argumentfolge (siehe Funktion STANDARD-DEVIATION).
3. Besteht die Argumentfolge nur aus einem Wert oder besteht sie aus variabel vielen Datenelementen und die Gesamtzahl dieser Datenelemente beträgt 1, so ist der Returnwert 0.
4. Der Fehler-Returnwert ist -1.

Siehe auch: STANDARD-DEVIATION

Beispiel 9-47

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 R PIC 99V9999.
01 RES PIC 99.9999.
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION VARIANCE (2 4 6).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Ergebnis: 02.6666

9.2.50 WHEN-COMPILED - Datum und Uhrzeit der Übersetzung

Die WHEN-COMPILED-Funktion liefert Datum und Uhrzeit der Übersetzung des Programms.
Funktionstyp: alphanumerisch.

Format

```
FUNCTION WHEN-COMPILED
```

Returnwert

1. Die Zeichenpositionen des Returnwerts enthalten (von links nach rechts gezählt):

Zeichenposition	Inhalt
1-4	vier Ziffern für die Jahresangabe (Gregorianischer Kalender)
5-6	zwei Ziffern für die Monatsangabe; Wertebereich: 01 bis 12
7-8	zwei Ziffern für die Tagesangabe; Wertebereich: 01 bis 31
9-10	zwei Ziffern für die Stundenangabe; Wertebereich: 00 bis 23
11-12	zwei Ziffern für die Minutenangabe; Wertebereich: 00 bis 59
13-14	zwei Ziffern für die Sekundenangabe; Wertebereich: 00 bis 59
15-21	0000000

Siehe auch: CURRENT-DATE, DATE-OF-INTEGER, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY

Beispiel 9-48

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DATUM PIC X(21).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION WHEN-COMPILED TO DATUM.
    DISPLAY DATUM UPON T.
    STOP RUN.
```

Ergebnis: 199604211434270000000

9.2.51 YEAR-TO-YYYY - Jahreszahlenkonversion

Die YEAR-TO-YYYY-Funktion wandelt eine 2-stellige Jahreszahl in eine 4-stellige Jahreszahl um. Das Ende des 100-Jahr Intervalls, in welches das in argument-1 angegebene Jahr fällt, wird bestimmt, indem argument-2 zum aktuellen Jahr (das Jahr, in dem die Funktion ausgeführt wird) addiert wird („gleitendes Fenster“). Funktionstyp: ganzzahlig.

Format

```
FUNCTION YEAR-TO-YYYY (argument-1 [argument-2])
```

Argumente

1. argument-1 muss eine positive ganze Zahl kleiner als 100 sein.
2. argument-2, wenn angegeben, muss eine ganze Zahl sein.
3. Falls argument-2 nicht angegeben ist, wird als zweites Argument der Wert 50 angenommen.
4. Die Summe aus dem aktuellen Jahr und argument-2 muss kleiner als 10000 und größer als 1699 sein.

Returnwerte

1. Der Returnwert ist die in argument-1 angegebene Jahreszahl ergänzt durch das Jahrhundert.
   Der Returnwert ist abhängig vom Zwischenwert

   $L_1L_2L_3L_4$ = aktuelles Jahr +argument-2 (= letztes Jahr des 100-Jahr Intervalls).
   Das Jahrhundert wird folgendermaßen berechnet:

100 * L_1L_2 + JJ	falls L_3L_4 >= JJ	(JJ=argument-1)
100 * (L_1L_2 -1) + JJ	falls L_3L_4 < JJ	

2. Der Fehler-Returnwert ist 0.

Siehe auch: DATE-TO-YYYYMMDD, DAY-TO-YYYYDDD

Beispiel 9-49

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DATUM               PIC 9(7).
01 AKTUELLES-JAHR      PIC 9(7).
01 JAHR                PIC 9(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
*
* Berechnung der Funktion mit gleitendem Fenster:
*
* Das 100-Jahr Intervall, in das das berechnete Jahr fällt, soll
* die Jahre (aktuelles Jahr -35) bis (aktuelles Jahr +64) umfassen:
*
    COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (59 64).
    DISPLAY DATUM UPON T. (1)
*
* Ohne 2. Argument (bzw. 2. Argument =50)
* Das 100-Jahr Intervall umfasst die Jahre (aktuelles Jahr -49)
* bis (aktuelles Jahr +50):
```

```
*
    COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (0).
    DISPLAY DATUM UPON T. (2)
*
* Das 2. Argument kann auch negativ sein
* Das 100-Jahr Intervall umfasst die Jahre (aktuelles Jahr -109)
* bis (aktuelles Jahr -10):
*
    COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (96 -10).
    DISPLAY DATUM UPON T. (3)
*
* Berechnung der Funktion mit festem Fenster
*
* Das 100-Jahr Intervall, in das das berechnete Jahr fällt, soll
* die Jahre 1950 bis 2049 umfassen:
*
* Berechnung des letzten Jahres des 100-Jahr Intervalls
* relativ zum aktuellen Jahr
*
    MOVE FUNCTION CURRENT-DATE(1:4) TO AKTUELLES-JAHR.
    COMPUTE JAHR = 2049 - AKTUELLES-JAHR.
* Berechnung der Funktionswerte
    COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (50 JAHR).
    DISPLAY DATUM UPON T. (4)
    COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (1 JAHR).
    DISPLAY DATUM UPON T. (5)
* Das 100-Jahr Intervall, in das das berechnete Jahr fällt, soll
* die Jahre 1890 bis 1989 umfassen:
*
* Berechnung des letzten Jahres des 100-Jahr Intervalls
* relativ zum aktuellen Jahr
*
    MOVE FUNCTION CURRENT-DATE(1:4) TO AKTUELLES-JAHR.
    COMPUTE JAHR = 1989 - AKTUELLES-JAHR.
* Berechnung der Funktionswerte
  COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (89 JAHR).
  DISPLAY DATUM UPON T. (6)
  COMPUTE DATUM = FUNCTION YEAR-TO-YYYY (90 JAHR).
  DISPLAY DATUM UPON T. (7)
  STOP RUN.
```

Ergebnis:

Im Jahr 1996 liefert das Programm folgende Ergebnisse:
(1) 2059
(2) 2000
(3) 1896
(4) 1950
(5) 2001
(6) 1989
(7) 1890

Im Jahr 2050 liefert das Programm folgende Ergebnisse:
(1) 2059
(2) 2100
(3) 1996

(4) 1950
(5) 2001
(6) 1989
(7) 1890

10 Listenprogramm (Report-Writer)

In diesem Kapitel werden folgende Themen behandelt:

10.1 Allgemeine Beschreibung

Die speziell zur Erzeugung von Listen zur Verfügung stehenden Sprachmittel (Listenprogramm-Sprachelemente) erlauben es, in der DATA DIVISION Format und Inhalte einer Liste so ausführlich zu beschreiben, dass in der PROCEDURE DIVISION nur noch wenige Anweisungen nötig sind, um die Liste zu erstellen.

Eine ausgedruckte Liste gibt eine bestimmte Information formatiert wieder. Die Gesamtzahl aller Dateien und Listenerklärungen in einer Übersetzungseinheit darf 254 nicht übersteigen.

In der DATA DIVISION werden

- die Listen benannt, die das Listenprogramm erstellen soll,
- den Listen eine Datei (Listendatei) zugeordnet, in die sie geschrieben werden sollen,
- die Formate und Informationsversorgung der Listen beschrieben.

In der PROCEDURE DIVISION werden die Anweisungen angegeben, die die Erstellung der Listen bewirken.

Das Listenprogramm erstellt die gewünschten Listen gemäß den definierten Formaten und führt dabei folgende Aktionen aus:

- formatgerechtes Übertragen der Programmdaten in die Listen,
- Bilden von Datensummen, die zu Teil- und Endsummen addiert werden,
- Aktualisieren der Zeilen- und Seitenzähler,
- Druckaufbereitung der erzeugten Listen.

Das in diesem Handbuch beschriebene Listenprogramm ist dem optionalen Modul Report Writer des ANS85 funktional gleichwertig und unterscheidet sich von ihm nur in einigen syntaktischen Details.

10.1.1 Allgemeine Beschreibung der DATA DIVISION

In der DATA DIVISION muss mit einer Dateierklärung (FD) eine Ausgabedatei deklariert werden, in die die erzeugten Listen geschrieben werden sollen. Die Dateierklärung enthält auch die Namen der Listen (siehe „REPORT-Klausel").

Als letztes Kapitel der DATA DIVISION muss die REPORT SECTION angegeben werden, in der die Listen bezüglich ihrer Formate und Informationsversorgung beschrieben werden. Dieses Kapitel enthält zweierlei Eintragungen:

1. Die **Listenerklärung** (RD)
   Sie muss den Namen der Liste enthalten. Ein Seitenformat (PAGE LIMIT-Klausel), ein Zeichen (CODE-Klausel), das jeder Zeile der Liste zur Trennung von verschiedenen Listen vorangestellt wird (darf allerdings nicht gedruckt werden), und eine Hierarchie von Gruppenwechseldatenfeldern (CONTROL-Klausel) können hier ebenfalls angegeben werden.
2. Die **Leistenerklärung**
   In ihr werden unter anderem die Druckfelder der Leiste beschrieben, das sind Datenfelder mit Zeilen- und Spaltenpositionierung, die mit Information direkt (VALUE) oder indirekt (Sendefeld, Summen) versorgt werden.

In der Listenerklärung wird durch die Angabe der Zeilenzahl pro Seite und der Zeilennummern als Grenze von leistenspezifischen Zonen das Seitenformat bestimmt. Mit diesen Angaben wird bei der Erstellung der Liste die Positionierung der verschiedenen Leisten gesteuert.

Die Angabe von **Gruppenwechseldatenfeldern** in ihrer hierarchischen Rangordnung hat den Zweck, die Postenleisten zu strukturieren.

Die Postenleisten, die funktionell mit den hierarchischen Begriffen verknüpft sind, werden so in der Reihenfolge ihrer Erstellung zu einzelnen Gruppen zusammengefasst, dass allen Postenleisten einer Gruppe derselbe Wert des hierarchisch niedrigsten Begriffes zukommt. Je nachdem, ob für diese hierarchische Stufe ein Gruppenkopf und/oder ein Gruppenfuß definiert ist, wird jede Gruppe mit dem Gruppenkopf eingeleitet und mit dem Gruppenfuß abgeschlossen. Der Gruppenfuß enthält im Allgemeinen Information, die sich nur auf die Gruppe beschränkt, z.B. Zwischensummen.

Ganz analog zu den Postenleisten wird, wenn ein weiterer hierarchischer Begriff vorliegt, die oben beschriebene Gruppe von Postenleisten ihrerseits in Gruppen zusammengefasst, wobei jetzt der hierarchisch nächsthöhere Begriff maßgebend ist, usw.

Diese Strukturierung wird vom Listenprogramm folgendermaßen erzeugt:Vor der Erstellung einer Postenleiste testet das Listenprogramm die Gruppenwechseldatenfelder in ihrer hierarchischen Reihenfolge von oben nach unten. Sobald das Listenprogramm eine Wertänderung, d.h. einen **Gruppenwechsel** feststellt, werden zuerst alle vorhandenen Gruppenfüße in der hierarchischen Rangfolge von unten nach oben bis zur Stufe mit der zuerst festgestellten Wertänderung erstellt, dann alle vorhandenen Gruppenköpfe von derselben Stufe in umgekehrter hierarchischer Rangfolge und zuletzt die Postenleiste selbst.

Durch die Angabe von Gruppenwechseldatenfeldern in Verbindung mit den Leisten Gruppenkopf und Gruppenfuß hat der Programmierer die Möglichkeit, sich die Liste strukturiert erstellen zu lassen.

Die **Leistenerklärung**, die formal einer Satzbeschreibung gleicht, beschreibt die Eigenschaften aller Datenfelder der Liste. Über den Rahmen der in COBOL üblichen Beschreibungselemente für ein Datenfeld hinaus wird einem Listendatenfeld Zeile und Spalte, also eine Seitenposition zugeordnet, d.h. das Datenfeld wird zu einem Druckfeld erklärt. Jedes dieser Felder wird mit einer Information versorgt.

Drei Arten der Informationsversorgung sind möglich:

Die **SOURCE-Information** (Quellinformation der SOURCE-Klausel), die über ein Datenfeld zur Verfügung steht, das außerhalb der REPORT SECTION definiert ist.

Die **SUM-Information** (Summeninformation der SUM-Klausel) als Additionsergebnis von Daten, die ihrerseits wieder SOURCE-Information und/oder SUM-Information darstellen.

Die **VALUE-Information** (Wertinformation der VALUE-Klausel) als konstant festgelegter Informationswert.

Da eine Liste mehrere Arten von Leisten enthalten kann, enthält die Leistenbeschreibung eine Angabe über den Typ der Leiste. Die sieben möglichen Typen sind in Tabelle 38 aufgezeigt.

Mit der Listen- und allen zugehörigen Leistenerklärungen ist die Liste dem Format und dem Inhalt (einschließlich der nötigen Summationsoperationen) nach vollständig beschrieben, d.h. die Voraussetzungen für die Erzeugung der Liste sind abgeschlossen.

Typ	Definition
REPORT HEADING (Listenkopf)	Die Leiste wird pro Liste nur einmal erstellt und zwar als erste Leiste der Liste.
REPORT FOOTING (Listenfuß)	Diese Leiste bildet die Schlussleiste einer Liste und darf daher pro Liste nur einmal gedruckt werden.
PAGE HEADING (Seitenkopf)	Diese Leiste wird als erste Leiste einer Seite geschrieben; Ausnahme: 1. Der Seitenkopf wird nach dem Listenkopf geschrieben. 2. Kein Seitenkopf wird geschrieben, wenn die Seite explizit für den Listenkopf bzw. -fuß reserviert wurde.
PAGE FOOTING (Seitenfuß)	Diese Leiste wird als letzte Leiste einer Seite geschrieben; Ausnahme: entsprechend dem Seitenkopf.
DETAIL (Postenleiste)	Diese Leiste muss als einziger Leistentyp in einer GENERATE-Anweisung angegeben werden, um überhaupt erzeugt zu werden. Jede Ausführung dieser Anweisung erzeugt zur Ablaufzeit die angegebene Postenleiste.
CONTROL HEADING (Gruppenkopf)	Diese Leiste wird als Kopfleiste für eine Gruppe von Postenleisten infolge eines Gruppenwechsels erzeugt.
CONTROL FOOTING (Gruppenfuß)	Diese Leiste wird als Fußleiste für eine Gruppe von Postenleisten infolge eines Gruppenwechsels erzeugt. Sie enthält zumeist Summen überDaten, die die vorausgehende Gruppe von Postenleisten betreffen.

Tabelle 38: Leistentypen der Leistenbeschreibung einer Liste

10.1.2 Allgemeine Beschreibung der PROCEDURE DIVISION

In der PROCEDURE DIVISION veranlasst der Programmierer das Listenprogramm durch die Anweisungen INITIATE, GENERATE und TERMINATE, die gewünschte und im Datenteil vollständig beschriebene Liste zu erzeugen.

Die INITIATE-Anweisung bewirkt, dass das Listenprogramm für die in ihr angegebene(n) Liste(n) die zur Erstellung notwendige Initialisierung ausführt. Diese Initialisierung ermöglicht es dem Listenprogramm z.B. zu erkennen, ob eine GENERATE-Anweisung als im Programmablauf zeitlich erste GENERATE-Anweisung ausgeführt wird.

Die GENERATE-Anweisung ist für die Erstellung des Großteils der Liste zuständig. Sie gibt den Anstoß, dass neben der Erzeugung der Postenleiste eine Reihe automatischer Funktionen ausgeführt wird. So wird der Gruppentest und - wenn nötig - ein Gruppenwechsel durchgeführt, d.h. es werden Gruppenfüße und Gruppenköpfe erstellt. Ist für eine Rumpfleiste (Gruppenfuß, Gruppenkopf oder Postenleiste) kein Platz in der zugehörigen (Leisten-)-Zone der Seite, dann wird automatisch der Seitenfuß erstellt, ein Seitenvorschub impliziert und die Kopfleiste erzeugt. Alle bei der Erstellung der einzelnen Leisten anfallenden Detailaufgaben, wie z.B. das Erhöhen oder Zurücksetzen von Zählern, die Belegung der Druckfelder mit Wert-, Quell- oder Summeninformation, das Positionieren und Schreiben der Zeile(n), in der (denen) die Druckfelder einer Leiste organisiert sind, werden automatisch erledigt.

Die TERMINATE-Anweisung veranlasst das Listenprogramm, die Erzeugung der in ihr angegebenen Liste(n) abzuschließen. So werden alle zur Liste gehörigen Gruppenfüße und - falls vorhanden - der Listenfuß als letzte Leiste der Liste geschrieben.

Im Bereich der Prozedurvereinbarungen (DECLARATIVES) darf der Programmierer für Leisten (außer Postenleisten) Benutzerprozeduren im Anschluss an eine USE BEFORE REPORTING-Anweisung angeben. Da solche Prozeduren unmittelbar vor der eigentlichen Erstellung (Druckaufbereitung usw.) der zugehörigen Leiste durchlaufen werden, hat der Programmierer durch sie ein Instrument, auf die Druckstellung der Leiste zur Ablaufzeit Einfluss zu nehmen.

Folgende vier Sonderregister kennt und verwendet das Listenprogramm: LINE-COUNTER, PAGE-COUNTER, PRINT-SWITCH und CBL-CTR.

Eine Listendatei ist eine sequenziell organisierte Datei, für die die folgende Einschränkung gilt:

Vor einer INITIATE-Anweisung muss eine OPEN OUTPUT- oder OPEN EXTEND-Anweisung ausgeführt werden. Einer TERMINATE-Anweisung muss eine CLOSE-Anweisung (ohne REEL- oder UNIT-Angaben) folgen. Für diese Datei darf keine andere Ein-/Ausgabe-Anweisung ausgeführt werden.

10.2 Sprachelemente DATA DIVISION

In diesem Kapitel werden folgende Themen behandelt:

10.2.1 REPORT-Klausel

Funktion

Die REPORT-Klausel ordnet einer oder mehreren Listen, die in der REPORT SECTION beschrieben werden, eine Datei zu, in deren Dateierklärung (FD) diese Klausel angegeben ist. Jede dieser Listen wird in die durch die REPORT-Klausel zugeordnete und als Ausgabedatei beschriebene Datei zeilenweise geschrieben.

Format

```
{REPORT | REPORTS} {IS | ARE} {listenname-1}...
```

Syntaxregeln

1. Zu jedem Listennamen aus der REPORT-Klausel muss es eine Listenerklärung geben, die diesen Namen als Listennamen definiert.
2. Innerhalb einer Klassendefinition darf die REPORT-Klausel nur in einer Methodendefinition angegeben werden.

Allgemeine Regeln

1. Der Name jeder Liste, die durch das Listenprogramm erstellt werden soll, muss in der REPORT-Klausel der Dateierklärung derjenigen sequenziellen Ausgabedatei enthalten sein, in welche die Liste geschrieben werden soll.
2. Die Angabe mehrerer Listennamen in einer REPORT-Klausel ordnet diese Listen jener Datei zu, deren Dateierklärung die REPORT-Klausel enthält. Unabhängig von der Reihenfolge der angegebenen Listennamen, den Listenformaten, den Umfängen der Liste und dergleichen werden diese Listen bei ihrer Erstellung in die zugeordnete Datei
geschrieben.
3. Jeder durch eine Listenerklärung definierte Listenname muss in einer und nur einer REPORT-Klausel angegeben sein, d.h. eine Liste darf nur einer Datei zugeordnet werden.
4. Das Satzformat, variable, feste oder undefinierte Länge, wird durch die Angabe in der RECORD CONTAINS-Klausel bestimmt.

10.2.2 REPORT SECTION

Funktion

Die REPORT SECTION beschreibt die Formate und Inhalte der Listen, die erzeugt werden sollen.

Format

```
REPORT SECTION.

{listenerklärung {leistenerklärung} ... } ...
```

Allgemeine Regeln

1. Die REPORT SECTION muss das letzte Kapitel in der DATA DIVISION sein, falls nicht auch SUB-SCHEMA SECTION angegeben ist (siehe Allgemeines Format in Abschnitt „Struktur der DATA DIVISION").
2. Die Leistenerklärungen müssen alle der zugehörigen Listenerklärung geschlossen folgen.
3. Eine Leistenerklärung entspricht formal einer Satzbeschreibung der FILE SECTION oder WORKING-STORAGE SECTION. Die Leistenerklärung erfasst die Gesamtheit der Datenerklärungen für eine einzelne Leiste. Der erste Eintrag in der Leistenerklärung beginnt mit Stufennummer 01. Alle zugehörigen Datenerklärungen müssen mit Stufennummer 02 (siehe „Leistenerklärung") beginnen.

10.2.3 Listenerklärung

Funktion

Die Listenerklärung (RD) dient zur Benennung einer Liste, zur Festlegung eines Seitenformats, zur Kennzeichnung der Zeilen und zur Strukturierung einer Liste. Nach Bedarf können folgende Funktionen übernommen werden.

1. Die Angabe eines Zeichens zur Identifizierung der Druckzeilen einer Liste (CODE-Klausel).
2. Die Deklaration einer Gruppenhierarchie zur Strukturierung der Liste (CONTROL-Klausel).
3. Die Festlegung eines Seitenformats durch Angabe von vertikalen Grenzen für leistenspezifische Zonen (PAGE LIMIT-Klausel).

Format

```
RD listenname

    [CODE-Klausel]
    [CONTROL-Klausel]
    [GLOBAL-Klausel]
    [PAGE LIMIT-Klausel]
```

Syntaxregeln

1. RD ist die Stufenbezeichnung, die den Beginn der Listenbeschreibung anzeigt und dem Listennamen unmittelbar vorausgehen muss.
2. Der Listenname muss in der REPORT-Klausel in der Dateierklärung für die Datei, in welche die Liste geschrieben wird, angegeben sein.
3. Wenn in der Listenerklärung die GLOBAL-Klausel angegeben ist, dürfen die in den Klauseln der Listenerklärung und den Klauseln der Leistenerklärungen angesprochenen Datenfelder nicht in einer LOCAL-STORAGE SECTION definiert sein.
4. Innerhalb einer Listenerklärung dürfen nicht mehr als 31 Datennamen in einer CONTROL-Klausel angegeben sein.
5. Der Listenname identifiziert die Liste und muss daher eindeutig sein.

Die Klauseln der Listenerklärung werden anschließend beschrieben.

10.2.4 CODE-Klausel

Funktion

Die CODE-Klausel legt ein Zeichen zur Identifizierung der zu einer Liste gehörigen Druckzeilen fest. Dieses Zeichen wird unmittelbar am Anfang einer jeden Druckzeile der Liste eingefügt, darf aber nicht gedruckt werden. Es dient dazu, die Druckzeilen zweier oder mehrerer Listen, die vermischt oder aufeinanderfolgend in die Listendatei geschrieben wurden, zu trennen.

Format

```
CODE literal
```

Syntaxregeln

1. Das Literal muss ein alphanumerisches Literal sein. Zur Markierung der Liste verwendet der COBOL2000-Compiler nur das erste Zeichen des Literals.
2. Enthält eine Listenerklärung die CODE-Klausel, dann müssen diejenigen Listenerklärungen ebenfalls eine CODE-Klausel aufweisen, die über die REPORT-Klausel derselben Ausgabedatei zugeordnet sind.
3. Mit dem Literal ist das Zeichen verknüpft, das jeder Druckzeile vorangestellt wird.
4. Das Zeichen zur Markierung der Druckzeilen einer Liste wird am Zeilenanfang hinter dem Vorschubzeichen hinterlegt. Dieses Zeichen darf im Druckbild nicht erscheinen.

Allgemeine Regel

1. Die CODE-Klausel darf nicht angegeben werden, wenn die Liste sofort („online“) gedruckt wird oder der Listendatei SYSLST zugeordnet ist. Es muss dafür gesorgt werden, dass sich die Listen nicht überlappen, d. h. dass in der Zeit zwischen der INITIATE- und der TERMINATE-Anweisung einer Liste keine GENERATE-Anweisung einer anderen Liste ausgeführt wird.

10.2.5 CONTROL-Klausel

Funktion

Die CONTROL-Klausel definiert die Datenfelder, die als Gruppenwechseldatenfelder der Liste die Gruppenhierarchie und damit die hierarchische Strukturierung der Liste bestimmen.

Format

```
{CONTROL | CONTROLS} {IS | ARE} {{datenname-1}... | FINAL [datenname-1]...}
```

Syntaxregeln

1. In der CONTROL-Klausel dürfen nicht mehr als 31 Datennamen angegeben werden, wobei die Angabe FINAL mitzuzählen ist, falls sie angegeben wurde.
2. datenname-1... darf gekennzeichnet, aber nicht indiziert sein.
3. Einem Datennamen darf kein Datenfeld untergeordnet sein, dessen Größe in der OCCURS-Klausel als variabel definiert ist.
4. datenname-1... darf nicht in der REPORT SECTION definiert sein. Er muss entweder in der FILE SECTION, LOCAL-STORAGE SECTION oder WORKING-STORAGE SECTION definiert sein. Wenn sichergestellt ist, dass das aufgerufene Programm in der Zeit von der Initialisierung bis einschließlich der Beendigung der Listenerstellung ohne Unterbrechung im Arbeitsspeicher zur Verfügung steht, dann ist es auch zulässig, dass ein in der CONTROL-Klausel angegebener Datenname in der LINKAGE SECTION des gerufenen Programms definiert ist.
5. Das verwendete Datenfeld darf maximal 256 Zeichen lang sein.
6. Jeder datenname-1 muss ein anderes Datenfeld bezeichnen.

   Zwei verschiedene Wiederholungen von datenname-1 dürfen sich auch nicht auf Datenfelder beziehen, die mittels Redefinition denselben Speicherplatz belegen.
7. Ein Gruppenwechseldatenfeld ist ein in einer CONTROL-Klausel angegebenes Datenfeld, das bei Ausführung einer auf dieselbe Liste bezogenen GENERATE-Anweisung dahingehend untersucht wird, ob sich sein Wert gegenüber der zuletzt ausgeführten GENERATE-Anweisung (zur selben Liste) geändert hat.

   Aus der Feststellung einer solchen Wertänderung resultiert ein Gruppenwechsel („control break“), d.h. es werden bestimmte (unten beschriebene) Aktionen vor der Erstellung der durch die GENERATE-Anweisung angegebenen Postenleiste ausgeführt. Liegen bei der Ausführung der GENERATE-Anweisung Wertänderungen bei mehreren Gruppenwechseldatenfeldern vor, wird immer die hierarchisch höchstrangige Wertänderung bestimmt, auf die sich alle Gruppenwechsel-Aussagen (wie Gruppenwechsel und Gruppenwechselstufe) beziehen.
8. FINAL, datenname-1... legen die Gruppenhierarchie der Liste fest.
9. Die Datennamen kennzeichnen in der Reihe ihrer Angabe von links nach rechts in absteigender Form den Rang der Gruppenhierarchiestufen. Dem letzten Datennamen (ganz rechts) entspricht die niedrigste, dem vorletzten Datennamen die nächst höhere Hierarchiestufe, usw.
10. FINAL beschreibt den hierarchisch höchsten Gruppenwechsel. Der zugehörige Gruppenkopf wird bei der ersten GENERATE-Anweisung, der zugehörige Gruppenfuß bei der TERMINATE-Anweisung erzeugt.

Allgemeine Regeln

1. Die durch einen Gruppenwechsel implizierten Aktionen hängen davon ab, ob zu je einer Gruppenhierarchiestufe der Liste ein Gruppenkopf und/oder ein Gruppenfuß oder keine der beiden definiert sind. Sobald ein Gruppenwechsel eintritt, erstellt das Listenprogramm die folgenden Gruppenköpfe und Gruppenfüße (soweit sie vorhanden sind) in der Reihenfolge:
   a. Gruppenfuß der niedrigsten Stufe

b. Gruppenfuß der nächsthöheren Stufe

.
.
.

c. Gruppenfuß der für den Gruppenwechsel zuständigen Stufe
d. Gruppenkopf der für den Gruppenwechsel zuständigen Stufe
e. Gruppenkopf der nächstniedrigeren Stufe

.
.
.

f. Gruppenkopf der niedrigsten Stufe.
g. Anschließend wird vom Listenprogramm die Postenleiste infolge der GENERATE-Anweisung erstellt. Wenn z.B. für eine Liste die Gruppenwechseldatennamen mit JAHR, MONAT und TAG (in dieser Reihenfolge in der CONTROL-Klausel angeführt) gegeben und jeder dieser Datennamen mit je einem Gruppenkopf und einem Gruppenfuß gekoppelt ist, werden bei einem Gruppenwechsel für JAHR (Änderung des Feldinhaltes von JAHR zwischen zwei chronologisch aufeinanderfolgenden GENERATE-Anweisungen) die Rumpfleisten in folgender Reihenfolge gedruckt:

Gruppenfuß	für TAG
Gruppenfuß	für MONAT
Gruppenfuß	für JAHR
Gruppenkopf	für Jahr
Gruppenkopf	für MONAT
Gruppenkopf	für TAG

Postenleiste infolge der GENERATE-Anweisung.

2. Wenn zu Beginn der Rumpfleistenerstellung vom Listenprogramm erkannt wird, dass eine der Bedingungen für einen Seitenvorschub erfüllt ist, dann wird zuerst der Seitenfuß (wenn vorhanden), anschließend ein Seitenvorschub, dann der Seitenkopf (wenn ein solcher vorliegt) und schließlich die Rumpfleiste erzeugt.
3. Die Erstellung einer Postenleiste muss vom Programmierer mittels einer entsprechenden GENERATE-Anweisung (durch Angabe der Postenleiste) in der PROCEDURE DIVISION vom Listenprogramm gefordert werden. Alle übrigen Leisten, wie Listenkopf, Listenfuß, Seitenkopf, Seitenfuß, Gruppenköpfe und Gruppenfüße, werden vom Listenprogramm automatisch erstellt, sobald die Voraussetzungen dafür erfüllt sind.
4. Zur Feststellung eines Gruppenwechsels wird ohne Rücksicht auf die Beschreibungen der Gruppenwechseldatenfelder ein alphanumerischer Vergleich vorgenommen. Das bedeutet z.B., dass das Listenprogramm eine Wertänderung feststellt, wenn sich der Inhalt eines mit COMPUTATIONAL-3 beschriebenen Gruppenwechseldatenfeldes von sedezimal ’7F’ auf sedezimal ’7C’ ändert, obwohl beide Darstellungsvarianten die positive Zahl 7 ausdrücken.
5. Wenn die Listenerklärung die CONTROL-Klausel nicht enthält, dürfen und können für diese Liste keine Gruppenköpfe und Gruppenfüße definiert werden.

Beispiel 10-1

Ausschnitt aus einer Liste:

```
JANUAR 14  B10                   4  B    8.36
           B10                   1  C    9.00
EINKAEUFE & KOSTEN FUER 1-14     5     $17.36  $136.36
JANUAR 15  B10                   2  A   16.00
```

Die zugehörige Listenerklärung enthält die CONTROL-Klausel

```
CONTROLS ARE FINAL MONAT TAG
```

Bis auf die Druckzeile, die mit EINKAEUFE beginnt und die den Gruppenfuß zu TAG bildet, sind alle übrigen Druckzeilen aus dem obigen Listenausschnitt Postenleisten (selbe Listenerklärung). Der Gruppenfuß von TAG wurde infolge der Datumsänderung vom 14. auf den 15. Januar erzeugt.

10.2.6 GLOBAL-Klausel

Funktion

Die GLOBAL-Klausel kann nur innerhalb eines geschachtelten Programms verwendet werden. Sie legt fest, dass ein Listenname global ist. Auf einen globalen Namen kann das Programm, das ihn vereinbart, und jedes in diesem Programm direkt oder indirekt enthaltene Programm zugreifen.

Format

```
IS  GLOBAL
```

Formate für Dateierklärung siehe bei FILE SECTION.

Syntaxregel

1. Die GLOBAL-Klausel darf nur in Listenerklärungen angegeben werden:

Allgemeine Regeln

1. Ein Listenname, dessen Beschreibung eine GLOBAL-Klausel enthält, ist ein globaler Name. Alle einem globalen Namen untergeordneten Datennamen, die mit einem globale Namen in Verbindung stehen, sind globale Namen.
2. Auf einen globalen Namen darf jedes Programm zugreifen, das in dem Programm enthalten ist, das den globalen Namen beschreibt. Der globale Name braucht in dem Programm, das ihn referenziert, nicht nochmals beschrieben zu werden. Bei Referenzen auf gleiche Namen haben die lokalen Namen Vorrang (siehe "Bestimmung des gültigen Namens").
3. Die Sonderregister LINE-COUNTER und PAGE-COUNTER sind ebenfalls global.

10.2.7 PAGE LIMIT-Klausel

Funktion

Die PAGE LIMIT-Klausel gibt die Länge der Druckseite und deren vertikale Unterteilung in leistenspezifische Zonen an.

Format

```
PAGE [{LIMIT | LIMITS} {IS | ARE}] integer-p [{LINE | LINES}]
     [HEADING ganzzahl-h]
     [FIRST DETAIL ganzzahl-d]
     [LAST DETAIL ganzzahl-e]
     [FOOTING ganzzahl-f]
```

Syntaxregeln

1. ganzzahl-p gibt die maximal mögliche Anzahl der Druckzeilen einer Seite an.
2. ganzzahl-p darf 999 nicht übersteigen.
3. Die restlichen Ganzzahlen der PAGE LIMIT-Klausel stellen Zeilennummern dar. Da die Nummerierung der Zeilen einer Seite mit 1 beginnt, kann auch ganzzahl-p als Zeilennummer interpretiert werden.
4. Die Ganzzahlen der PAGE LIMIT-Klausel unterliegen folgenden Beziehungen: ganzzahl-h muss größer oder gleich 1 sein.

   ganzzahl-d muss größer oder gleich ganzzahl-h sein.
   ganzzahl-e muss größer oder gleich ganzzahl-d sein.
   ganzzahl-f muss größer oder gleich ganzzahl-e sein.
   ganzzahl-p muss größer oder gleich ganzzahl-f sein.
5. ganzzahl-h der HEADING-Angabe stellt die erste mögliche Druckzeile einer Seite dar.
6. ganzzahl-h darf 999 nicht übersteigen.
7. ganzzahl-p der LIMIT-Angabe fixiert die letzte zulässige Druckzeile einer Seite.
8. ganzzahl-d der FIRST-DETAIL-Angabe legt die niedrigste erlaubte Zeilennummer für alle Rumpfleisten fest.
9. ganzzahl-d darf 999 nicht übersteigen.
10. ganzzahl-e der LAST DETAIL-Angabe bestimmt die höchste zulässige Zeilennummer für alle Postenleisten und Gruppenköpfe.
11. ganzzahl-e darf 999 nicht übersteigen.
12. Innerhalb einer Listenerklärung dürfen nicht mehr als 127 Postenleisten (DETAIL) angegeben sein.
13. ganzzahl-f der FOOTING-Angabe bildet für alle Gruppenfüße die höchste erlaubte Zeilennummer.
14. Innerhalb einer Listenerklärung dürfen nicht mehr als 31 Gruppenfüße angegeben werden.
15. ganzzahl-f darf 999 nicht übersteigen.
16. Wenn die PAGE LIMIT-Klausel zwar vorliegt, aber nicht alle der in ihr möglichen Angaben gemacht wurden, dann werden diese Angaben intern mit folgenden Wertzuordnungen gemäß der folgenden tabellarischen Übersicht vorgenommen:

ganzzahl	angenommener Wert, falls nicht angegeben
ganzzahl-h (HEADING-Angabe)	1
ganzzahl-d (FIRST DETAIL-Angabe)	Wert von ganzzahl-h der HEADING-Angabe

ganzzahl-e (LAST DETAIL-Angabe)	Wert von ganzzahl-f der FOOTING-Angabe
ganzzahl-f (FOOTING-Angabe)	Wert von ganzzahl-p der LIMIT-Angabe

17. Wird die PAGE LIMIT-Klausel nicht angegeben, dann legt der Compiler für alle Ganzzahlen der PAGE LIMIT-Klausel folgende Werte fest:

ganzzahl	**angenommener Wert**
ganzzahl-p (LIMIT-Angabe)	50
ganzzahl-h (HEADING-Angabe)	1
ganzzahl-d (FIRST DETAIL-Angabe)	1
ganzzahl-e (LAST DETAIL-Angabe)	48
ganzzahl-f (FOOTING-Angabe)	48

Allgemeine Regeln

1. Das Listenprogramm verwendet die Angaben der PAGE LIMIT-Klausel zur Aufteilung der Seite in Zonen. Nur Leisten bestimmter Typen dürfen in den einzelnen Zonen
   gedruckt werden. Die Seitenzonen für die einzelnen Leistentypen sind folgende:
   a. Der Listenkopf kann sich von der Zeile mit der Nummer ganzzahl-h bis einschließlich der Zeile mit ganzzahl-p erstrecken, wenn seine Beschreibung NEXT GROUP NEXT PAGE enthält. Andernfalls darf sich der Listenkopf nicht über die Zeile mit ganzzahl-d minus 1 hinaus erstrecken, was eine explizite Angabe von ganzzahl-d erfordert (siehe unter b).
   b. Der Seitenkopf muss in der Zone von ganzzahl-h bis einschließlich ganzzahl-d minus 1 liegen. Diese Zone ist nicht existent, falls vom Compiler der Wert von ganzzahl-d festgelegt wird. Es ist also notwendig, ganzzahl-d explizit in der PAGE LIMIT-Klausel anzugeben, wenn ein Seitenkopf oder ein Listenkopf, der keine Seite für sich beansprucht, gedruckt werden soll.
   c. Postenleisten und Gruppenköpfe dürfen nur in der Zone von ganzzahl-d bis ganzzahl-e einschließlich dieser Grenzen gedruckt werden.
   d. Die für den Ausdruck von Gruppenfüßen zulässige Zone erstreckt sich von ganzzahl-d bis einschließlich ganzzahl-f.
   e. Der Seitenfuß darf nur in der Zone von ganzzahl-f +1 bis einschließlich ganzzahl-p gedruckt werden. Diese Zone gibt es nicht, wenn die PAGE LIMIT-Klausel ohne ganzzahl-f-Angabe vorliegt. Eine explizite Angabe von ganzzahl-f ist daher in diesem Fall unerlässlich, wenn ein Seitenfuß gedruckt werden soll.
   f. Wenn dem Listenfuß durch LINE NEXT PAGE eine ganze Seite zugeordnet ist, darf er sich von ganzzahl-h bis einschließlich ganzzahl-p erstrecken. Ansonsten gilt für den Listenfuß die Regel e).
2. NEXT GROUP- und LINE-Klauseln der Leistenbeschreibungen dürfen nicht zum Widerspruch mit der PAGE LIMIT-Klausel führen, d.h. die Zeilen einer Leiste müssen innerhalb der zugeordneten Zone liegen.
3. In Tabelle 39 wird schematisch die Aufteilung der Seite in Zonen für den Fall dargestellt, dass alle Ganzzahlen der PAGE LIMIT-Klausel verschiedene Werte aufweisen mit

   ganzzahl-h < ganzzahl-d < ganzzahl-e < ganzzahl-f < ganzzahl-p

   Dies ist auch die Voraussetzung für die nachfolgenden Regeln zum Gebrauch dieser Zonen:

ganzzahl	Zonen einer Seite und mögliche Angaben				
	Zone 1 REPORT HEADING und REPORT FOOTING	Zone 2 PAGE HEADING und (evtl.) REPORT HEADING	Zone 3 DETAIL und CONTROL	Zone 4 CONTROL FOOTING	Zone 5 PAGE FOOTING und (evtl.) REPORT FOOTING
ganzzahl-h					
ganzzahl-h +1 . . . ganzzahl-d −1					
ganzzahl-d					
ganzzahl-d +1 . . . ganzzahl-e −1					
ganzzahl-e					
ganzzahl-e +1 . . . ganzzahl-f −1					
ganzzahl-f					
ganzzahl-f +1 . . . ganzzahl-p −1					
ganzzahl-p					

Tabelle 39: Schematische Darstellung der Seitenaufteilung in Zonen

Zone 1:

Reichweite: Von ganzzahl-h bis einschließlich ganzzahl-p.

Inhalt: Listenkopf oder Listenfuß

Regeln: Wenn die Listenkopf-Beschreibung die NEXT GROUP-Klausel mit der Angabe NEXT PAGE enthält, dann wird für den Listenkopf eine ganze Druckseite reserviert, d.h. keine andere Leiste wird auf dieser Seite gedruckt. Ob der Listenkopf beim Ausdrucken die ganze Zone oder irgend einen Teil davon belegt, wird über die LINE-Klausel der Listenkopf-Beschreibung entschieden.

Falls in der Listenfuß-Beschreibung die erste (oder einzige) LINE-Klausel die Angabe NEXT PAGE aufweist, dann wird für den Listenfuß eine ganze Druckseite reserviert, d.h. es gelten dieselben Aussagen wie für den Listenkopf.

Zone 2:

Reichweite: Von ganzzahl-h bis einschließlich ganzzahl-d minus 1.

Inhalt: Listenkopf und Seitenkopf, oder Seitenkopf allein

Regeln: In Zone 2 wird der Listenkopf nur dann geschrieben, wenn seine Leistenerklärung nicht die Klausel NEXT GROUP PAGE enthält. Da der Listenkopf pro Liste nur einmal gedruckt wird, darf ihn nur Zone 2 der ersten Seite enthalten.

Mit Ausnahme jener Seiten, die nur für den Listenkopf und Listenfuß reserviert sind, muss der Seitenkopf auf jeder Seite in Zone 2 gedruckt werden, sofern es ihn gibt.

Wenn die erste Seite sowohl mit dem Listenkopf als auch mit dem Seitenkopf versehen ist, so muss der Seitenkopf nach dem Listenkopf stehen. LINE- und NEXT GROUP-Klausel der beiden Listenerklärungen dürfen dem nicht widersprechen.

Zone 3:

Reichweite: Von ganzzahl-d bis einschließlich ganzzahl-e.

Inhalt: Rumpfleiste

Regeln: Wie aus der Reichweite abzulesen ist, reicht Zone 4 im Allgemeinen nach unten über die Zone 3 hinaus.

In die Zone 3 bzw. in den von ihr überlappten Teil der Zone 4 darf jede beliebige Rumpfleiste geschrieben werden. Der restliche Teil von Zone 4 darf nur von Gruppenfüßen belegt werden.

Zone 4:

Reichweite: Von ganzzahl-d bis einschließlich ganzzahl-f.

Inhalt: Rumpfleiste

Regeln: Wie aus der Reichweite abzulesen ist, reicht Zone 4 im Allgemeinen nach unten über die Zone 3 hinaus.

In die Zone 3 bzw. in den von ihr überlappten Teil der Zone 4 darf jede beliebige Rumpfleiste geschrieben werden. Der restliche Teil von Zone 4 darf nur von Gruppenfüßen belegt werden.

Für die NEXT GROUP-Klausel einer Rumpfleiste ist unabhängig von ihrem Typ Zone 4 zuständig.

Zone 5:

Reichweite: Von ganzzahl-f+1 bis einschließlich ganzzahl-p.

Inhalt: Listenfuß und Seitenfuß.

Regeln: Wenn die erste LINE-Klausel aus der Listenerklärung für den Listenfuß nicht die Angabe NEXT PAGE enthält, wird der Listenfuß in Zone 5 der letzten Seite der Liste gedruckt.

Der Seitenfuß einer Liste wird immer in Zone 5 geschrieben.

Wenn sowohl der Seitenfuß als auch der Listenfuß in Zone 5 der letzten Seite geschrieben werden sollen, dann muss der Seitenkopf vor dem Listenfuß liegen. Dieser Regel dürfen die LINE-Klauseln der beiden Leisten nicht widersprechen.

10.2.8 Leistenerklärung

Funktion

Die Leistenerklärung beschreibt und definiert das Format, den Typ und die Eigenschaften einer Leiste als eine Folge von elementaren und mit Informationen gekoppelten Feldern, die - in Zeilen und Spalten organisiert - die Druckzeile(n) der Leiste formieren.

Eine Vorpositionierung für die als nächste zu druckende Leiste ist durch eine NEXT GROUP-Angabe möglich.

Über den Typ der Leiste sind sowohl Anforderungen an die Beschreibung der Leiste, als auch an das Listenprogramm bei der Erzeugung der Leiste gestellt. So hängt die Seitenpositionierung einer Leiste auch vom Typ der Leiste ab. Ähnliches gilt für den Zeitpunkt der Leistenerstellung. Summationen können z.B. nur in Gruppenfüßen definiert werden. Durchgeführt werden sie je nach Art der Summation unmittelbar vor Erstellung von Postenleisten oder der betroffenen Gruppenfüße.

Format

```
01 [datenname-1]
    TYPE-Klausel
   [LINE-Klausel]
   [NEXT GROUP-Klausel]

02 [datenname-2]
   [LINE-Klausel]
   [COLUMN-Klausel]
   [GROUP INDICATE-Klausel]
   [PICTURE-Klausel]
   [BLANK WHEN ZERO-Klausel]
   [JUSTIFIED RIGHT-Klausel]
   [SIGN-Klausel]
   [USAGE-Klausel]

   {SOURCE-Klausel | SUM-Klausel | VALUE-Klausel}
```

Syntaxregeln

1. Die Leistenerklärung muss mit einem Eintrag auf Stufe 01 beginnen. Mindestens ein Eintrag mit Stufe 02 muss folgen.
2. Der Datenname muss unmittelbar der Stufennummer folgen. Die Klauseln können in beliebiger Reihenfolge angegeben werden.
3. Eine Liste besteht aus einer Folge von Leisten. Bis zu sieben verschiedene Leistentypen (siehe „TYPE-Klausel") können in der Liste vorkommen.

   Eine Leiste setzt sich ihrer Beschreibung nach aus einem oder mehreren elementaren Feldern zusammen, wobei jedes Feld, das mit einer COLUMN-Klausel versehen ist (druckfähiges Feld) einer Zeile zugeordnet sein muss. Leisten ohne druckfähige Felder sind nicht druckbar. Eine druckfähige Leiste ist eine Einheit, die sich aus einer oder mehreren Zeilen zusammensetzt.

   Mit Ausnahme des Listenkopfes und des Listenfußes kann eine Leiste im Laufe der Listenerstellung wiederholt erzeugt werden, wobei sich am Leistenformat und an den konstanten Informationsinhalten keine Änderung ergeben darf.
4. Innerhalb einer Listenerklärung dürfen nicht mehr als 127 Leistenerklärungen angegeben sein.

5. Der Datenname, der unmittelbar auf die Stufennummer 01 folgt, ist der Name der Leiste. Seine Angabe ist erforderlich, wenn die Leiste in einer GENERATE-Anweisung (Postenleiste), einer USE BEFORE REPORTING-Anweisung, einer UPON-Angabe einer SUM-Klausel (Postenleiste) direkt angesprochen werden muss oder zur Kennzeichnung verwendet wird.
6. Die TYPE-Klausel spezifiziert die Leistenart, durch die das Listenprogramm entscheidet, wann und wo diese Leiste erstellt wird.
7. Die LINE-Klausel ordnet den in ihrem Gültigkeitsbereich angegebenen druckfähigen Feldern eine Zeile der aktuellen oder der nächsten Druckseite zu. Diese erste LINE-Klausel der Leistenerklärung ist daher für die Positionierung der Leiste maßgeblich.
   Der Gültigkeitsbereich erstreckt sich bis zur nächsten LINE-Klausel oder bis zum Ende der Leistenerklärung.
8. Die NEXT GROUP-Klausel bewirkt, dass am Ende der Leistenerstellung für die als nächste zu erzeugende Leiste eine Positionierung auf die Zeilennummer aus den Angaben der NEXT GROUP-Klausel vorgenommen wird.

   Die COLUMN-, GROUP INDICATE-, BLANK-, JUSTIFIED- und PICTURE-Klauseln definieren Lage und Format des druckfähigen Datenfeldes bezüglich einer Druckzeile der Leiste.
9. SOURCE-, SUM- und VALUE-Klauseln verknüpfen die Datenfelder mit Informationen. Darüberhinaus definiert die SUM-Klausel einen Summenzähler, den das Listenprogramm automatisch aktualisiert.
10. Jedes druckfähige Feld muss durch eine LINE-Klausel abgedeckt sein. Eine solche LINE-Klausel stellt eine Druckzeile dar.
11. Die PICTURE-Klausel darf für ein Datenelement nur weggelassen werden, wenn eine VALUE-Klausel mit einem alphanumerischen oder hexadezimal-alphanumerischen Literal angegeben ist. Der Compiler nimmt dann eine PICTURE-Klausel PICTURE X(länge) an, wobei länge die Anzahl der durch das Literal dargestellten Zeichen ist.

Allgemeine Regeln

1. Der Name einer Leiste kann in der REPORT SECTION und in der PROCEDURE DIVISION angegeben werden. Nur Namen von Gruppenfüßen und Postenleisten können in der REPORT SECTION angesprochen werden. Während der Name einer Postenleiste in einer UPON-Angabe einer SUM-Klausel angegeben werden kann, darf der Name eines Gruppenfußes nur als Kennzeichen eines Summenzählers verwendet werden. In der PROCEDURE DIVISION darf der Name einer Leiste nur in zwei Fällen angegeben werden:
   a. Der Name einer Postenleiste darf in einer GENERATE-Anweisung angegeben werden.
   b. In einer USE BEFORE REPORTING-Anweisung darf mit Ausnahme der Namen von Postenleisten jeder Leistenname verwendet werden.

   Als Kennzeichner eines Leistennamens ist nur der zugehörige Listenname zulässig.
2. Die Angabe eines Datennamens unmittelbar nach der Stufennummer 02 einer Leisten-beschreibung hat nur dann einen Sinn, wenn diese Datenfeldbeschreibung eine SUM-Klausel einschließt. In einem solchen Falle wird der Datenname als Name des infolge der SUM-Klausel angelegten Summenzählers interpretiert (siehe „SUM-Klausel"). Der Name eines Summenzählers kann wieder in einer SUM-Klausel angeführt werden. Wird der Datenname auch dann angegeben, wenn keine SUM-Klausel vorliegt, d.h. wird der Datenname als Name des Feldes definiert, so darf er grundsätzlich nicht verwendet werden.

   Zur Kennzeichnung von Summenzählern können nur Leistennamen und/oder Listennamen verwendet werden.

Zusammenfassung der Klauseln aus der Leistenerklärung

In Tabelle 40 sind die Klauseln der Leistenerklärung mit Kurzbeschreibungen aufgeführt.

Die BLANK WHEN ZERO-, JUSTIFIED-, PICTURE-, USAGE- und VALUE-Klauseln werden auch in anderen Kapiteln des Datenteils verwendet, d.h. sie werden hier als bekannt vorausgesetzt (siehe die entsprechenden Beschreibungen).

Die anderen Klauseln aus der Listenerklärung sind anschließend detailliert in alphabetischer Reihenfolge der englischen Bezeichnungen beschrieben.

Klausel	Kurzbeschreibung
BLANK WHEN ZERO-Klausel	Sie bewirkt, dass nur ein von Null verschiedener Inhalt des Feldes ausgedruckt wird.
COLUMN-Klausel	Sie gibt die Anfangsposition (Spaltennummer) des Feldes auf der Druckzeile an. Damit ist das Feld als druckfähig definiert.
GROUP INDICATE-Klausel	Sie zeigt an, dass das druckfähige Feld nur dann gedruckt werden darf, wenn die Postenleiste, zu der sie gehört, zum ersten Mal in der Liste nach einem Seitenvorschub oder Gruppenwechsel erstellt wird.
JUSTIFIED-Klausel	Sie gibt explizit die Datenausrichtung (Positionierung) im zugehörigen Feld an.
LINE-Klausel	Sie ordnet den druckfähigen Feldern ihres Wirkungsbereiches eine Zeile einer Druckseite zu.
NEXT GROUP-Klausel	Sie bewirkt eine Vorpositionierung für die eventuell als nächste zu druckende Leiste.
PICTURE-Klausel	Sie gibt die charakteristischen Eigenschaften eines Datenfeldes oder eines Datenfeldes und Summenzählers an.
SIGN-Klausel	Sie beschreibt die Position und die Darstellung des Rechenvorzeichens für numerische Datenfelder.
SOURCE-Klausel	Sie ordnet einem Datenfeld ein Sendefeld als Informationsquelle zu.
SUM-Klausel	Sie gibt an, wie und welche Summanden im Summenzähler aufsummiert werden sollen. Der Summenzähler wird automatisch angelegt. Außerdem ordnet sie dem Datenfeld den Summenzähler als Informationsquelle zu.
TYPE-Klausel	Durch sie wird der Leiste ein spezieller Typ zugeschrieben.
USAGE-Klausel	Sie legt fest, in welchem Datenformat ein Datenelement abgespeichert wird.
VALUE-Klausel	Sie gibt einen konstanten Wert für ein druckfähiges Feld an.

Tabelle 40: Klauseln der Leistenerklärung

10.2.9 COLUMN-Klausel

Funktion

Die COLUMN-Klausel erklärt ein Datenfeld als druckfähiges Feld, indem sie ihm mittels der Spaltennummer (ganzzahl) eine Position (Feldanfang) bezüglich der Druckzeile zuweist.

Format

```
COLUMN ganzzahl
```

Syntaxregeln

1. ganzzahl muss als vorzeichenlose ganze Zahl im Bereich 1 ganzzahl 251 angegeben sein.
2. Die Spaltennummer (ganzzahl) gibt die Position an, die die erste (ganz linke) Zeichenposition des druckfähigen Feldes auf der Druckzeile einnehmen soll. Die erste bedruckbare Zeichenposition der Druckzeile hat die Spaltennummer 1. Die höchste zulässige Spaltennummer hängt vom verwendeten Druckertyp ab.

Allgemeine Regeln

1. Wenn die Beschreibung eines Datenfeldes, die neben der PICTURE-Klausel auch eine der SOURCE-, SUM- oder VALUE-Klauseln enthalten muss, auch noch die COLUMN-Klausel einschließt, so ist ein druckfähiges Feld definiert.
2. Die Beschreibung eines druckfähigen Feldes muss entweder selbst eine LINE-Klausel enthalten oder es muss ihr (in derselben Leistenerklärung) eine LINE-Klausel voraus-gehen. Das druckfähige Feld wird in jener Druckzeile gedruckt, die durch die LINE-Klausel angegeben ist.
3. Innerhalb des Gültigkeitsbereiches einer LINE-Klausel (bis zur nächsten LINE-Klausel ausschließlich oder bis zum Ende der Leistenerklärung) müssen die druckfähigen Felder in der Reihenfolge aufsteigender Spaltennummern definiert sein. Die druckfähigen Felder werden auf die Zeile in der Reihenfolge ihrer Definitionen gedruckt.
4. Die druckfähigen Felder einer Zeile dürfen nicht so definiert werden, dass eine Überlappung eintritt (betrifft COLUMN- und PICTURE-Klausel).
5. Unmittelbar bevor die druckfähigen Felder einer Zeile gedruckt werden, d.h. die Druckzeile in die Listendatei geschrieben wird, werden sie automatisch durch implizite MOVE-Anweisungen belegt. Je nachdem, ob die Beschreibung des Empfangsfeldes (=druckfähiges Feld) die SOURCE-, VALUE- oder SUM-Klausel enthält, ist das Sendefeld der Bezeichner aus der SOURCE-Klausel, das Literal aus der VALUE-Klausel oder der Summenzähler, der für die SUM-Klausel angelegt wurde.
6. Das Listenprogramm sorgt dafür, dass alle Positionen der Druckzeilen, die nicht durch druckfähige Felder abgedeckt sind, Leerzeichen erhalten.

Beispiel 10-2

Wenn in einer Leistenerklärung ein elementares Feld mit

```
02 LINE 3 COLUMN 30 PIC A(12) VALUE IS "AUFWENDUNGEN"
```

beschrieben ist, dann wird bei der Erstellung der Leiste die Zeichenfolge AUFWENDUNGEN in Zeile 3 ab Spalte 30 auf der aktuellen Druckseite geschrieben.

10.2.10 GROUP INDICATE-Klausel

Funktion

Die GROUP INDICATE-Klausel zeigt dem Listenprogramm an, dass das betroffene druckfähige Feld einer Postenleiste nur dann gedruckt werden soll, wenn die Postenleiste erstmals auf einer Seite der Liste oder erstmals nach einem Gruppenwechsel geschrieben wird.

Format

```
GROUP INDICATE
```

Syntaxregeln

1. Die GROUP INDICATE-Klausel darf nur in einer Postenleiste verwendet werden. Die Feldbeschreibung, die die GROUP INDICATE-Klausel enthält, muss auch eine COLUMN-Klausel aufweisen.
2. Die GROUP INDICATE-Klausel veranlasst das Listenprogramm, nur für die Fälle, dass die betroffene Postenleiste erstmals
   a. in der Liste
   b. nach einem Seitenvorschub oder
   c. nach einem Gruppenwechsel

   erstellt wird, die Belegung des druckfähigen Feldes entsprechend der spezifizierten SOURCE- oder VALUE-Klausel vorzunehmen. In allen übrigen Fällen wird das druckfähige Feld mit Leerzeichen belegt, d.h. es wird zugeordnete Information im Druckbild unterdrückt.

Beispiel 10-3

a) Angaben DATA DIVISION

```
          .
          .
          .
     CONTROLS ARE FINAL, MONAT, TAG
          .
          .
          .
 01  EINZEL-ZEILE LINE PLUS 1 TYPE DETAIL.
     02  COLUMN 2 GROUP INDICATE PIC A(9)
         SOURCE MONATSNAME (MONAT).
     02  COLUMN 13 GROUP INDICATE PIC 99 SOURCE TAG.
         .
         .
         .
```

b) Listenauszug

```
         .
         .
         .
     JANUAR 15 B11 16 A 20.00
               B11  4 B 13.45
               B11 20 D 35.40
```

Der obige Listenausschnitt umfasst drei unmittelbar nach einem Gruppenwechsel (Inhaltsänderung von TAG) erstellte Postenleisten derselben Leistendefinition. Die aktuellen Inhalte JANUAR und 15 des Sendefeldes (SOURCE-Felder) für die ersten zwei druckfähigen Felder der aus einer Zeile bestehenden Postenleiste erscheinen daher nur in der ersten Zeile (Postenleiste).

10.2.11 LINE-Klausel

Funktion

Die LINE-Klausel ordnet den in ihrem Gültigkeitsbereich definierten druckfähigen Feldern (siehe „COLUMN-Klausel") jeweils diejenige Zeile einer Seite der Liste zu, auf welche sie gedruckt werden. In spezieller Anwendung dient sie nur zum Seitenvorschub.

Format

```
LINE NUMBER IS {ganzzahl-1 | PLUS ganzzahl-2 | NEXT PAGE}
```

Syntaxregeln

1. ganzzahl-1 und ganzzahl-2 dürfen nur als vorzeichenlose ganze Zahlen angegeben werden. ganzzahl-1 muss größer oder gleich 1 sein, ganzzahl-2 muss größer oder gleich 0 sein. Der Wert von ganzzahl-1 oder ganzzahl-2 darf 999 nicht überschreiten.
2. Als absolut wird eine LINE-Klausel der Alternative mit ganzzahl-1 bezeichnet. ganzzahl-1 ist als Zeilennummer zu interpretieren. Bei der Erstellung der betroffenen Leiste gibt ganzzahl-1 also jene Zeile der aktuellen Druckseite der Liste an, auf welche die zu der LINE-Klausel gehörigen und daher als Zeile organisierten druckfähigen Felder letztlich gedruckt werden. So definiert eine absolute LINE-Klausel immer eine Druckzeile.
3. Als relativ gilt eine LINE-Klausel, die mit der Alternative PLUS ganzzahl-2 gebildet wird. Die Zeile, auf welche die zu einer relativen LINE-Klausel gehörigen druckfähigen Felder in der Zeit, in der die betroffene Leiste erzeugt wird, gedruckt werden, wird relativ zu der zuletzt vorgenommenen vertikalen Positionierung bestimmt. Dazu wird die Summe aus der Nummer der Zeile, auf die zuletzt positioniert wurde (siehe „NEXTGROUP-Klausel" und „LINE-COUNTER-Sonderregister"), und der ganzzahl-2 gebildet. Die Summe gibt die Nummer der aktuellen Druckzeile wieder.

   Abweichungen von den obigen Regeln gibt es nur für spezielle Leistentypen, wenn die relative LINE-Klausel für die erste Zeile der Leiste verwendet wurde. Diese Abweichungen werden an geeigneter Stelle angeführt.
4. Eine LINE-Klausel mit der Angabe NEXT PAGE bewirkt, dass die zugehörige Leiste auf eine freie Seite (meistens die nächste Seite) gedruckt wird. Der Vorschub findet also noch vor dem Druck der Leiste statt.

   Ob eine LINE-Klausel mit der Angabe NEXT PAGE noch zusätzlich eine Druckzeile bestimmt, kann den nachfolgenden Regeln entnommen werden.

Allgemeine Regeln

Die nachfolgenden Programmierregeln werden in zwei Kategorien - die allgemeinen Regeln und die Regeln für Leistentypen - eingeteilt. Die Regeln der letzteren Kategorie beschreiben die Verwendung der LINE-Klausel in der Beschreibung der verschiedenen Leistentypen.

Die Abkürzungen, die in den Regeln für Leistentypen verwendet werden, haben folgende Bedeutung:

Abkürzung	Bedeutung
A	eine oder mehrere absolute LINE-Klauseln
R	eine oder mehrere relative LINE-Klausel
NP	neine LINE-Klausel mit NEXT PAGE-Angabe

1. Ein druckfähiges Feld einer Leiste, das auf eine Zeile einer Seite der Liste gedruckt werden soll, muss entweder selbst in seiner Beschreibung eine LINE-Klausel enthalten oder seiner Beschreibung muss innerhalb der zugehörigen Leistenerklärungen eine LINE-Klausel vorausgehen. Umgekehrt werden alle druckfähigen Felder, die in der entsprechenden Leistenerklärung auf die LINE-Klausel bis zur nächsten

LINE-Klausel oder dem Ende der Leistenerklärung folgen, auf die durch die LINE-Klausel bestimmte Zeile gedruckt. Nachfolgende druckfähige Felder aus der Leistenerklärung werden analog zusammengefasst in Zeilen gedruckt.

2. Die vertikalen Positionierungsparameter (LINE- und NEXT PAGE-Klauseln) müssen für die einzelnen Leistentypen so gewählt werden, dass diese Leisten innerhalb der für sie vorgesehenen Seitenzonen gedruckt werden können (siehe auch „PAGELIMIT-Klau-sel" und „TYPE-Klausel"). Eine Fortsetzung der Leiste in der für sie spezifischen Zone einer anderen Seite ist nicht möglich.

   Wenn eine Rumpfleiste nur deswegen keinen Platz in ihrer spezifischen Zone findet, weil ein Teil dieser Zone nicht mehr zur Verfügung steht, dann wird sie als Ganzes in die gleiche Zone der nächsten Seite gedruckt, nachdem zuerst der Seitenfuß auf der alten Seite und der Seitenkopf auf der neuen Seite erstellt wurde.
3. Wenn in einer Leistenerklärung absolute LINE-Klauseln (eine oder mehrere) enthalten sind, dann müssen sie alle
   a. vor der ersten (falls vorhanden) relativen LINE-Klausel,
   b. sowie in der Reihenfolge aufsteigender Ganzzahlen angegeben sein.
4. Die LINE-Klausel mit der Angabe NEXT PAGE darf nur in einer Leistenerklärung (01-Stufe) zur Positionierung auf die nächste Seite angegeben werden.

Syntaxregeln für Leistentypen

1. Listenkopf

   Die folgenden Reihenfolgen von LINE-Klauseln dürfen in der Leistenbeschreibung des Listenkopfes verwendet werden:

   ```
   {A | A R}
   ```
2. Seitenkopf

   In der Leistenerklärung des Seitenkopfes sind lediglich folgende Reihenfolgen von LINE-Klauseln zulässig:

   ```
   {A  | A R | R}
   ```

   Wenn die erste LINE-Klausel aus der Beschreibung des Seitenkopfes relativ ist, dann wird im Allgemeinen die erste Zeile der Leiste relativ zu der Zonengrenze ganzzahl-h der PAGE LIMIT-Klausel (siehe „PAGE LIMIT-Klausel") gedruckt. Nur wenn der Listenkopf auf derselben Seite gedruckt wurde, ergibt sich insofern eine Abweichung, als die erste Zeile der Leiste relativ zu der vertikalen Positionierung, die aus der Erstellung des Listenkopfes resultiert, gedruckt wird.
3. Rumpfleisten

   Als Rumpfleisten werden Postenleisten, Gruppenköpfe und Gruppenfüße bezeichnet.
   a. Die Beschreibung einer Rumpfleiste darf LINE-Klauseln in einer der folgenden Reihenfolgen enthalten:

      ```
      {NP | NP A  | NP A R | NP R | A | A R | R}
      ```
   b. Die LINE-Klausel mit der Angabe NEXT PAGE der Reihenfolge NP A oder NP A R dient nicht zur Bestimmung einer Druckzeile, sondern nur zum Seitenvorschub. Es muss also die erste absolute LINE-Klausel in oder vor der Beschreibung des ersten druckfähigen Feldes angegeben sein.
   c. Die LINE-Klausel mit der Angabe NEXT PAGE als einzige LINE-Klausel oder als eine mit der Reihenfolge NP R impliziert neben dem Seitenvorschub auch die vertikale Positionierung der ersten Zeile der Leiste.
   d. Die LINE-Klausel mit der Angabe NEXT PAGE in der Beschreibung einer Rumpfleiste signalisiert dem Listenprogramm, die Rumpfleiste auf die nächste noch nicht durch eine Rumpfleiste belegte Seite zu drucken. So kann die Situation eintreten, dass kein Seitenvorschub ausgeführt wird. Dies ist sicher der Fall, wenn die Rumpfleiste als erste Rumpfleiste in der Liste gedruckt wird.
   e. Wenn eine Rumpfleiste mit der LINE-Klausel-Reihenfolge NP, NP R oder R als erste Rumpfleiste auf eine Seite gedruckt werden soll, dann wird die erste Zeile der Rumpfleiste auf die Zeile mit der Nummer

ganzzahl-d aus der PAGE-Klausel
gedruckt. Nun ist es allerdings möglich, dass z.B. durch die zuletzt gedruckte Rumpfleiste infolge einer NEXT GROUP-Klausel (siehe "NEXT GROUP-Klausel") schon eine Positionierung über die Zonengrenze ganzzahl-d hinaus vorliegt. In diesem Falle wird die erste Zeile der Rumpfleiste auf die Zeile gedruckt, die unmittelbar auf die vorliegende Positionierung folgt.

f. Wenn eine Rumpfleiste, deren erste LINE-Klausel relativ ist, nicht als erste Rumpfleiste einer Seite gedruckt werden soll, dann wird die vorliegende Positionierung (Zeilennummer) um die Zeilenzahl, die durch ganzzahl-2 der ersten LINE-Klausel angegeben ist, vorgeschoben. Auf diese neue Position wird die erste Zeile der Rumpfleiste gedruckt.

4. Seitenfuß

   Die Angabe der LINE-Klausel in der Beschreibung des Seitenfußes ist nur in den zwei folgenden Möglichkeiten zulässig:

   ```
   {A | A R}
   ```

5. Listenfuß

   a. Folgende Arten von LINE-Klausel-Folgen sind in der Beschreibung des Listenfußes realisierbar:

      ```
      {NP A |NP A R| A | A R | R}
      ```

   b. Die LINE-Klausel mit der Angabe NEXT PAGE dient einzig und allein

      zum Seitenvorschub. Es muss daher die erste absolute LINE-Klausel in oder vor der Beschreibung des ersten druckfähigen Feldes auftreten, um die Positionierung der ersten Zeile der Leiste zu ermöglichen.

      Beispiel 10-4

      ```
      01  DETAIL-GRUPPE TYPE DETAIL LINE NEXT PAGE.
          02  LINE 10 COLUMN 1  PIC X(10) VALUE "1. ELEMENT".
          02          COLUMN 15 PIC X(4)  SOURCE KARTEN-FELD-1.
          02  LINE 12 COLUMN 1  PIC X(10) VALUE "2. ELEMENT".
          02          COLUMN 15 PIC 9(5)  SOURCE-ARB-FELD-1.
      ```

      Die aus zwei Zeilen sich zusammensetzende Postenleiste wird auf die Zeilen 10 und 12 einer noch nicht von anderen Rumpfleisten belegten Seite gedruckt, da die LINE-Klausel-Folge NP A vorliegt.

      Beispiel 10-5

      ```
      01  DETAIL-ZEILE LINE PLUS 1 TYPE DETAIL.
          02  COLUMN 2 GROUP INDICATE PIC A(9) SOURCE FELD-NR-1.
      ```

      Dieses Beispiel zeigt eine Postenleiste, deren Beschreibung nur eine relative LINE-Klausel enthält. Die Postenleiste wird auf die Zeile gedruckt, die sich aus der vorliegenden Position durch den Vorschub um eine Zeile ergibt. Für den Fall, dass die Postenleiste als erste Rumpfleiste der Seite gedruckt werden soll und die vorliegende Positionierung die Zonengrenze ganzzahl-d (FIRST DETAIL-Angabe der PAGE LIMIT-Klausel) nicht überschritten hat, wird die Postenleiste auf die Zeile mit der Nummer ganzzahl-d gedruckt.

10.2.12 NEXT GROUP-Klausel

Funktion

Die NEXT GROUP-Klausel gibt dem Listenprogramm an, auf welche Zeile einer Seite es nach dem Druck der letzten Zeile der die NEXT GROUP-Klausel enthaltenden Leiste positionieren muss. Damit bewirkt sie eine Vorpositionierung für die chronologisch anschließend zu druckende Leiste (Mindestabstand zur nächsten Leiste).

Format

```
NEXT GROUP IS { ganzzahl-1 | PLUS ganzzahl-2 | NEXT PAGE}
```

Syntaxregeln

1. ganzzahl-1 und ganzzahl-2 müssen als vorzeichenlose ganze Zahlen, die größer oder gleich 1 sind, angegeben werden. Der Wert darf 999 nicht überschreiten.
2. Die NEXT GROUP-Klausel darf in einer Leistenbeschreibung nicht angegeben werden, wenn eine LINE-Klausel fehlt.
3. Ein NEXT GROUP-Klausel mit Angabe von ganzzahl-1 wird als absolute NEXT GROUP-Klausel bezeichnet. Nachdem die letzte Zeile der zugehörigen Leiste gedruckt wurde, positioniert das Listenprogramm auf die Zeile weiter (vorwärts), deren Nummer durch ganzzahl-1 angegeben ist. Dabei kann sich auch ein Seitenvorschub ergeben, der mit dem Druck des Seitenfußes auf der alten und des Seitenkopfes auf der neuen Seite einhergeht (nur bei Rumpfleisten).
4. Als relativ gilt eine NEXT GROUP-Klausel mit Angabe von PLUS ganzzahl-2. Der Positionierungsvorschub von der letzten Druckzeile der zugehörigen Leiste aus beträgt ganzzahl-2 Zeilen.
5. Die NEXT GROUP-Klausel mit Angabe von NEXT PAGE zeigt an, dass nach dem Druck der zugehörigen Leiste ein Seitenvorschub ausgeführt wird (Erstellung von Seitenfuß und Seitenkopf bei Rumpfleisten automatisch).
6. Wenn in einer USE BEFORE REPORTING-Prozedur eines Gruppenfußes das PRINT-SWITCH-Sonderregister durch eine MOVE-Anweisung mit 1 belegt wird, dann unterdrückt das Listenprogramm die Funktion der NEXT GROUP-Klausel.

Allgemeine Regeln

1. Die NEXT GROUP-Klausel darf nur in einem 01-Eintrag einer Leistenerklärung auftreten.
2. Regeln für den Listenkopf
   a. Wenn für den Listenkopf allein eine ganze Seite zur Verfügung stehen soll, dann muss seine Beschreibung die NEXT GROUP-Klausel mit Angabe von NEXT PAGE enthalten.
   b. Soll für den Listenkopf keine ganze Seite reserviert werden, so müssen folgende Regeln eingehalten werden:
      - Die NEXT GROUP-Klausel darf nicht die Angabe NEXT PAGE enthalten.
      - ganzzahl-1 einer absoluten NEXT GROUP-Klausel muss eine größere Zeilennummer als jene angeben, die der letzten Druckzeile des Listenkopfes zukommt.
      - Eine absolute oder relative NEXT GROUP-Klausel ist so zu wählen, dass der Seitenkopf als nächste Leiste noch in seiner spezifischen Zone Platz findet.
        Eine Positionierung auf die Zeile mit der Nummer ganzzahl-d (siehe „PAGELIMIT-Klausel") oder sogar eine Zeile mit höherer Nummer infolge der NEXT GROUP-Klausel, ist unzulässig.
3. Regeln für den Seitenkopf

   Die NEXT GROUP-Klausel darf in der Beschreibung des Seitenkopfes nicht verwendet werden.
4. Regeln für die Rumpfleisten

a. ganzzahl-1 einer absoluten NEXT GROUP-Klausel muss einerseits größer oder gleich ganzzahl-d, andererseits aber kleiner oder gleich ganzzahl-f (siehe „PAGELIMIT-Klausel") sein.

   Wenn die Zeilennummer ganzzahl-1 der absoluten NEXT GROUP-Klausel kleiner oder gleich der Nummer jener Zeile ist, die als letzte Zeile der Rumpfleiste gedruckt wurde, deren Beschreibung die NEXT GROUP-Klausel enthält, dann führt das Listenprogramm einen Seitenvorschub (automatische Erstellung des Seitenfußes und Seitenkopfes inbegriffen) aus und positioniert auf die Zeile, die ganzzahl-1 angibt.

b. Wenn die Positionierung infolge einer relativen NEXT GROUP-Klausel über die untere Zonengrenze ganzzahl-f (siehe „PAGE LIMIT-Klausel") hinausführen würde, dann führt das Listenprogramm einen Seitenvorschub (mit Seitenfuß- und Seitenkopferstellung) aus und positioniert auf die obere Zonengrenze ganzzahl-d (siehe „PAGE LIMIT-Klausel").

c. Die Angabe NEXT PAGE in einer NEXT GROUP-Klausel zeigt an, dass keine weitere Rumpfleiste auf die aktuelle Seite gedruckt werden soll. Das Listenprogrammsteuersystem positioniert auf die obere Zonengrenze ganzzahl-d der nächsten Seite (Seitenfuß und Seitenkopf werden dabei erstellt).

d. Für den Fall, dass infolge eines Gruppenwechsels mehrere Gruppenfüße unmittelbar nacheinander erstellt werden, besteht die Möglichkeit, dem Listenprogramm anzuzeigen, dass es die Funktion der NEXT GROUP-Klausel nur für jenen Gruppenfuß ausführt, der zu der hierarchischen Stufe gehört, auf welcher der Gruppenwechsel auftrat (siehe dazu „CBL-CTR-Sonderregister").

5. Regel für den Seitenfuß

   Die NEXT GROUP-Klausel in der Beschreibung des Seitenfußes ist nicht erlaubt.

6. Regel für den Listenfuß

   Die Beschreibung des Listenfußes darf die NEXT GROUP-Klausel nicht enthalten.

Beispiel 10-6

```
01  LINE PLUS 2 NEXT GROUP PLUS 1.
    TYPE CONTROL FOOTING TAG.
    02...
```

Die NEXT GROUP-Klausel bewirkt, dass das Listenprogramm nach Erstellung des obigen Gruppenfußes nur eine Zeile weiterpositioniert.

10.2.13 PICTURE-Klausel

Format

```
{PICTURE | PIC} IS maskenzeichenfolge
```

Syntaxregel

1. Das Maskenzeichen N ist nicht erlaubt.

Weitere Syntaxregeln und Allgemeine Regeln zu PICTURE siehe „PICTURE-Klausel".

10.2.14 SIGN-Klausel

Funktion

Die SIGN-Klausel beschreibt die Position und die Darstellungsart des Rechenvorzeichens für numerische Datenfelder.

Format

```
[SIGN IS] {LEADING | TRAILING} SEPARATE CHARACTER
```

Syntaxregeln

1. Die SIGN-Klausel darf nur für eine numerische Datenerklärung angegeben werden, die in der PICTURE-Klausel mit dem Symbol S beschrieben ist.
2. Die Datenerklärungen müssen explizit oder implizit mit USAGE IS DISPLAY beschrieben sein.
3. Wenn die SIGN-Klausel in einem Leistenbeschreibungseintrag vorhanden ist, muss sie die SEPARATE CHARACTER-Angabe enthalten.
4. Die SIGN-Klausel gibt die Position und Darstellungsart des Rechenvorzeichens an. Sie wird nur für numerische Datenerklärungen verwendet, die in der Maskenzeichenfolge das Symbol S enthalten. Das S zeigt lediglich das Vorhandensein, nicht jedoch die Darstellungsart des Rechenvorzeichens an.
5. Eine numerische Datenerklärung, deren Maskenzeichenfolge ein S enthält, für die aber die SIGN-Klausel nicht angegeben wurde, hat ein Rechenvorzeichen. Die Darstellung und Position werden durch das Symbol S jedoch nicht spezifiziert (Darstellung des Rechenvorzeichens siehe „USAGE-Klausel").

Allgemeine Regeln

1. Für die erforderliche SEPARATE CHARACTER-Angabe gelten folgende Regeln:
   a. Das Symbol S in einer Maskenzeichenfolge wird zur Größe des Datenfeldes gerechnet.
   b. Es wird angenommen, dass das Rechenvorzeichen im Platz der führenden (LEADING) bzw. der letzten (TRAILING) Ziffer des numerischen Datenelementes enthalten ist. Diese Zeichenposition ist keine Ziffernposition.
   c. Die Rechenvorzeichen für positiv und negativ sind standardmäßig „+“ bzw. „-“.
2. Jede numerische Datenerklärung, die in der Maskenzeichenfolge das Symbol S enthält, ist eine mit Vorzeichen versehene Datenerklärung. Wird eine SIGN-Klausel für eine solche Erklärung angegeben und ist eine Konvertierung für arithmetische Operationen oder für Vergleiche notwendig, erfolgt diese Konvertierung automatisch.

10.2.15 SOURCE-Klausel

Funktion

Die SOURCE-Klausel nennt jenes Datenfeld, dessen Inhalt das Listenprogrammsteuersystem durch eine implizite MOVE-Anweisung in das druckfähige Feld, in dessen Beschreibung die SOURCE-Klausel enthalten ist, transportiert, sobald diese gedruckt werden soll.

Format

```
SOURCE IS bezeichner
```

Syntaxregel

1. Jeder Bezeichner, der in einem der Kapitel der DATA DIVISION definiert ist, kann in einer SOURCE-Klausel angegeben werden. Aus der REPORT SECTION darf in einer SOURCE-Klausel allerdings nur das PAGE-COUNTER-Sonderregister ("Seitenzähler“), das LINE-COUNTER-Sonderregister ("Zeilenzähler“) oder ein Summenzähler an-gegeben werden, wenn er zu der Liste gehört, in deren Beschreibung die SOURCE-Klausel auftritt.

Allgemeine Regeln

1. Die Beschreibung eines elementaren Feldes, die eine SOURCE-Klausel enthält, muss auch eine COLUMN-Klausel ausweisen, d.h. druckfähig sein.
2. Die SOURCE-Klausel erzeugt in Verbindung mit der COLUMN-Klausel eine implizite MOVE-Anweisung. Für diese MOVE-Anweisung ist das Sendefeld durch den Bezeichner aus der SOURCE-Klausel festgelegt. Als Empfangsfeld dient das druckfähige Feld, dessen Beschreibung die SOURCE-Klausel enthält. Die Maskenzeichenfolgen der PICTURE-Klauseln der beiden Felder müssen mit den Regeln der MOVE-Anweisung (siehe "MOVE-Anweisung") in Einklang sein.
3. Da die SOURCE-Klausel niemals zu einer Wertänderung des in ihr angeführten Datenfeldes führen kann, darf sie sich auch auf ein Gruppenwechseldatenfeld beziehen. Generell wird der Wertinhalt des Sendefeldes gedruckt, der bei der Ausführung der MOVE-Anweisung (Erstellung der zugehörigen Leiste) vorhanden ist. Sollen in Gruppenfüßen die alten Werte der Gruppenwechseldatenfelder stehen, so ist die Funktion 1 des CBL-CTR-Sonderregisters (siehe "CBL-CTR-Sonderregister") zu verwenden.

Beispiel 10-7

```
FILE SECTION.
    ...
    02 ABTEILUNG  PIC XXX.
    ...
REPORT SECTION.
    ...
    02 COLUMN 19 PIC XXX SOURCE ABTEILUNG.
    ...
```

Die SOURCE-Klausel bewirkt, dass der Inhalt aus dem Datenfeld ABTEILUNG in das betroffene druckfähige Feld transportiert wird, sobald die Leiste erstellt wird, zu der das druckfähige Feld gehört.

10.2.16 SUM-Klausel

Funktion

Die SUM-Klausel bewirkt, dass das Listenprogramm zu bestimmten Zeiten der Listenerzeugung arithmetische Summationen ausführt und deren Ergebnisse anschließend oder später ausdruckt. Die SUM-Klausel gibt dem Listenprogramm zur Summenbildung die Summanden an. Sie stellt außerdem ein numerisches Feld, das für sie automatisch angelegt wird, zur Aufsummierung der Summanden bereit. Dieses Feld, d.h. der Summenzähler, ist zugleich das Sendefeld für die implizite MOVE-Anweisung, die zur Druckaufbereitung des druckfähigen Feldes dient, in dessen Beschreibung die SUM-Klausel enthalten ist.

Format

```
SUM {bezeichner-1}... [UPON datenname-1]
    [RESET ON {datenname-2 | FINAL}]
```

Syntaxregeln

1. Ein Bezeichner, der als Summand in einer SUM-Klausel angegeben wurde, muss in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION, LINKAGE SECTION oder REPORT SECTION definiert sein. Aus der REPORT SECTION dürfen nur Summenzähler als Summanden einer SUM-Klausel angesprochen werden.
2. Das Datenfeld eines jeden Bezeichners, der als Summand in einer SUM-Klausel spezifiziert wurde, muss numerisch beschrieben sein.
3. datenname-1 ist nur als Name einer Postenleiste zulässig, die in der aktuellen Listenbeschreibung definiert ist.
4. FINAL oder datenname-2 muss in der CONTROL-Klausel der aktuellen Listenerklärung angegeben sein.
5. bezeichner-1... identifiziert die Felder, welche im Summenzähler aufaddiert werden.
6. Die UPON-Angabe führt dazu, dass die spezifizierten Summanden nur bei der Ausführung einer solchen GENERATE-Anweisung aufsummiert werden, die genau die Postenleiste bezeichnet, die in der UPON-Angabe genannt ist (siehe "Anwendung der UPON-Angabe").
7. Die RESET-Angabe hebt die Standardrücksetzung des Summenzählers auf Null auf (siehe "Anwendung der RESET-Angabe").

Allgemeine Regeln

1. Die SUM-Klausel darf nur in Beschreibungen von Gruppenfüßen angegeben werden.

   Im Listenausschnitt

   ```
   JANUAR 02 B10 2 A 3.00
    B12 1 A 4.00
    B12 3 C 17.00
   EINKAEUFE & KOSTEN FUER 1-02 6 $24.00
   ```

   sind für den 2. Januar die Kosten der einzelnen Posten und deren Summe ausgedruckt.

   Die in den ersten drei Druckzeilen wiedergegebenen Einzelkosten wurden infolge der GENERATE-Anweisung gedruckt, die auf die Postenleiste

   ```
   01  DETAIL-ZEILE TYPE DETAIL.
       02  ...
   ```

```
      .
      .
      02  COLUMN 50 PICTURE ZZ9.99 SOURCE KOSTEN.
```

Bezug nimmt. Diese GENERATE-Anweisung wurde ohne Gruppenwechsel dreimal durchlaufen, wodurch die Postenleiste dreimal hintereinander gedruckt wurde. Das Datenfeld mit dem Namen KOSTEN, das im FILE-Kapitel des Datenteils mit

```
02 K OSTEN PICTURE 999V99.
```

beschrieben ist, lieferte die Einzelkosten.

Bei der nächsten Ausführung der genannten GENERATE-Anweisung hatte sich der Inhalt des Gruppenwechseldatenfeldes TAG von 2 auf einen anderen Wert geändert. Der Gruppenfuß

```
01  ... TYPE CONTROL FOOTING TAG.
    02 ...
    .
    .
    02  SUMME-TAG COLUMN 49 PICTURE $9.99 SUM KOSTEN.
```

wurde als vierte Zeile des obigen Listenausschnittes erzeugt, wobei der Inhalt des Summenzählers SUMME-TAG ab Spalte 49 ausgedruckt wurde.

Folgende Aktionen wurden für die Summenbildung vorgenommen:
Der Compiler erzeugte zur Übersetzungszeit für die SUM-Klausel den Summenzähler SUMME-TAG. Bei der Ausführung der entsprechenden INITIATE-Anweisung (zur Programmausführungszeit) setzte das Listenprogramm den Summenzähler auf Null. Anschließend wurde bei jeder Ausführung der GENERATE-Anweisung GENERATE DETAIL-ZEILE der laufende Wert aus dem Datenfeld KOSTEN zum Inhalt des Summenzählers SUMME-TAG addiert. Da das Listenprogramm diese Summation (siehe unter 5. "Postenaufsummierung“) unmittelbar nach dem Gruppenwechseltest und den sich daraus ergebenden Aktionen vornimmt, wird mit dem Gruppenfuß für TAG die Summe aus den Einzelkosten für den 2.Januar ausgedruckt. Bei der Erzeugung der Gruppenfüße wird zuletzt der Summenzähler auf Null zurückgesetzt, da die SUM-Klausel keine RESET-Angabe enthält.

2. Anwendung der PICTURE-Klausel

   Wenn innerhalb einer Leistenerklärung die Beschreibung eines Datenfeldes eine SUM-Klausel enthält, dann beschreibt die zugehörige PICTURE-Klausel nicht nur das Datenfeld, sondern auch den Summenzähler, den der Compiler wegen der SUM-Klausel anlegt. Das Datenfeld dient für den Fall, dass es druckfähig ist, dazu, den Inhalt des zugeordneten Summenzählers auszudrucken. Die PICTURE-Klausel muss das Datenfeld als numerisch-druckaufbereitet beschreiben, wobei Druckaufbereitungssymbole für die Summenzähler ignoriert werden.

3. Anwendung des Summenzählers

   Wenn in einer Feldbeschreibung, die eine SUM-Klausel enthält, auf die Stufennummer unmittelbar ein Datenname folgt, dann ist dieser Datenname der Name des Summenzählers. Über diesen Namen hat der Programmierer die Möglichkeit, auf den Summenzähler zuzugreifen (z.B. um seinen Inhalt vor dem Druck zu runden). Der Summenzähler ist ein vom Compiler angelegtes Datenfeld, dem die Verwendung von COMP-3 zugeordnet ist und dessen numerische Eigenschaften durch die angegebene PICTURE-Klausel bestimmt sind.

4. Arten der Summation
   Der Programmierer kann drei Arten von Summationen spezifizieren:
   die Postenaufsummierung, die hierarchische Hochsummierung von Summen und die Addition hierarchisch gleichwertiger Summen.

5. Postenaufsummierung

Der Zeitpunkt, zu dem das Listenprogramm einen Summanden (bezeichner-1... aus der SUM-Klausel) im zugeordneten Summenzähler aufsummiert, hängt vom Summanden selbst ab.

Die Summanden, die für die Postenaufsummierung herangezogen werden, sind jene, die selbst keine Summenzähler sind, also jene, die außerhalb des REPORT-Kapitels definiert sind.

Die Postenaufsummierung bildet die Basis für die zwei restlichen Arten von Summationen. Die Bezeichnung Postenaufsummierung kommt davon, dass die von ihr erfassten Summanden meistens mit den Postenleisten der Liste ausgedruckt werden.

Die Postenaufsummierung findet bei jeder Ausführung einer GENERATE-Anweisung statt. Der Programmierer muss daher dafür sorgen, dass die betroffenen Summanden zu diesen Zeitpunkten die aktuellen Werte enthalten. Wenn in einer SUM-Klausel eine UPON-Angabe vorliegt, dann werden die vom Summenzähler verschiedenen Summanden der SUM-Klausel nur bei einer solchen Postenaufsummierung addiert, die bei der Ausführung einer GENERATE-Anweisung stattfindet, die dieselbe Postenleiste anführt wie die UPON-Angabe (für eine Summenliste ist es daher sinnlos, eine UPON-Angabe zu machen; siehe "Anwendung der UPON-Angabe“). Wenn die SUM-Klausel allerdings keine UPON-Angabe enthält, dann werden die nicht als Summenzähler definierten Summanden bei jeder Ausführung einer beliebigen GENERATE-Anweisung (bezüglich einer Liste) im zugehörigen Summenzähler aufsummiert (Postenaufsummierung).

Das Listenprogramm führt die Postensummation erst nach den Aktionen aus, die sie bezüglich des Gruppenwechsels (Test; Erstellung der Gruppenfüße und -köpfe, falls der Test positiv ist) unternimmt. Zu diesen Gruppenwechselaktionen zählt auch die Rücksetzung der Summenzähler auf Null, nachdem der Gruppenfuß, dessen Beschreibung die entsprechenden SUM-Klauseln enthält, erstellt worden war (siehe "Anwendung der RESET-Angabe"). Damit ist sichergestellt, dass die ausgedruckte Summe nur die Werte der Summanden über jene Gruppe von Postenleisten erfasst, die durch den zugehörigen Gruppenfuß abgeschlossen wird (z.B. die Summe der Kosten des 2. Januar).

6. Hierarchische Hochsummierung von Summen

   Die Voraussetzung für eine solche Summation besteht darin, dass eine SUM-Klausel eines Gruppenfußes mindestens einen Summenzähler als Summanden besitzt, der auf Grund einer SUM-Klausel eines Gruppenfußes mit einem niedrigeren hierarchischen Rang angelegt wurde. Die hierarchische Hochsummierung ist nur möglich, wenn es bezüglich einer Liste mindestens zwei Gruppenfüße gibt, in deren Beschreibung wenigstens je eine SUM-Klausel vorhanden ist.

   Der Inhalt eines Summenzählers, der in einer SUM-Klausel eines anderen Gruppenfußes als Summand angegeben ist, wird zu dem Zeitpunkt, in dem der zugehörige (also mit niedrigerem Rang) Gruppenfuß erstellt wird, auf den Inhalt des Summenzählers addiert, in dessen SUM-Klausel er als Summand auftritt.

   Beispiel 10-8 erläutert die hierarchische Hochsummierung:

   Beispiel 10-8

   In der Beschreibung des Gruppenfußes

```
01  ... TYPE CONTROL FOOTING MONAT.
    02  ...
    .
    .
    02  SUMME-MONAT COLUMN 46 PICTURE $$$9.99 SUM
        SUMME-TAG.
```

   wird die hierarchische Hochsummierung in Verbindung mit der Gruppenfußbeschreibung

```
01  ... TYPE CONTROL FOOTING TAG.
    02  ...
    .
```

```
    .
02  SUMME-TAG COLUMN 49 PICTURE $$$9.99 SUM KOSTEN.
```

spezifiziert (siehe Beispiel 10-7 in Kapitel "SOURCE-Klausel").

Bei jeder Erstellung des Gruppenfußes mit dem Gruppenwechseldatenfeld TAG addiert das Listenprogramm den Inhalt des Summenzählers SUMME-TAG auf den Inhalt des Summenzählers SUMME-MONAT, und zwar bevor SUMME-TAG auf Null zurückgesetzt wird. Wenn der Gruppenfuß mit MONAT infolge eines Gruppenwechsels oder der TERMINATE-Anweisung erstellt wird, enthält der Summenzähler SUMME-MONAT (vor dem Rücksetzen auf Null) die Summe aller Tagessummen (Werte von SUMME-TAG zu den Summationszeitpunkten) des aktuellen Monats.

7. Addition hierarchisch gleichwertiger Summen

   Diese Summationsart liegt vor, wenn eine SUM-Klausel Summenzähler als Summanden enthält, die in dem vorliegenden Gruppenfuß über entsprechende SUM-Klauseln definiert sind. Normalerweise enthalten solche Summanden (also Summenzähler) Werte, die durch Postenaufsummierung erzeugt wurden.

   Beispiel 10-9

```
01  UNTER-BETRAG TYPE CONTROL FOOTING...
    02  SUMME-1 SUM ARB-FELD-1...
    02  SUMME-2 SUM ARB-FELD-2...
    02  SUMME   SUM SUMME-1 SUMME-2...
```

   Die Datenfelder ARB-FELD-1 und ARB-FELD-2 sind in der WORKING-STORAGE SECTION des Datenteils definiert. Im Summenzähler SUMME werden die Werte von SUMME-1 und SUMME-2 addiert, die voher durch Postenaufsummierung erzeugt wurden.

   Die Addition hierarchisch gleichwertiger Summen führt das Listenprogrammsteuersystem zu der Zeit aus, in welcher der betroffene Gruppenfuß erstellt wird. Wenn mehr als eine SUM-Klausel eine solche Addition erfordert, dann ist die zeitliche Reihenfolge durch die Reihenfolge dieser SUM-Klauseln bestimmt. Diese Reihenfolge ist für das Additionsergebnis wesentlich.

   Es ist einleuchtend, dass diese Addition vor der hierarchischen Hochsummierung durchgeführt wird. Damit ist sichergestellt, dass bei der Summierung hierarchisch gleichwertige Summen auch hierarchisch hochsummiert werden können.

   Beispiel 10-10

```
    ...
    CONTROLS ARE LAND, STADT.
    ...
01  LINE PLUS 2 TYPE CONTROL FOOTING STADT.
    02  SUMME-1 SUM MAENNER...
    02  SUMME-2 SUM FRAUEN...
    02  SUMME-STADT SUM SUMME-1, SUMME-2...
01  LINE PLUS 1 TYPE CONTROL FOOTING LAND.
    02  SUMME-LAND SUM SUMME-STADT...
    ...
```

   Die im Summenzähler SUMME-STADT durch die hierarchisch gleichwertige Summation der Werte aus den Summenzählern SUMME-1 und SUMME-2 (Postenaufsummierung) gebildeten Werte werden im Summenzähler SUMME-LAND hierarchisch hochsummiert (der Gruppenfuß mit LAND hat gegenüber dem Gruppenfuß mit STADT den höheren hierarchischen Rang). Dies ist nur möglich, weil der Summenzähler SUMME-STADT vor der Hochsummierung im Summenzähler SUMME-LAND den geeigneten Wert enthält.

8. Mischoperanden

   Eine SUM-Klausel ohne UPON-Angabe darf einen oder mehrere Operanden (= Summanden) folgender Eigenschaften enthalten:

   a. Operanden, die in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION definiert sind.
   b. Operanden, die in einem hierarchisch rangtieferen Gruppenfuß als Summenzähler definiert sind.
   c. Operanden, die in demselben Gruppenfuß (dessen Beschreibung die SUM-Klausel enthält) als Summenzähler definiert sind.

      Die Addition findet für jeden der oben beschriebenen Operationstypen zu den Zeiten statt, die weiter oben angeführt wurden.

9. Anwendung der UPON-Angabe

   a. Wenn in einer SUM-Klausel eine UPON-Angabe vorliegt, müssen alle Summanden aus dieser SUM-Klausel außerhalb der REPORT SECTION definiert sein, d.h. sie dürfen nur in der FILE SECTION, WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION und LINKAGE SECTION definiert sein.
   b. Die UPON-Angabe bewirkt, dass die Postenaufsummierung der Summanden aus der vorliegenden SUM-Klausel nur dann stattfindet, wenn eine GENERATE-Anweisung durchlaufen wird, die die Postenleiste anführt, die in der UPON-Angabe genannt wurde. Eine Postenaufsummierung infolge einer anderen GENERATE-Anweisung erfasst also keine der Summanden aus der zur Diskussion stehenden SUM-Klausel.

Beispiel 10-11

```
DATA DIVISION.
FILE SECTION.
FD  INFILE...
...
    RECORDS ARE MUELLER, MEIER.
01  MUELLER PICTURE 999.
01  MEIER PICTURE 9999.
REPORT SECTION.
...
01  MUELLER-DETAIL TYPE DETAIL.
02  LINE PLUS 1 COLUMN 1 PIC 999 SOURCE MUELLER.
01  MEIER-DETAIL TYPE DETAIL.
02  LINE PLUS 1 COLUMN 1 PIC 9999 SOURCE MEIER.
...
01  BETRAG TYPE CONTROL FOOTING...
02  SUMME-1 SUM MUELLER UPON MUELLER-DETAIL...
02  SUMME-2 SUM MEIER UPON MEIER-DETAIL...
...
PROCEDURE DIVISION.
...
GENERATE MUELLER-DETAIL.
...
GENERATE MEIER-DETAIL.
```

Da MUELLER und MEIER die Namen zweier verschiedener Datensätze in derselben Datei sind, können die aktuellen Satzinhalte nicht gleichzeitig zur Verfügung gestellt werden. Sobald ein MUELLER-Satz gelesen ist, wird die Anweisung GENERATE MUELLER-DETAIL ausgeführt. Dabei wird der laufende Wert von MUELLER im Summenzähler SUMME-1 aufaddiert. Die Addition des vorliegenden Wertes von MEIER

auf den Inhalt des Summenzählers SUMME-2 hingegen findet zu diesem Zeitpunkt nicht statt. Wenn ein MEIER-Satz gelesen ist, wird die Anweisung GENERATE MEIER-DETAIL ausgeführt, wobei die Postenaufsummierung im Summenzähler SUMME-2, nicht aber SUMME-1 vorgenommen wird.

10. Anwendung der RESET-Angabe
    a. Nur ein Datenname (FINAL eingeschlossen), der in der CONTROL-Klausel der selben Liste enthalten ist, darf in einer RESET-Angabe verwendet werden. Außerdem muss das Gruppenwechseldatenfeld einen höheren hierarchischen Rang einnehmen als der Gruppenfuß, in dessen Beschreibung die RESET-Angabe gemacht wird.
    b. Normalerweise setzt das Listenprogramm einen Summenzähler unmittelbar nach der Erzeugung des Gruppenfußes, in dessen Beschreibung er definiert ist, auf den Wert Null zurück. Ein Summenzähler, dessen SUM-Klausel eine RESET-Angabe enthält, wird nur zu solchen Zeitpunkten auf Null zurückgesetzt, wenn der (vorhandene oder gedachte) Gruppenfuß zu dem Gruppenwechseldatenfeld (oder FINAL), das in der RESET-Angabe auftritt, erstellt wird (bzw. werden würde). Die RESET-Angabe dient also dazu, bezüglich der angegebenen hierarchischen Stufe die Gesamtsumme zu bilden.

Beispiel 10-12

```
01  ... TYPE CONTROL FOOTING TAG.
    02  ...
    .
    .
    .
    02  COLUMN 65 PIC $$$$9.99 SUM KOSTEN RESET ON FINAL.
01  ... TYPE CONTROL FOOTING FINAL.
    02  ...
    .
    .
    .
    02  COLUMN 45 PIC $$$$9.99 SUM SUMME-TAG.
```

Weil die SUM-Klausel in der Beschreibung des Gruppenfußes mit dem Gruppenwechseldatenfeld TAG die Angabe RESET ON FINAL enthält, wird der laufende Wert des zugehörigen Summenzählers bei jeder Erzeugung des Gruppenfußes von TAG ausgedruckt, ohne dass der Summenzähler jemals auf Null gesetzt wurde. Erst bei der Erstellung des Gruppenfußes für FINAL wird er auf Null gesetzt. Jeder gedruckte Gruppenfuß für TAG weist daher die laufenden Gesamtkosten von der ersten (1. Tag) bis einschließlich der zuletzt vor dem aktuellen Gruppenkopf gedruckten Postenleiste der Liste aus.

Einem Gruppenwechseldatenfeld, das in einer RESET-Angabe auftritt, muss kein Gruppenfuß zugeordnet sein. Die Summenzählerrücksetzung wird, wie bereits erwähnt, auch dann ausgeführt, wenn zu einem Gruppenwechseldatenfeld, das in einer RESET-Angabe auftritt, kein Gruppenfuß existiert.

11. Aktionen des Listenprogramms

    Bei der Erstellung eines Gruppenfußes führt das Listenprogramm folgende Schritte aus (schematisch, da oft Schritte entfallen):

    a. Addition hierarchisch gleichwertiger Summen.
    b. Ausführung der USE BEFORE REPORTING-Prozeduren für den Gruppenfuß (siehe "USE BEFORE REPORTING-Anweisung").
    c. PRINT-SWITCH-Test.
       Enthält das PRINT-SWITCH-Sonderregister den Wert 1, dann wird Schritt d) übersprungen, d.h. Schritt e) folgt unmittelbar auf Schritt c), nachdem das Sonderregister auf Null zurückgesetzt wurde, andernfalls folgt Schritt d).
    d. Erzeugung des Gruppenfußes (falls druckfähig).

e. Hierarchische Hochsummierung.

f. Alle Summenzähler des Gruppenfußes, in deren SUM-Klausel keine RESET-Angaben vorliegen, werden durch implizite MOVE-Anweisungen mit Null belegt. Gleiches geschieht mit all jenen Summenzählern der übrigen Gruppenfüße, in deren SUM-Klauseln je eine solche RESET-Angabe auftritt, die sich gerade auf das Gruppenwechseldatenfeld bezieht, mit dem der aktuelle (also gerade erstellte) Gruppenfuß verknüpft ist.

10.2.17 TYPE-Klausel

Funktion

Die TYPE-Klausel gibt für die Leiste, in deren Beschreibung sie enthalten ist, an, um welchen Leistentyp es sich handelt, d.h. welche funktionellen Eigenschaften der Leiste zukommen. Damit sind auch die Umstände festgelegt, unter welchen das Listenprogramm die Leiste erzeugt.

Format

```
TYPE IS {  {REPORT HEADING | RH}
         | {PAGE HEADING | PH}
         | {CONTROL HEADING | CH} {datenname-1 | FINAL}
         | {DETAIL | DE}
         | {CONTROL FOOTING | CF} {datenname-2 | FINAL}
         | {PAGE FOOTING | PF}
         | {REPORT FOOTING | RF}
        }
```

Syntaxregeln

1. RH ist die Abkürzung für REPORT HEADING,
   PH ist die Abkürzung für PAGE HEADING usw.
2. FINAL, datenname-1 und datenname-2 müssen in der CONTROL-Klausel der zugehörigen Listenerklärung (RD) angegeben sein.
3. REPORT HEADING kennzeichnet die Leiste, die pro Liste nur einmal und zwar als erste Leiste der Liste erzeugt wird, d.h. den Listenkopf. Der Listenkopf wird automatisch erstellt, sobald die erste GENERATE-Anweisung ausgeführt wird.
4. PAGE HEADING kennzeichnet solche Leisten, die jeweils als erste Leiste einer jeden Seite erstellt werden, d.h. den Seitenkopf. Eine Seite, die ganz für den Listenkopf oder Listenfuß reserviert ist, erhält keinen Seitenkopf. Wenn ein Listenkopf vorhanden ist und für ihn allein keine Seite reserviert ist, dann wird der Seitenkopf auf der ersten Seite der Liste als zweite Leiste erzeugt.
5. CONTROL HEADING kennzeichnet solche Leisten, die bei der Ausführung der (chronologisch) ersten GENERATE-Anweisung und bei jedem Gruppenwechsel in Serie erstellt werden, d.h. die Gruppenköpfe. Jeder Gruppenkopf ist durch die Angabe von FINAL oder datenname-1 mit einem und nur einem hierarchischen Rang gekoppelt, der die Reihenfolge, in der die Gruppenköpfe erstellt werden, bestimmt (siehe „CONTROL-Klausel").
6. DETAIL kennzeichnet solche Leisten, die auf Grund ihrer Angaben in einer GENERATE-Anweisung erstellt werden, d.h. die Postenleisten. Der Name einer Postenleiste muss bezüglich einer Liste eindeutig sein.
7. CONTROL FOOTING kennzeichnet solche Leisten, die bei jedem Gruppenwechsel und bei der Ausführung der TERMINATE-Anweisung in Serie erstellt werden, d.h. Gruppenfüße. Jeder Gruppenfuß ist durch die Angabe von FINAL oder datenname-2 mit einem und nur einem hierarchischen Rang gekoppelt, der die Reihenfolge, in der die Gruppenfüße erstellt werden, bestimmt (siehe „CONTROL-Klausel").
8. PAGE FOOTING kennzeichnet die Leiste, die als letzte einer jeden Seite der Liste erstellt wird, d.h. den Seitenfuß. Eine Seite, die zur Gänze für den Listenkopf oder den Listenfuß reserviert ist, erhält keinen Seitenfuß. Wenn für den Listenfuß keine ganze Seite vorgesehen ist, dann wird der Seitenfuß auf der letzten Seite der Liste als vorletzte Leiste erzeugt.
9. REPORT FOOTING kennzeichnet die Leiste, die nur einmal, und zwar als letzte Leiste der Liste erstellt wird, d.h. den Listenfuß. Der Listenfuß wird als letzte Leiste bei der Ausführung der TERMINATE-Anweisung erzeugt.

Allgemeine Regeln

1. Regeln für den Listenkopf
   a. Pro Liste darf nur ein Listenkopf definiert werden.
   b. Der Listenkopf kann so beschrieben werden, dass er als einzige Leiste einer Seite gedruckt wird. Die NEXT GROUP-Klausel mit NEXT PAGE-Angabe in der Beschreibung des Listenkopfes stellt sicher, dass neben dem Listenkopf keine weitere Leiste auf dieselbe Seite gedruckt wird (siehe „NEXT GROUP-Klausel").
2. Regeln für den Seitenkopf und den Seitenfuß
   a. Nur ein Seitenkopf und ein Seitenfuß dürfen pro Liste definiert werden.
   b. Mit folgenden Ausnahmen wird der Seitenkopf als erste Leiste und der Seitenfuß als letzte Leiste einer jeden Seite der Liste gedruckt:
      - Auf der ersten Seite der Liste wird der Listenkopf vor dem Seitenkopf gedruckt, wenn für den Listenkopf alleine keine ganze Seite vorgesehen ist.
      - Auf den Seitenfuß der letzten Seite der Liste folgt der Listenfuß, wenn für ihn allein keine ganze Seite zur Verfügung gestellt wurde.
3. Regeln für Gruppenköpfe und Gruppenfüße
   a. Zu einer hierarchischen Stufe dürfen nicht mehr als ein Gruppenkopf und ein Gruppenfuß angegeben werden.
   b. Gruppenköpfe und Gruppenfüße werden zu folgenden Zeitpunkten erstellt:
      - Sobald die erste GENERATE-Anweisung bezüglich einer Liste ausgeführt wird, erzeugt das Listenprogramm alle Gruppenköpfe in der Reihenfolge ihrer hierar chischen Ränge (höchster Rang zuerst niedrigster Rang zuletzt), bevor die Postenleiste als direktes Produkt der GENERATE-Anweisung erstellt wird.
      - Sobald das Listenprogramm bei der Ausführung einer der (chronologisch) nächsten GENERATE-Anweisungen einen Gruppenwechsel feststellt, erstellt sie vor der durch die aktuelle GENERATE-Anweisung direkt aufgerufenen Postenleiste die entsprechenden Folgen von Gruppenfüßen und Gruppenköpfen (siehe „GENERATE-Anweisung" und „CONTROL-Klausel").
      - Wenn die Erstellung der Liste durch die Ausführung der TERMINATE-Anweisung abgeschlossen wird, erzeugt das Listenprogramm alle Gruppenfüße in der Reihenfolge vom niedrigsten zum höchsten hierarchischen Rang.
   c. FINAL, datenname-1 oder datenname-2, die in der TYPE-Klausel eines Gruppenkopfes oder Gruppenfußes angegeben sind, müssen in der CONTROL-Klausel der zugehörigen Listenerklärung auftreten.
   d. Ein mit FINAL gekoppelter Gruppenkopf oder Gruppenfuß darf bei der Listenerstellung nur einmal erzeugt werden und zwar als erste (bei Ausführung der ersten GENERATE-Anweisung) bzw. als letzte (bei Ausführung der TERMINATE-Anweisung) Rumpfleiste der Liste.
4. Regeln für Postenleisten
   Das Listenprogramm erzeugt nur dann eine Postenleiste, wenn diese in einer GENERATE-Anweisung angegeben ist und wenn diese GENERATE-Anweisung ausgeführt wird. Für jede Liste muss mindestens eine Postenleiste definiert werden. Dies gilt auch dann, wenn zur Erzeugung der Liste keine Postenleiste explizit verwendet wird, d.h. die Liste ohne auf Einzelheiten (Posten) einzugehen, im wesentlichen nur summarische Information enthält (siehe „GENERATE-Anweisung").
5. Regeln für den Listenfuß
   a. Für eine Liste darf nur ein Listenfuß definiert werden. Der Listenfuß wird als letzte Leiste der Liste erzeugt.

b. Es ist möglich, für den Listenfuß eine ganze Seite zu reservieren. Um dies zu erreichen, muss in der Beschreibung des Listenfußes die erste LINE-Klausel mit
NEXT PAGE-Angabe angegeben werden (siehe „LINE-Klausel").

6. Leistenfolge innerhalb einer Liste
   a. Keine Leiste der Liste darf vor dem Listenkopf und hinter dem Listenfuß gedruckt werden.
   b. Die Folge der Kopf- und Fußleisten zu einer Liste sieht schematisch folgendermaßen aus:

      Listenkopf (darf nur einmal erscheinen)
      Seitenkopf
      ...
      Gruppenkopf
      Postenleiste
      Gruppenfuß
      ...
      Seitenfuß
      Listenfuß (darf nur einmal erscheinen)
   c. Die Gruppenköpfe werden stets in der Reihenfolge ihres hierarchischen Ranges unmittelbar hintereinander erzeugt:
      Gruppenkopf mit FINAL (höchstmöglicher Rang, darf nur einmal erscheinen)
      Gruppenkopf mit nächstniedrigem Rang
      ...
      Gruppenkopf mit niedrigstem Rang
   d. Die Gruppenfüße werden stets in der Reihenfolge ihrer hierarchischen Ränge unmittelbar hintereinander erstellt:

      Gruppenfuß mit niedrigstem Rang
      Gruppenfuß mit nächsthöherem Rang
      ...
      Gruppenfuß mit FINAL (höchstmöglicher Rang, darf nur einmal erscheinen)

10.2.18 USAGE-Klausel

Format

```
[USAGE IS] DISPLAY
```

Syntaxregel

1. In der REPORT SECTION wird die USAGE-Klausel nur verwendet, um das Datenformat druckfähiger Datenelemente festzulegen.

Weitere Syntaxregeln und Allgemeine Regeln zu USAGE IS DISPLAY siehe „USAGE-Klausel".

10.2.19 VALUE-Klausel

Format

```
[VALUE is literal]
```

Syntaxregel

Syntaxregeln und Allgemeine Regeln siehe Format 1 der „VALUE-Klausel".

10.3 Sprachelemente PROCEDURE DIVISION

In diesem Kapitel werden folgende Themen behandelt:

10.3.1 GENERATE-Anweisung

Funktion

Die GENERATE-Anweisung veranlasst das Listenprogramm, einen Teil der Liste in Übereinstimmung mit der Listenbeschreibung aus der REPORT SECTION der DATA DIVISION zu erzeugen.

Format

```
GENERATE {datenname | listenname}
```

Syntaxregeln

1. Der Datenname muss in der REPORT SECTION der DATA DIVISION als Name einer Postenleiste (01-Eintrag) definiert sein.
2. Der Listenname muss in einer Listenerklärung in der REPORT SECTION der DATA DIVISION als solcher (RD-Eintrag) definiert sein.
3. Infolge einer GENERATE-Anweisung, die eine Postenleiste anspricht, wird ein Teil der Liste gedruckt (siehe Regel 5).
4. Die Angabe eines Listennamens in einer GENERATE-Anweisung zeigt an, dass ein Teil der Liste erstellt werden soll (siehe Regel 8).
5. Das Listenprogramm erzeugt auf Grund einer GENERATE-Anweisung, in der eine Postenleiste angesprochen ist, einen Teil der Liste. Wie sich dieser Teil zusammensetzt, geben die Regeln 6 und 7 an. Außerdem werden im Allgemeinen diverse Summationen (siehe „SUM-Klausel") durchgeführt.
6. Bei der Ausführung der (chronologisch) ersten GENERATE-Anweisung (relativ zur Ausführung der entsprechenden INITIATE-Anweisung) werden für den Fall, dass sie eine Postenleiste angibt, folgende Leisten erstellt (vorausgesetzt, sie sind definiert):
   a. Listenkopf
   b. Seitenkopf
   c. alle Gruppenköpfe von der höchsten zu der niedrigsten Hierarchiestufe und
   d. die in der GENERATE-Anweisung angegebene Postenleiste.
7. Wenn eine GENERATE-Anweisung, die eine Postenleiste angibt, nicht als (chronologisch) erste Anweisung ausgeführt wird, dann prüft des Listenprogrammsteuersystem zuerst, ob ein Gruppenwechsel vorliegt. Ist dies der Fall, so erstellt es in der angegebenen Reihenfolge folgende Leisten:
   a. alle Gruppenfüße vom niedrigsten bis einschließlich jenem Rang, auf dem der Gruppenwechsel stattfand, und alle Gruppenköpfe vom Rang des Gruppenwech sels bis zum niedrigsten hierarchischen Rang.
   b. die in der GENERATE-Anweisung angegebene Postenleiste.

   Punkt a) entfällt, wenn kein Gruppenwechsel eintrat.
8. Aufgrund einer GENERATE-Anweisung, die eine Liste angibt, führt das Listenprogramm dieselben Aktionen bis auf die Erstellung der Postenleiste aus, die nun entfällt. Es gibt gegenüber den Regeln 5, 6, und 7 keine zusätzlichen Aktionen.
9. Wenn im Laufe der Listenerzeugung die aktuelle Seite voll ist, dann erzeugt das Listenprogramm automatisch einen Seitenvorschub. Vor dem Seitenvorschub wird noch der Seitenfuß (falls vorhanden) und nach dem Seitenvorschub der Seitenkopf erstellt.

Allgemeine Regeln

1. Zu der Zeit, in der eine GENERATE-Anweisung mit der Angabe einer Postenleiste ausgeführt wird, müssen dem Listenprogramm folgende Informationen zur Verfügung stehen:
   a. Jede Quellinformation, die der Postenleiste und allen übrigen Leisten, die infolge der GENERATE-Anweisung erzeugt werden, durch SOURCE-Klauseln zugeordnet ist.
   b. Die numerischen Daten jener Summanden, die das Listenprogramm benötigt, um die erforderlichen Summen zu bilden.
2. Die Summenlistenerzeugung hat nur für solche Listen einen Sinn, für die auch Gruppenfüße definiert sind, in deren Beschreibung SUM-Klauseln enthalten sind.
3. Für Listen, in deren Beschreibungen mehr als eine Postenleiste definiert ist, sollte keine Summenlistenerzeugung vorgenommen werden, da bei der Summenlistenerzeugung zu den einzelnen Postenleisten keine Beziehung hergestellt werden kann.
4. Das CBL-CTR-Sonderregister (siehe Abschnitt „CBL-CTR-Sonderregister") wird bei der Ausführung der (chronologisch) ersten GENERATE-Anweisung vom Listenprogramm abgefragt. Indem der Programmierer durch eine MOVE-Anweisung dieses Sonderregister zwischen den Ausführungen der INITIATE- und der (chronologisch) ersten GENERATE-Anweisung mit einem der möglichen Werte belegt, kann er eine oder sogar beide der folgenden wahlweisen Funktionen anfordern:
   a. Die Versorgung der Gruppenfüße, des Seitenfußes und des Seitenkopfes mit den passenden Werten der Gruppenwechseldatenfelder.
   b. Die bedingte Ausführung der NEXT GROUP-Klauseln der Gruppenfüße (siehe „CBL-CTR-Sonderregister").

10.3.2 INITIATE-Anweisung

Funktion

Die INITIATE-Anweisung veranlasst das Listenprogramm, die Erzeugung einer oder mehrerer Listen einzuleiten.

Format

```
INITIATE {listenname-1}...
```

Syntaxregeln

1. listenname-1... muss in einer Listenerklärung in der REPORT SECTION der DATA DIVISION (RD-Eintrag) definiert sein.
2. Die INITIATE-Anweisung nennt dem Listenprogramm jene Listen (listenname-1...), deren Erzeugung es einzuleiten hat.
3. Die INITIATE-Anweisung bewirkt, dass das Listenprogramm für jede in ihr benannte Liste
   - alle Summenzähler auf Null,
   - das LINE-COUNTER-Sonderregister (Zeilenzähler) auf Null und
   - das PAGE-COUNTER-Sonderregister (Seitenzähler) auf Eins setzt.

Allgemeine Regeln

1. Die INITIATE-Anweisung eröffnet nicht die Datei, die einer in ihr benannten Liste zugeordnet ist. Diese Listendatei, die als sequenzielle Ausgabedatei beschrieben ist, muss daher noch vor Ausführung der INITIATE-Anweisung durch eine OPEN-Anweisung mit OUTPUT oder EXTEND eröffnet werden.
2. Wenn nach einer INITIATE-Anweisung eine zweite INITIATE-Anweisung durchlaufen wird, die dieselbe Liste wie die erste INITIATE-Anweisung benennt, dann muss vorher eine TERMINATE-Anweisung für diese Liste ausgeführt werden.
3. Wenn in Bezug auf eine Liste zwischen einer INITIATE- und einer TERMINATE-Anwei-sung keine GENERATE-Anweisung durchlaufen wurde, so hebt die TERMINATE-Anweisung die INITIATE-Anweisung auf, ohne dass die Liste (auch kein Teil davon) erzeugt wurde.
4. Nach der Ausführung der INITIATE-Anweisung und vor der (chronologisch) ersten GENERATE-Anweisung kann der Programmierer wahlweise spezielle Funktionen vom Listenprogramm anfordern, indem er mittels einer MOVE-Anweisung das CBL-CTR-Sonderregister (siehe "CBL-CTR-Sonderregister ") mit einem entsprechenden Wert belegt.

10.3.3 TERMINATE-Anweisung

Funktion

Die TERMINATE-Anweisung veranlasst das Listenprogramm, die Erzeugung der in ihr angegebenen Listen abzuschließen.

Format

```
TERMINATE {listenname-1}...
```

Syntaxregeln

1. listenname-1... muss in einer Listenerklärung in der REPORT SECTION der DATA DIVISION (RD-Eintrag) definiert sein.
2. Die TERMINATE-Anweisung nennt dem Listenprogramm jene Listen, deren Erzeugung es abzuschließen hat.
3. Für jede in der TERMINATE-Anweisung benannte Liste führt das Listenprogrammsteuersystem folgende Aktionen in der angegebenen Reihenfolge aus:
   a. Alle Gruppenfüße werden so erstellt, als ob sie infolge eines Gruppenwechsels auf höchster hierarchischer Stufe zu erzeugen wären (dies impliziert natürlich die Erstellung des Seitenfußes und des Seitenkopfes, wenn dabei ein Seitenvorschub erforderlich ist, und die Ausführung von Summationen).
   b. Der Seitenfuß wird erstellt.
   c. Der Listenfuß wird erzeugt.

   Diese Aktionen werden allerdings nicht ausgeführt, wenn für eine Liste zwischen den Ausführungen der INITIATE- und TERMINATE-Anweisung keine GENERATE-Anweisung durchlaufen wurde.

Allgemeine Regeln

1. Vor jeder TERMINATE-Anweisung muss für listenname-1 eine INITIATE-Anweisung für listenname-1 durchlaufen worden sein.
2. Da die TERMINATE-Anweisung die betroffene Listendatei nicht schließt, muss ihr (chronologisch) eine CLOSE-Anweisung folgen. Jede Liste, die in einem initialisierten Zustand ist, muss abgeschlossen werden (durch eine TERMINATE-Anweisung), bevor eine CLOSE-Anweisung für diese Listendatei ausgeführt wird.

10.3.4 USE BEFORE REPORTING-Anweisung

Funktion

Die USE BEFORE REPORTING-Anweisung ermöglicht es, in der PROCEDURE DIVISION Anweisungen anzugeben, die unmittelbar vor der Ausgabe der genannten Liste vom Listenprogramm ausgeführt werden.

Format

```
USE [GLOBAL] BEFORE REPORTING leistenname.
```

Syntaxregeln

1. Der Leistenname muss als Name (Datenname) in einem 01-Eintrag einer Leistenerklärung der REPORT SECTION der DATA DIVISION definiert sein. Mit Ausnahme der Postenleiste darf jeder Leistentyp in der USE BEFORE REPORTING-Anweisung angegeben werden.
2. Der Leistenname nennt die Leiste, für welche die der USE BEFORE REPORTING-Anweisung folgenden USE-Prozeduren (USE BEFORE REPORTING-Prozeduren) auszuführen sind.
3. Die USE BEFORE REPORTING-Anweisung selbst wird nie ausgeführt. Sie definiert vielmehr die Aufrufsbedingungen für die Ausführung der anschließend vereinbarten USE-Prozeduren.
4. Die für eine Leiste vereinbarten USE-Prozeduren werden unmittelbar vor der Erstellung dieser Leiste ausgeführt.
5. Die Anzahl der USE BEFORE REPORTING-Anweisungen in einer PROCEDURE DIVISION darf 39 nicht überschreiten.
6. Eine USE-Prozedur mit GLOBAL-Angsbe ist nur zulässig, wenn die entsprechende FD- bzw. RD-Erklärung das GLOBAL-Attribut enthält und wenn die USE-Prozedur im selben Programm wie die zugehörige FD- bzw. RD-Erklärung vereinbart wird. Weitere Angaben zur Verwendung der GLOBAL-Klausel siehe Kapitel "PROCEDURE DIVISION", Abschnitt "Format 3 USE Anweisung mit GLOBAL-Angabe".

Allgemeine Regeln

1. Der Name einer Leiste darf nur in einem einzigen Prozedurvereinbarungskapitel angegeben werden.
2. Die INITIATE-, GENERATE-, EXIT PROGRAM, GOBACK-, CANCEL- und TERMINATE-Anweisungen dürfen nicht im Bereich der Prozedurvereinbarungen verwendet werden.
3. Eine USE BEFORE REPORTING-Prozedur darf weder den Wert irgend eines Gruppenwechseldatenfeldes noch einen der Werte der Subskripte ändern, die das Listenprogramm verwendet, um auf ein Gruppenwechseldatenfeld zuzugreifen.
4. Was die gegenseitigen Beziehungen zwischen USE BEFORE REPORTING-Prozeduren und dem Rest des Prozedurteiles betrifft, gelten dieselben Regeln wie für die anderen USE-Prozeduren.
5. Einige typische Anwendungen für USE BEFORE REPORTING-Prozeduren:
   a. Unterdrückung der Leistenerstellung:
      Wenn die Anweisung MOVE 1 TO PRINT-SWITCH in einer USEBEFORE REPORTING-Prozedur ausgeführt wird, dann erstellt das Listenprogrammsteuersystem die Leiste nicht, zu der die USE-Prozedur gehörte. Da das Listenprogramm sofort nach Unterdrückung der Leiste das PRINT-SWITCH-Sonderregister (siehe "PRINT-SWITCH-Sonderregister") auf Null setzt, muss es unmittelbar vor jeder gewünschten Unterdrückung neu auf 1 gesetzt werden, was bei jenen Leisten, die automatisch erstellt werden, nur mittels USE BEFORE REPORTING-Prozeduren möglich ist.
   b. Änderung des Inhaltes eines in einer SOURCE-Klausel angegebenen Datenfeldes:

      Bei der Druckaufbereitung einer Zeile mit einem druckfähigen Feld, dessen Beschreibung eine SOURCE-Klausel enthält, wird der Inhalt des in der SOURCE-Klausel angegebenen Feldes durch eine implizite MOVE-Anweisung in das druckfähige Feld transportiert. Durch eine USE BEFORE REPORTING-Prozedur ist es möglich, den Inhalt des Sendefeldes gerade vor der Ausführung der impliziten MOVE-Anweisung zu ändern.

c. Summenzählerrundung:
   Soll der Wert eines Summenzählers, bevor er zur Druckaufbereitung durch eine implizite MOVE-Anweisung in das zugehörige Druckfeld transportiert wird, noch gerundet werden, so ist dies nur mittels einer USE BEFORE REPORTING-Prozedur für den Gruppenfuß, in dem der Summenzähler definiert wurde, möglich.

d. Wenn in Abhängigkeit von der hierarchischen Stufe, auf der ein Gruppenwechsel stattfand, spezielle Aktionen vor Erstellung eines Seitenkopfes, Gruppenkopfes, Gruppenfußes oder Seitenfußes vorgenommen werden sollen, dann kann das nur - wegen der automatischen Erstellung - durch USE BEFORE REPORTING-Prozeduren für diese Leiste geschehen (siehe "SOURCE-Klausel" und "CBL-CTR-Sonderregister").

10.4 Sonderregister des Listenprogramms

In diesem Kapitel werden folgende Themen behandelt:

10.4.1 LINE-COUNTER-Sonderregister

LINE-COUNTER (Zeilenzähler) ist ein Sonderregister, das für jede Liste, die in der REPORT SECTION der DATA DIVISION beschrieben ist, automatisch angelegt wird. Die interne Beschreibung dieses Sonderregisters ist in der COBOL-Schreibweise mit PICTURE S999 USAGE COMPUTATIONAL gegeben.

Das Listenprogramm benutzt bei der Listenerzeugung stets das LINE-COUNTER-Sonderregister zur vertikalen Positionierung der einzelnen Leisten. Aus diesem Grunde darf das LINE-COUNTER-Sonderregister durch keine Anweisung der PROCEDURE DIVISION geändert werden.

In bezug auf die PAGE LIMIT-, NEXT GROUP- und LINE-Klausel wird das LINE-COUNTER-Sonderregister vom Listenprogramm automatisch abgefragt und erhöht (oder auf Null gesetzt: Seitenvorschub). Die INITIATE-Anweisung setzt das LINE-COUNTER-Sonderregister auf Null.

Das LINE-COUNTER-Sonderregister darf sowohl in der REPORT SECTION als auch in der PROCEDURE DIVISION verwendet werden. Was die REPORT SECTION betrifft, kann das LINE-COUNTER-Sonderregister nur in SOURCE-Klauseln gemäß

```
SOURCE IS LINE-COUNTER [OF report-name]
```

angegeben werden. Da das Listenprogramm das LINE-COUNTER-Sonderregister stets auf die aktuelle Zeile (Druckzeile oder NEXT GROUP-Positionierung) setzt, wird die Zeilennummer ausgewiesen, auf die das druckfähige Feld, dem die obige SOURCE-Klausel zugeordnet ist, gedruckt wird. Das LINE-COUNTER-Sonderregister darf in der PROCEDURE DIVISION jederzeit abgefragt werden, es enthält zu diesem Zeitpunkt den Wert, den ihm das Listenprogramm infolge der NEXT GROUP-Klausel der zuletzt (vor der Abfrage) erstellten Leiste zugewiesen hat (letzte Druckzeile der Leiste, wenn keine NEXT GROUP-Klausel vorlag).

Wenn in der REPORT SECTION mehrere Listen beschrieben sind, muss jeder LINE-COUNTER bei explizitem Ansprechen mit dem Listennamen gekennzeichnet werden.

10.4.2 PAGE-COUNTER-Sonderregister

PAGE-COUNTER (Seitenzähler) ist ein Sonderregister, das für jede Liste, die in der REPORT SECTION der DATA DIVISION beschrieben ist, automatisch angelegt wird. Die interne Beschreibung dieses Sonderregisters ist in der COBOL-Schreibweise mit PICTURE S9(7) USAGE COMPUTATIONAL-3 gegeben.

Wenn die INITIATE-Anweisung für eine Liste ausgeführt wird, setzt das Listenprogrammsteuersystem das PAGE-COUNTER-Sonderregister durch eine interne MOVE-Anweisung auf 1. Sobald das Listenprogramm einen Seitenvorschub absetzt - nach der Erstellung des Seitenfußes und vor Erstellung des Seitenkopfes - erhöht es den Inhalt des PAGE-COUNTER-Sonderregisters um 1.

Das PAGE-COUNTER Sonderregister ist beliebig zugänglich, d.h. sein Inhalt darf nicht nur abgefragt, sondern auch geändert werden. Meistens wird es dazu verwendet, im Seitenkopf oder Seitenfuß die Seitennummer auszudrucken. Zu diesem Zweck wird das PAGE-COUNTER-Sonderregister einem druckfähigen Feld der betreffenden Liste durch die SOURCE-Klausel

```
SOURCE IS PAGE-COUNTER [OF report-name]
```

zugeordnet.

Die Verwendung eines PAGE-COUNTERS muss mit dem Listennamen gekennzeichnet werden, wenn mehr als eine Liste in der REPORT SECTION beschrieben ist.

10.4.3 PRINT-SWITCH-Sonderregister

PRINT-SWITCH (Druckschalter) ist ein Sonderregister, das für ein Programm, in dessen DATA DIVISION die REPORT SECTION beschrieben ist, automatisch angelegt wird. Die interne Beschreibung dieses Sonderregisters ist in der COBOL-Schreibweise mit PICTURE S9 USAGE COMPUTATIONAL-3 gegeben. Pro Übersetzungseinheit wird nur ein solches Sonderregister angelegt.

Der Zweck dieses Sonderregisters ist es, die Erstellung (Druck) einer Leiste zu unterbinden. Dies wird dadurch erreicht, dass innerhalb einer USE BEFORE REPORTING-Prozedur für die zur Diskussion stehende Leiste die Anweisung MOVE 1 TO PRINT-SWITCH ausgeführt wird. Das Listenprogramm prüft unmittelbar nach der Ausführung aller zu einer Leiste gehörenden USE BEFORE REPORTING-Prozeduren, ob das PRINT-SWITCH-Sonderregister den Wert 1 enthält. Nach dieser Auswertung setzt das Listenprogramm das PRINT-SWITCH-Sonderregister auf Null, wodurch verhindert wird, dass ggf. ungewollt auch andere Leisten unterdrückt werden.

Die Unterdrückung der Leistenerstellung durch das Listenprogramm besteht darin, dass

- keine Seitenpositionierung der LINE-Klauseln der Leiste vorgenommen wird,
- keine Druckzeilen aufbereitet werden und
- keine Seitenpositionierung infolge einer NEXT GROUP-Klausel der Leiste stattfindet.

Die Unterdrückung der Leiste auf Grund des PRINT-SWITCH-Sonderregisters beinhaltet als Konsequenz der Punkte a) und c), dass das LINE-COUNTER-Sonderregister nicht verändert wird.

Das PRINT-SWITCH-Sonderregister sollte nur innerhalb von USE BEFORE REPORTING-Prozeduren der Prozedurvereinbarungen gesetzt werden. Wird das Sonderregister an einer anderen Stelle des Prozedurteils gesetzt, dann ist es allgemein unmöglich, vorherzusagen, welche Leiste unterdrückt wird.

10.4.4 CBL-CTR-Sonderregister

CBL-CTR (**C**ontrol **B**reak **L**evel **C**ounter) ist ein Sonderregister, das für jede Liste, die in der REPORT SECTION der DATA DIVISION beschrieben ist, automatisch angelegt wird. Die interne Beschreibung dieses Sonderregisters ist in der COBOL-Schreibweise mit PICTURE S999 USAGE COMPUTATIONAL gegeben.

Das Listenprogramm und das Programm verwenden dieses Sonderregister, um Informationen über den Gruppenwechsel zu übergeben bzw. um Aktionen in Abhängigkeit eines Gruppenwechsels zu unternehmen.

Das CBL-CTR-Sonderregister kann in Verbindung mit zwei verschiedenen Funktionen verwendet werden. Es steht zur Wahl, die Funktionen überhaupt nicht, eine von beiden oder beide Funktionen pro Liste zu verwenden. Der Einfachheit halber seien die beiden Funktionen als Funktion 1 und Funktion 2 bezeichnet. Das Benutzerprogramm teilt dem Listenprogramm mit, welche Funktionen es für eine Liste erwartet. Durch eine MOVE-Anweisung wird das CBL-CTR-Sonderregister mit dem Wert belegt, der die geforderte Funktion symbolisiert. Wertbelegung und Funktion hängen wie folgt zusammen:

Geforderte Funktion	**MOVE-Anweisung**
Funktion 1	MOVE 1 TO CBL-CTR.
Funktion 2	MOVE 2 TO CBL-CTR.
Funktion 1 und 2	MOVE 3 TO CBL-CTR.

Die entsprechende MOVE-Anweisung muss nach der INITIATE-Anweisung für die Liste, aber noch vor der (chronologisch) ersten GENERATE-Anweisung ausgeführt werden. Nur in diesem Zeitraum ist es dem Programmierer erlaubt, den Wert des CBL-CTR-Sonderregisters zu ändern.

Wenn in der REPORT SECTION mehrere Listen definiert sind, müssen alle Bezugnahmen auf CBL-CTR durch den Listennamen gekennzeichnet sein.

10.4.5 Funktion 1 des CBL-CTR-Sonderregisters

Funktion 1 wird durch die MOVE-Anweisung mit 1 (oder 3) in das CBL-CTR-Sonderregister hervorgerufen. Sie besteht aus zwei Teilen, deren Aktionen im Folgenden beschrieben sind.

a. **Zuweisung der alten Werte der Gruppenwechseldatenfelder in Gruppenfüße**

   Teil 1 der Funktion 1 stellt die alten Werte aus den Gruppenwechseldatenfeldern für SOURCE-Klauseln oder USE BEFORE REPORTING-Prozeduren von Gruppenfüßen zur Verfügung.

   SOURCE-Klauseln (oder USE BEFORE REPORTING-Prozeduren) von Gruppenfüßen, die sich auf Gruppenwechseldatenfelder beziehen, führen zu Schwierigkeiten. Zu dem Zeitpunkt, zu dem die Gruppenfüße erstellt werden, haben einige oder alle Gruppenwechseldatenfelder geänderte Werte. Da jedoch sinngemäß die Gruppenfüße, die infolge eines Gruppenwechsels erstellt werden, zu den Werten vor dem Gruppenwechsel gehören, ist es oft wünschenswert, dass die Werte vor dem Gruppenwechsel (alte Werte) gedruckt werden. Wenn die Funktion 1 des CBL-CTR-Sonderregisters angefordert wurde, werden diese alten Werte aus den Gruppenwechseldatenfeldern für SOURCE-Klauseln (oder USE BEFORE REPORTING-Prozeduren) von Gruppenfüßen geliefert.

   Wenn z.B. das Datenfeld MONATS-NAME als Gruppenwechseldatenfeld der Liste definiert ist und der Gruppenfuß MONATS-FUSS mit

```
01 MONATS-FUSS TYPE CONTROL FOOTING
                     MONATS-NAME LINE PLUS 1.
   02 COLUMN 10 PIC X(21) VALUE "***** DATEN-ENDE FUER"
   02 COLUMN 33 PIC X(9)  SOURCE MONATS-NAME.
```

   definiert ist, dann wünscht der Programmierer, dass, nachdem alle Daten für den JANUAR in Postenleisten gedruckt sind, der abschließende Gruppenfuß

   `***** DATEN-ENDE FUER JANUAR`

   ausgedruckt wird. Da gerade die Änderung des Feldes MONATS-NAME von JANUAR auf FEBRUAR die Erstellung des obigen Gruppenfußes bewirkte, würde aber FEBRUAR statt JANUAR, also der laufende (aktuelle) Inhalt gedruckt werden. Falls aber die Funktion 1 angefordert ist, wird statt FEBRUAR der gewünschte Monat JANUAR ausgedruckt.

b. **Anzeige der hierarchischen Stufe des Gruppenwechsels**

   im CBL-CTR-Sonderregister

   Teil 2 der Funktion 1 besteht darin, dass das Listenprogramm einen Wert, der die hierarchische Stufe des Gruppenwechsels anzeigt, im CBL-CTR-Sonderregister hinterlegt. Dies geschieht noch bevor eine USE BEFORE REPORTING-Prozedur zur Ausführung angesteuert wird. So eine Prozedur beginnt mit:

   `kapitelname SECTION. USE BEFORE REPORTING leistenname.`

   Die Anzeige der hierarchischen Stufe des aktuellen Gruppenwechsels wird dadurch erreicht, dass die Gruppenwechseldatenfelder von höchster hierarchischer Stufe aus durchnummeriert werden. FINAL wird dabei nicht berücksichtigt. So erhält der erste (ganz links stehende) Datenname der CONTROL-Klausel die Nummer eins, der nächste die Nummer zwei usw.

   Wenn z.B. in einer Listenerklärung die CONTROL-Klausel

   `CONTROLS ARE FINAL LAND KREIS STADT`

   vorliegt, dann wird LAND die Nummer 1, KREIS die Nummer 2 und STADT die Nummer 3 zugeordnet.

   Die Bedeutung der Werte des CBL-CTR-Sonderregisters in den USE-Prozeduren für den Seitenkopf und die Gruppenköpfe sind in Tabelle 41 aufgezeigt.

Wenn sich im obigen Beispiel der Wert im Gruppenwechseldatenfeld KREIS ändert, dann legt das Listenprogramm den Wert 2 im CBL-CTR-Sonderregister ab (vorausgesetzt, dass sich der Wert von LAND nicht gleichzeitig geändert hat).

Wert	Bedeutung
0	zeigt an, dass keine GENERATE-Anweisung ausgeführt wurde oder dass gerade die erste GENERATE-Anweisung ausgeführt wird
1-254	zeigt an, dass das Gruppenwechselfeld mit der vorliegenden Nummer eine Wertänderung erfuhr, und dass die laufenden Aktionen durch diesen Gruppenwechsel verursacht sind
255	zeigt an, dass kein Gruppenwechsel auftrat (kann nicht bei USE-Prozeduren von Gruppenköpfen eintreten)

Tabelle 41: Werte des CBL-CTR-Sonderregisters in USE-Prozeduren für Seitenköpfe und Gruppenköpfe

Die Bedeutung der Werte des CBL-CTR-Sonderregisters in den USE-Prozeduren für den Seitenfuß und die Gruppenfüße sind in Tabelle 42 aufgezeigt.

Wert	Bedeutung
0	zeigt an, dass die Abschlussphase vorliegt, d.h. dass die TERMINATE-Anweisung eben ausgeführt wird
1-254	zeigt an, dass das Gruppenwechseldatenfeld mit der vorliegenden Nummer eine Wertänderung erfuhr und dass die laufenden Aktionen durch diesen Gruppenwechsel verursacht sind
255	zeigt an, dass kein Gruppenwechsel vorliegt (kann nicht bei USE-Prozeduren von Gruppenfüßen auftreten)

Tabelle 42: Werte des CBL-CTR-Sonderregisters in USE-Prozeduren für Seitenfüße und Gruppenfüße

10.4.6 Funktion 2 des CBL-CTR-Sonderregisters

Die Funktion 2 wird durch die Belegung des CBL-CTR-Sonderregisters mit dem Wert 2 (oder 3) aufgerufen. Die bedingte Ausführung der NEXT GROUP-Klauseln der Gruppenfüße ist die Folge.

Normalerweise wird die NEXT GROUP-Klausel unmittelbar nach Erstellung jener Leiste, in deren Beschreibung sie angegeben ist, ausgeführt (vertikale Positionierung). Wenn infolge eines Gruppenwechsels mehrere Gruppenfüße hintereinander erzeugt werden, entstehen durch die NEXT GROUP-Klauseln dieser Gruppenfüße unbeabsichtigte Leerzeilen. Solche Leerzeilen sind eigentlich als Abstand des Gruppenfußes zu einem Gruppenkopf oder einer Postenleiste gedacht und nicht zu einem Gruppenfuß.

Wenn die Funktion 2 angefordert ist, dann sorgt das Listenprogramm dafür, dass nur die NEXT GROUP-Klausel desjenigen Gruppenfußes realisiert wird, der in der Folge der Gruppenfüße, die durch einen Gruppenwechsel geschlossen hintereinander erstellt werden, als letzte erzeugt wird. Vorhandene NEXT GROUP-Klauseln der übrigen Gruppenfüße einer geschlossenen Folge von Gruppenfüßen werden ignoriert. Dies gilt auch, wenn der letzte Gruppenfuß der Folge keine NEXT GROUP-Klausel aufweist.

11 XML

In diesem Kapitel werden folgende Themen behandelt:

11.1 Allgemeine Beschreibung

COBOL2000 unterstützt das Lesen und die Analyse von XML-Dokumenten. Dies ist auf zwei verschiedene Arten möglich:

- strukturorientiert, d.h. das XML-Dokument wird in einen Baum transformiert. Dieser Baum erlaubt das Positionieren auf Knoten und die Übertragung der Daten von (mehreren) Knoten in das COBOL-Programm.
- ereignisorientiert, d.h. das XML-Dokument wird rein sequenziell verarbeitet und jede erkannte syntaktische Einheit in Sonderregistern zur weiteren Verarbeitung bereitgestellt.

Die zur Verfügung stehenden Sprachmittel sind spezifisch für die gewählte Verarbeitungsart. Die eigentliche Analyse und Zerlegung von XML-Dokumenten erfolgt durch ein separates, Parser genanntes Open Source-Programmpaket. Für den Ablauf derartiger Programme müssen daher weitere Voraussetzungen erfüllt sein, siehe Handbuch "COBOL2000 Benutzerhandbuch" [1].

Strukturorientierte Verarbeitung

Diese Verarbeitungsart betrachtet ein XML-Dokument als Datei der neuen Dateiorganisationsform 'XML', das sowohl in einer traditionellen Datei, als auch im Speicher bereitgestellt werden kann. Sie braucht in beiden Fällen einen Dateisteuereintrag sowie eine Dateibeschreibung. Die Satzbeschreibungen der FD geben dabei nicht den Inhalt der Datei wieder, sondern beschreiben die Baumstruktur des XML-Dokuments und zwar als Ganzes oder nur die davon zu verarbeitenden Teile. Mit Hilfe neuer Klauseln in einer Datenerklä-rung wird die Zuordnung der Datenfelder zu den Knoten in der Baumdarstellung des XML-Dokuments festgelegt. Die eigentliche Verarbeitung, d.h. das Positionieren und das sequenzielle bzw. wahlfreie Lesen der Daten geschieht dann mit Hilfe der dafür erweiterten Anweisungen zur Dateiverarbeitung.

Ereignisorientierte Verarbeitung

Diese Verarbeitungsart kennt keine besonderen eigenen Sprachmittel zur Beschreibung eines XML-Dokuments. Die neue Anweisung XML PARSE analysiert und zerlegt ein im Speicher bereitgestelltes XML-Dokument sequenziell. Jede dabei erkannte syntaktische Einheit des Dokuments wird dem Anwenderprogramm in Sonderregistern zur weiteren Verarbeitung zur Verfügung gestellt. Es bleibt dem Anwenderprogramm überlassen, daraus ein Bild der übergreifenden Struktur des Dokuments zu gewinnen.

Zuordnung der neuen Sprachmittel zu den Verarbeitungsarten

Sprachmittel	Strukturorientiert	Ereignisorientiert
Dateisteuereintrag		
erweitertes ASSIGN Format	x	
Dateiorganisation XML	x	
Zugriffsmethode XML	x	
Dateierklärung, Satzbeschreibung		
IDENTIFIED-Klausel	x	
COUNT-Klausel	x	
Anweisungen		
OPEN, CLOSE	x	
OPEN DOCUMENT, CLOSE DOCUMENT	x	
START, READ	x	

XML PARSE		x
Ausnahmesituation EC-XML-CODESET-CONVERSION	x	
Sonderregister		x

Die Beschreibung der Sprachmittel basiert auf der Darstellung des XML-Dokuments als Baum und verwendet Begriffe, die bei der grafischen Darstellung von Baumstrukturen üblich sind (siehe auch Abschnitt „Sprachmittel zur Verarbeitung von XML").

11.2 Sprachelemente ENVIRONMENT DIVISION

Bei strukturorientierter Verarbeitung legt die ASSIGN-Klausel fest, ob das XML-Dokument im Speicher oder in einer physikalischen Datei bereitgestellt wird. Wenn das Dokument in einer physikalischen Datei steht, spielt deren BS2000/OSD-Zugriffsmethode keine Rolle. Die Satzstruktur der Datei ist jedoch für das Programm transparent in Form von End-ofLine-Zeichen. Diese Zeichen lassen sich aber auch optionengesteuert ausblenden (siehe dazu Handbuch "COBOL2000 Benutzerhandbuch" [1]).

11.2.1 FILE-CONTROL-Paragraf

Format XML-organisierte Datei

```
FILE-CONTROL.
  SELECT-Klausel
  ASSIGN-Klausel
  [ACCESS MODE-Klausel]
  [FILE STATUS-Klausel]
  ORGANIZATION-Klausel .
```

Syntaxregeln

1. Die SELECT-Klausel muss die erste Klausel im Dateisteuereintrag sein. Alle anderen Klauseln können in beliebiger Reihenfolge angegeben werden.
2. Wenn in der Dateierklärung die GLOBAL-Klausel angegeben ist, dürfen die in den Klauseln des FILE-CONTROL-Paragrafen angesprochenen Datenfelder nicht in einer LOCAL-STORAGE SECTION definiert sein.

11.2.2 SELECT-Klausel

Funktion

Die SELECT-Klausel dient dazu, einer Datei einen Namen zu geben.

Format

```
SELECT [OPTIONAL] dateiname-1
```

Syntaxregeln

1. dateiname-1 ist der Name, mit dem die Datei in der Übersetzungseinheit angesprochen wird. Innerhalb eines Programms, einer Factory, eines Objekts oder einer Methode darf dateiname-1 nur in einer SELECT-Klausel auftreten.
2. Für dateiname-1 muss es einen Dateibeschreibungseintrag (FD) in der DATA DIVISION des gleichen Programms, der gleichen Factory, des gleichen Objekts oder der gleichen Methode geben.

Allgemeine Regeln

1. Wenn sich die OPTIONAL-Angabe auf eine externe Datei bezieht, muss in allen Programmen, die diese externe Datei beschreiben, OPTIONAL angegeben werden.

11.2.3 ASSIGN-Klausel

Funktion

Die ASSIGN-Klausel stellt die Verbindung zwischen dem Dateisteuereintrag und einer Datei her. Die Datei kann einer physikalischen Datei zugeordnet werden oder im Speicher liegen.

Format

```
ASSIGN { TO {  literal-1
            | DATA datenname-1
            | datenname-2 LENGTH IS datenname-3 [FOR {ALPHANUMERIC | NATIONAL}]
           }

       | USING datenname-4

      }
```

Syntaxregeln

1. literal-1 muss ein alphanumerisches Literal sein und darf keine figurative Konstante sein.
2. datenname-1 muss ein alphanumerisches oder nationales Datenfeld sein.
3. datenname-2 muss ein Datenelement der Kategorie datenzeiger sein.
4. datenname-3 muss ein ganzzahliges, vorzeichenloses Datenelement sein, dessen Definition kein Maskenzeichen 'P' enthält.
5. datenname-1, datenname-2 und datenname-3 dürfen nicht in einer Datensatzerklärung einer XML-organisierten Datei definiert sein.
6. datenname-4 muss ein alphanumerisches Datenfeld sein und darf nicht innerhalb der FILE SECTION definiert sein.
7. Die DATA- bzw. LENGTH-Angabe darf nicht gemacht werden, wenn in der zugehörigen SELECT-Klausel OPTIONAL angegeben ist.
8. Wenn weder ALPHANUMERIC noch NATIONAL angegeben ist, wird ALPHANUMERIC angenommen.

Allgemeine Regeln

1. Wenn literal-1 oder datenname-4 angegeben ist, steht das XML-Dokument in einer physikalischen Datei. Andernfalls steht das XML-Dokument im Speicher.
2. literal-1 oder der Inhalt von datenname-4 gibt den Linknamen für die Datei an. Bei literal-1 werden nur die ersten 8Zeichen verwendet. Wenn das achte Zeichen ein Bindestrich (-) ist, wird dieser durch das Nummernzeichen (#) ersetzt.
3. Ein XML-Dokument im Speicher wird zugewiesen
   a. direkt durch das Datenfeld mit datenname-1, das das XML-Dokument enthält.
   b. indirekt durch ein Zeigerdatenfeld datenname-2, das auf einen Speicherbereich zeigt, der das XML-Dokument enthält. In diesem Fall gibt datenname-3 die Länge des Speicherbereichs in **Zeichen** an.
4. Die FOR-Angabe legt die Interpretation des Speicherbereichs, der das XML-Dokument enthält, als alphanumerische oder als nationale Zeichen fest.
5. Adresse und Länge des Speicherbereichs, der das XML-Dokument enthält, werden bei der OPEN-Anweisung ausgewertet. Der Inhalt des Speicherbereichs wird erst bei der OPEN DOCUMENT-Anweisung ausgewertet. Zeitlich nachfolgende Änderungen haben keinen Einfluss mehr auf die ausgewertete Adresse und Länge bzw. den Inhalt.

11.2.4 ACCESS MODE-Klausel

Funktion

Die ACCESS MODE-Klausel bestimmt die Art des Zugriffs auf das XML-Dokument.

Format

```
ACCESS MODE IS XML
```

Syntaxregeln

1. Die XML-Angabe darf nur für eine XML-organisierte Datei gemacht werden.

Allgemeine Regeln

1. Wenn die ACCESS MODE-Klausel fehlt und die Dateiorganisation XML ist, wird ACCESS XML angenommen.

11.2.5 FILE STATUS-Klausel

Funktion

Die FILE STATUS-Klausel gibt ein Datenfeld an, das den Zustand einer XML-Datei-Operation anzeigt. Ferner wird bei Angabe eines weiteren Datenfeldes zusätzlich ein Fehlerschlüssel zur Verfügung gestellt.

Format

```
FILE STATUS IS datenname-1 [datenname-2]
```

Syntaxregeln

1. datenname-1 und datenname-2 müssen in der WORKING-STORAGE SECTION, LOCAL-STORAGE SECTION oder LINKAGE SECTION der DATA DIVISION definiert sein.
2. datenname-1 muss ein zwei Zeichen langes alphanumerisches Datenfeld sein.
3. datenname-2 muss ein sechs Bytes langes Gruppenfeld mit folgenden Aufbau sein:

   ```
   01 datenname-2.
      02 datenname-2-1 PIC 9(2) COMP.
      02 datenname-2-2 PIC X(4).
   ```

Allgemeine Regeln

1. Wenn die FILE STATUS-Klausel angegeben ist, wird nach einer Ein-/Ausgabe-Anweisung für diejenige Datei, deren Dateisteuereintrag diese Klausel untergeordnet ist, der Ein-/Ausgabe-Zustand nach datenname-1 übertragen.
2. Falls angegeben, ist datenname-2 wie folgt belegt:
   a. Wenn der Inhalt von datenname-1 den Wert Null hat, ist der Inhalt von datenname-2 undefiniert.
   b. Wenn der Inhalt von datenname-1 einen von Null verschiedenen Wert hat, enthält datenname-2 den zusätzlichen Fehlerschlüssel. Der Wert 96 in datenname-2-1 zeigt an, dass es sich um den SIS-Code (POSIX) handelt. Der Wert 231 zeigt an, dass es sich um den CBX-Code (Fehlerschlüssel des XML-Parsers) handelt.
3. Das Kommando HELP SIS <Inhalt von datenname-2-2> bzw. HELP CBX <Inhalt von datenname-2-2> liefert nähere Informationen zum jeweiligen Fehlerschlüssel.
4. Übertragen wird der Ein-/Ausgabe-Zustand während der Ausführung jeder OPEN-, OPEN DOCUMENT-, CLOSE-, CLOSE DOCUMENT, READ- oder START-Anweisung, die sich auf die angegebene Datei bezieht, und zwar vor Ausführung jeder entsprechenden USE-Prozedur (siehe Abschnitt "Ein-/Ausgabe-Zustand für XML-Dateien").

11.2.6 ORGANIZATION-Klausel

Funktion

Die ORGANIZATION-Klausel legt den logischen Aufbau einer Datei fest.

Format

```
[ORGANIZATION IS] XML
```

Allgemeine Regeln

1. Die Angabe XML bedeutet, dass die Datei ein XML-Dokument enthält.

11.3 Sprachelemente DATA DIVISION

COBOL-Datenstruktur für ein XML-Dokument

Bei der strukturorientierten Verarbeitung wird die hierarchische Struktur eines XML-Dokuments durch das Stufenkonzept der Datensatzbeschreibungen in COBOL nachgebildet. Einem Knoten aus dem XML-Baum, d.h. einem Attribut oder dem (Start-)Tag eines Ele-ments im XML-Dokument, entspricht dabei eine Datenerklärung mit einer IDENTIFIED-Klausel. Ende-Tags spielen dabei keine Rolle, da sie sich implizit aus der Hierarchie ergeben.

Optional kann in der Datenstruktur zu jedem Knoten zusätzlich Folgendes beschrieben werden:

- der Name des Knotens,
- der Namensraum des Knotens,
- der Wert des Knotens,
- ggf. weitere im XML-Dokument untergeordnete Elemente, ebenfalls wieder mit einer IDENTIFIED-Klausel.

Alle solchen zusätzlichen Datenfelder müssen in der Datenstruktur dem Datenfeld direkt untergeordnet sein, das dem Knoten entspricht. Unter diesen zusätzlichen Datenfeldern darf es nur ein einziges geben, das weder eine IDENTIFIED-Klausel hat, noch von einer IDENTIFIED-Klausel angesprochen wird: das Datenfeld für den Wert. Wenn es sich bei diesem Datenfeld um das einzige Datenfeld handelt, das der Knoten-Beschreibung untergeordnet ist, kann es entfallen, und seine PICTURE-Klausel auch direkt bei der Knoten-Beschreibung angegeben werden.

Die Datenstrukturen der FD müssen nur diejenigen Knoten aus dem XML-Baum beschreiben, die das Programm verarbeiten will. Zu jedem beschriebenen Knoten muss jedoch auch dessen Vater beschrieben sein.

Zuordnung und Element-Positions-Vektor

Grundlage für die Verarbeitung eines XML-Dokuments ist die **Zuordnung** von Knoten des XML-Baums zu den Datenerklärungen in den Datensatzbeschreibungen der FD. Diese Zuordnung wird in der logischen Informationseinheit **Element-Positions-Vektor** (EPV) festgehalten. Zu jedem Datenfeld, das eine IDENTIFIED-Klausel angibt, ist darin vermerkt, welcher Knoten aus dem XML-Baum dem Datenfeld zugeordnet ist (d.h. es hat 'eine gültige Position'), bzw. die Tatsache, dass kein Knoten zugeordnet ist (d.h. die 'Position ist ungültig'). Darüber hinaus wird vermerkt, welche Anweisung die gültige Position erzeugt hat.

Die IDENTIFIED-Klausel in einer Datenerklärung bestimmt, welche Knoten aus dem XML-Baum dieser Datenerklärung prinzipiell zugeordnet werden können.

Die Zuordnung eines Knotens ist möglich, wenn alle folgenden Bedingungen erfüllt sind (zu den einzelnen Angaben siehe „IDENTIFIED-Klausel"):

- Die Art des Knotens (Element- bzw. Attribut-Knoten) im XML-Baum stimmt mit der Angabe (ELEMENT bzw. ATTRIBUTE) aus der IDENTIFIED-Klausel überein.
- Der lokale Name des Knotens im XML-Baum stimmt mit dem in der IDENTIFIED-Klausel angegebenen lokalen Namen überein:
  - Wenn der Name mittels BY angegeben ist, müssen beide Namen bis auf endständige Leerzeichen identisch sein.
  - Wenn der Name mittels USING angegeben ist, werden beliebige Namen aus dem XML-Baum als übereinstimmend angesehen.
- Der Namensraum des Knotens im XML-Baum stimmt mit dem in der IDENTIFIED-Klausel angegebenen Namensraum (NAMESPACE) überein. Dies gilt gleichermaßen für eine explizite NAMESPACE-Angabe, als auch für eine implizite, d.h. von übergeordneten Datenfeldern geerbte Angabe:
  - Wenn der Namensraum mittels BY angegeben ist, müssen beide Namen bis auf endständige Leerzeichen identisch sein.

- Wenn der Namensraum mittels USING angegeben ist, werden beliebige Namen aus dem XML-Baum als übereinstimmend angesehen (insbesondere auch der leere Namensraum).
- Wenn der Namensraum als NULL angegeben ist, muss auch der Knoten im XML-Baum einen leeren Namensraum haben.

Zuordnungsvorgang

Die neuen Anweisungen zur Verarbeitung eines XML-Dokuments führen Zuordnungen durch. Der **aktuelle Vorgang der Zuordnung** läuft dabei in folgenden Schritten ab:

1. Höchstens ein Knoten aus dem XML-Baum wird genau einem Datenfeld in der FD zugeordnet. Welche Knoten und welche Datenfelder für diesen ersten Schritt zu betrachten sind, hängt von der jeweiligen Anweisung ab.
2. Für jedes Datenfeld, dem ein Knoten aus dem XML-Baum zugeordnet werden konnte, werden alle ihm direkt untergeordnete Datenfelder mit IDENTIFIED-Klausel betrachtet, und es wird versucht, ihnen je ein Kind des zugeordneten Knotens zuzuordnen, beginnend mit dem ältesten, noch nicht zugeordneten Kind.
3. Für jedes bei den vorhergehenden Schritten betrachtete Datenfeld, dem kein Knoten zugeordnet werden konnte, sowie für alle einem solchen Datenfeld untergeordnete Datenfelder im EPV, wird die Position 'ungültig' vermerkt.

> **i** Bei einer Zuordnung werden keine Daten zwischen XML-Dokument und den Datenbeschreibungen im Programm übertragen.

11.3.1 Dateierklärung

Format XML-organisierte Datei

```
FD dateiname-1
  [EXTERNAL-Klausel]
  [GLOBAL-Klausel] .
```

Syntaxregeln

1. dateiname-1 muss mit dem Dateinamen aus einem Dateisteuereintrag übereinstimmen.
2. Die Reihenfolge der auf den Dateinamen folgenden Klauseln ist beliebig.
3. Der Dateierklärung müssen eine oder mehrere Datensatzbeschreibungen folgen.
4. Datensatzbeschreibungen von XML-organisierten Dateien dürfen nur Datenfelder der Kategorien alphabetisch, alphanumerisch, national oder numerisch enthalten.

11.3.2 EXTERNAL-Klausel

Funktion

Mit der EXTERNAL-Klausel kann eine XML-organisierte Datei als extern definiert werden. Auf externe XML-Dateien kann von jedem Programm, in dem sie beschrieben sind, zugegriffen werden.

Format

```
IS EXTERNAL
```

Syntaxregeln

1. Namen von externen Dateien dürfen maximal 30 Zeichen lang sein.
2. Die EXTERNAL-Klausel darf nicht für XML-organisierte Dateien angegeben werden, die in ihrer SELECT-Klausel die DATA- oder LENGTH-Angabe machen.

Allgemeine Regeln

1. Wenn eine Datei als extern definiert ist, sind die Datensätze dieser Datei implizit ebenfalls extern.
2. Als Namen für externe Dateien dürfen nicht verwendet werden:
   a. Externe Datensatznamen aus der WORKING-STORAGE SECTION von anderen Übersetzungseinheiten in der Ablaufeinheit,
   b. PROGRAM-ID-Namen der Ablaufeinheit, ausgenommen Programmnamen von inneren Programmen eines geschachtelten Programms,
   c. Namen, die in einer ENTRY-Anweisung als Einsprungpunkte verwendet werden,
   d. Namen, die eine Schnittstelle benennen (z.B. Laufzeitsystem-Namen).
3. Die FILE STATUS-Klausel wirkt für externe Dateien stets programmlokal, d.h. der Dateizustand wird nur von Eingabeoperationen in dem Programm versorgt, das eine entsprechende Angabe in der Dateibeschreibung enthält.
4. Eine externe XML-organisierte Datei muss in allen Übersetzungseinheiten, die auf sie zugreifen wollen, weitgehend gleich beschrieben sein. Die folgende Tabelle zeigt Art und Umfang der notwendigen Übereinstimmung.

Klausel/Angabe	In allen Programmen
Name der externen Datei	gleich in voller Länge (30 Zeichen)
ORGANIZATION-Klausel	XML
ACCESS MODE-Klausel	XML
Satzbeschreibungen in der FD	identisch, auch die Reihenfolge der Satzbeschreibungen. Überprüft wird jedoch nur die Übereinstim-mung der hierarchischen Struktur der IDENTIFIED-Angaben und der COUNT-Angaben in den Satzbeschreibungen, sowie die Übereinstimmung der Angaben ATTRIBUTE bzw. ELEMENT, BY bzw. USING in IDENTIFIED und IS bzw. USING in NAMESPACE.

5. In der IDENTIFIED-Klausel unter USING sowie BY bzw. IS angegebene Datenfelder, die nicht in einer Satzbeschreibung der FD enthalten sind, wirken für externe Dateien stets programmlokal, d.h. sie werden nur in dem Programm versorgt bzw. herangezogen, das die Eingabeoperation ausführt.

Ist eine Datei als extern definiert, so ist der zugehörige Dateiname nicht implizit ein globaler Name.

11.3.3 GLOBAL-Klausel

Format und Regeln siehe Abschnitt „GLOBAL-Klausel".

11.3.4 Datenerklärung

Funktion

Eine Datenerklärung beschreibt die Eigenschaften eines Knotens aus dem XML-Baum oder zusätzliche, mit dem Knoten verbundene Daten.

Format 1 für Beschreibung eines Knoten

```
stufennummer [datenname | FILLER]
             IDENTIFIER-Klausel
             [COUNT-Klausel]
             [GROUP-USAGE-Klausel]
             [JUSTIFIED-Klausel]
             [PICTURE-Klausel]
             [SIGN-Klausel]
             [SYNCHRONIZED-Klausel]
             [USAGE-Klausel]
             [VALUE-Klausel].
```

Format 2 für Beschreibung zusätzlicher Daten

```
stufennummer [datenname | FILLER]
             [GROUP-USAGE-Klausel]
             [JUSTIFIED-Klausel]
             [OCCURS-Klausel]
             [PICTURE-Klausel]
             [SIGN-Klausel]
             [SYNCHRONIZED-Klausel]
             [TYPE-Klausel]
             [USAGE-Klausel]
             [VALUE-Klausel].
```

Syntaxregeln für beide Formate

1. Die definierten Datenfelder müssen von der Kategorie alphabetisch, alphanumerisch, national oder numerisch sein.
2. Die Klauseln dürfen in beliebiger Reihenfolge geschrieben werden, abgesehen von der FILLER- bzw. datenname-Angabe, die unmittelbar auf die Stufennummer folgen müssen.

Syntaxregeln für Format 1

1. stufennummer muss eine Nummer von 01 bis 49 sein.

Syntaxregeln für Format 2

1. stufennummer muss eine Nummer von 02 bis 49 oder 88 sein. Für Stufennummer 88 gelten die Regeln für Bedingungsnamen (siehe "Formate der Datenerklärung").
2. Die DEPENDING ON-Angabe der OCCURS-Klausel ist nicht erlaubt.
3. Die OCCURS-Klausel darf nicht für Datenfelder angegeben werden, die einem Datenfeld, für das die IDENTIFIED-Klausel angegeben ist, direkt untergeordnet sind.

Alle im Folgenden nicht beschriebenen Klauseln werden als bekannt vorausgesetzt, siehe auch die entsprechende Beschreibungen ab "Datenerklärung".

11.3.5 COUNT-Klausel

Funktion

Die COUNT-Klausel erzeugt zu einem mittels IDENTIFIED-Klausel beschriebenen Datenfeld ein zusätzliches Datenelement, das anzeigt, ob eine READ-Anweisung dem Element bzw. Attribut, das durch die IDENTIFIED-Klausel beschrieben wird, einen Knoten aus dem XML-Dokument zuordnen konnte.

Format

```
COUNT IN datenname-1
```

Syntaxregeln

1. Die COUNT-Klausel darf nur für Datenfelder angegeben werden, die auch eine korrekte IDENTIFIED-Klausel haben.
2. datenname-1 darf nicht in der gleichen Quelleinheit definiert sein und muss ohne Qualifizierung eindeutig ansprechbar sein.

Allgemeine Regeln

1. Die COUNT-Klausel erzeugt zu der Datenerklärung, bei der sie angegeben ist, ein internes Datenelement mit dem Namen datenname-1.
2. datenname-1 hat die implizite Beschreibung PIC S9(9) USAGE COMP-5.
3. Wenn die Datenerklärung, auf die sich die COUNT-Klausel bezieht, einer Dateierklärung untergeordnet ist, für die die EXTERNAL-Klausel angegeben ist, ist auch das COUNT-Datenelement ein externes Datenfeld.
4. Wenn die Datenerklärung, auf die sich die COUNT-Klausel bezieht eine GLOBAL-Klausel enthält oder einer solchen Datei- bzw. Datenerklärung untergeordnet ist, ist auch datenname-1 ein globaler Name.
5. Nach einer erfolgreichen READ-Anweisung hat datenname-1 den Wert 1, wenn dem Eintrag, auf den sich die COUNT-Klausel bezieht, ein Knoten aus dem XML-Dokument zugeordnet wurde. Wenn kein Knoten zugeordnet wurde, hat datenname-1 den Wert 0.
6. Eine erfolglose READ-Anweisung ändert den Wert von datenname-1 nicht.

> **i** Das COUNT-Datenelement zeigt nicht die Anzahl von Wiederholungen eines Elements im XML-Dokument an.

11.3.6 IDENTIFIED-Klausel

Funktion

Die IDENTIFIED-Klausel legt fest, welche Element- bzw. Attributnamen aus dem XML-Dokument sich einem COBOL-Datenfeld zuordnen lassen. Die NAMESPACE-Angabe legt die dabei zu berücksichtigenden Namensräume fest.

Format

```
IDENTIFIED {BY {datenname-1 | literal-1} | USING datenname-2} IS {ATTRIBUTE |
ELEMENT}
       [ NAMESPACE  {IS {datenname-3 | literal-2 | NULL} | USING datenname-4}]
```

Syntaxregeln

1. Die IDENTIFIED-Klausel darf nur in Satzbeschreibungen von XML-organisierten Dateien angegeben werden.
2. Alle der Datenerklärung übergeordneten Datengruppen müssen ebenfalls eine IDENTIFIED-Klausel enthalten.
3. literal-1 und literal-2 müssen alphanumerische oder nationale Literale, jedoch keine figurativen Konstanten sein.
4. literal-1 muss ein gültiger XML-Name für ein Element bzw. Attribut sein.
5. literal-2 muss ein gültiger XML-Name für einen Namensraum sein.
6. datenname-1, datenname-2, datenname-3 und datenname-4 dürfen gekennzeichnet sein.
7. datenname-1, datenname-2, datenname-3 und datenname-4 müssen alphanumeri-sche oder nationale Datenfelder sein. Ihre Datenerklärung darf keine IDENTIFIED-Klausel angeben.
8. Wenn datenname-1, datenname-2, datenname-3 und datenname-4 innerhalb einer Satzbeschreibung einer XML-organisierten Datei definiert sind, müssen sie dem Eintrag mit der IDENTIFIED-Klausel, die sie anspricht, direkt untergeordnet sein.
9. Einer Datenerklärung mit ATTRIBUTE-Angabe dürfen keine Datenerklärungen mit einer IDENTIFIED-Klausel untergeordnet sein.
10. Einer Datenerklärung mit IDENTIFIED-Klausel darf höchstens ein Datenfeld direkt untergeordnet sein, das weder in der IDENTIFIED-Klausel angesprochen wird, noch selbst eine IDENTIFIED-Klausel enthält.
11. Die nachstehende Tabelle gibt an, welche Kombinationen von Angaben zum Element- bzw. Attributnamen und Angaben zum Namensraum innerhalb einer IDENTIFIED-Klausel erlaubt sind.

IDENTIFIED	**NAMESPACE**		
	IS NULL	**IS datenname-3 / literal-2**	**USING datenname-4**
BY datenname-1 / literal-1	erlaubt	erlaubt	verboten
USING datenname-2	erlaubt	verboten	erlaubt

Tabelle 43: Erlaubte Kombination von IDENTIFIED- und NAMESPACE-Angabe in einer IDENTIFIED-Klausel

12. Wenn einer Datengruppe mehrere Datenerklärungen direkt untergeordnet sind, die die ELEMENT-Angabe machen, unterliegen die möglichen Angaben zu IDENTIFIED und NAMESPACE weiteren Einschränkungen. Gleiches gilt für Datenerklärungen, die die ATTRIBUTE-Angabe machen. Die nachstehende Tabelle gibt an, welche Kombinationen bei je zwei Datenerklärungen erlaubt sind. Die grau

unterlegten Zeilen und Spalten stellen Kombinationen dar, die schon für eine einzelne Datenerklärung nicht erlaubt sind.

erste Datenerklärung: IDENTIFIED / NAMESPACE zweite Daten-erklärung: IDENTIFIED / NAMESPACE		BY NULL	USING NULL	BY IS	USING IS	BY USING	USING USING
BY	NULL	erlaubt	verboten	erlaubt			verboten
USING	NULL	verboten	verboten	verboten			verboten
BY	IS	erlaubt	verboten	erlaubt			verboten
USING	IS						
BY	USING						
USING	USING	verboten	verboten	verboten			verboten

Tabelle 44: Erlaubte Kombination von zwei Datenerklärungen mit IDENTIFIED- und NAMESPACE-Angaben

13. Wenn einer Datengruppe mehrere Datenerklärungen direkt untergeordnet sind, die alle die gleiche explizite oder implizite NAMESPACE-Angabe machen, dann müssen
    a. für alle, die explizit oder implizit die ELEMENT-Angabe machen, deren Werte von literal-1 eindeutig sein.
    b. für alle, die die ATTRIBUTE-Angabe machen, deren Werte von literal-1 eindeutig sein.

    Dabei spielen für die Eindeutigkeit weder die Darstellung als alphanumerisches oder nationales Literal noch endständige Leerzeichen eine Rolle.
14. Wenn ATTRIBUTE nicht angegeben ist, wird ELEMENT angenommen.
15. Die IDENTIFIED-Klausel darf nicht innerhalb einer Typdeklaration verwendet werden.

Allgemeine Regeln

1. Die IDENTIFIED-Klausel beschreibt, welche Knoten aus dem XML-Dokument dem Datenfeld zugeordnet werden sollen:
    a. Wenn ELEMENT angegeben ist, werden nur Element-Knoten zugeordnet.
    b. Wenn ATTRIBUTE angegeben ist, werden nur Attribut-Knoten zugeordnet.
2. Bei Angabe von BY legen literal-1 bzw. der Inhalt von datenname-1 den Namen fest, den der zugeordnete Knoten haben soll. Dabei spielen weder die Darstellung als alphanumerisches oder nationales Literal bzw. Datenfeld noch endständige Leerzeichen eine Rolle.
3. Bei Angabe von IS legen literal-2 bzw. der Inhalt von datenname-3 den Namensraum fest, den der zugeordnete Knoten haben soll. Dabei spielen weder die Darstellung als alphanumerisches oder nationales Literal bzw. Datenfeld noch endständige Leerzeichen eine Rolle.
4. Bei Angabe von USING dürfen die zugeordneten Knoten beliebige Namen haben bzw. in beliebigen Namensräumen liegen. Bei erfolgreichem Lesen werden die Namen solcher Knoten in datenname-2 bzw. der zugehörige Namensraum in datenname-4 übertragen.
5. Eine Zuordnung eines Knotens zu einem Datenfeld ist nur möglich, wenn allen ihm in der Struktur übergeordneten Datenfeldern auch ein Knoten zugeordnet ist.
6. Die NAMESPACE-Angabe legt den Namensraum fest für einen Eintrag sowie für alle ihm untergeordneten Datenfelder, die eine IDENTIFIED-Klausel mit expliziter oder impliziter ELEMENT-Angabe haben.
7. Wenn ein untergeordnetes Datenfeld ebenfalls eine NAMESPACE-Angabe hat, hat diese Angabe Vorrang vor einer NAMESPACE-Angabe der übergeordneten Datengruppe.
8. NAMESPACE NULL bedeutet eine leere Zeichenfolge als Namensraum. Wenn die NAMESPACE-Angabe fehlt, und wenn auch keine übergeordnete Datengruppe eine NAMESPACE-Angabe hat, wird NAMESPACE NULL angenommen.

9. Wenn eine OPEN DOCUMENT-, READ- oder START-Anweisung eine Datengruppe anspricht, muss für diese und alle ihr untergeordneten Datengruppen Folgendes erfüllt sein:

   Wenn einer Datenerklärung, die der Datengruppe direkt untergeordnet ist, ein Knoten aus dem XML-Baum zugeordnet werden kann (siehe "Zuordnen von Knoten"), muss diese Zuordnung eindeutig sein. D.h. es darf keine weitere, der Da-tengruppe direkt untergeordnete Datenerklärung geben, der der Knoten aus dem XML-Baum ebenfalls zugeordnet werden könnte.

   Dabei spielen für die Eindeutigkeit weder die Beschreibung der Namen der Knoten als alphanumerisches oder nationales Literal bzw. Datenfeld eine Rolle, noch endständige Leerzeichen. Wenn die Eindeutigkeit nicht gegeben ist, ist die Anweisung nicht erfolgreich und der Ein-/Ausgabe-Zustand der Datei wird auf 4C gesetzt.

i
- Es ist immer die komplette Bezeichnung für den Namensraum, der Uniform Resource Identifier (URI), bereitzustellen, bzw. wird er zurückgeliefert. Die in XML verwendeten Präfixe sind in COBOL **nicht** sichtbar.
- Namensraum**deklarationen** haben im XML-Dokument zwar die Form eines Attributs, sie werden jedoch **nicht** als Attribute geliefert, selbst wenn sie in einer IDENTIFIED-Klausel entsprechend beschrieben sind.

11.4 Sprachelemente PROCEDURE DIVISION

Strukturorientierte Verarbeitung

Die Verarbeitung einer XML-organisierten Datei beginnt mit der OPEN-Anweisung und endet mit der CLOSE-Anweisung. Eine solche Datei enthält im Prinzip mehrere XML-Dokumente. Derzeit werden jedoch nur Dateien unterstützt, die genau ein XML-Dokument enthalten. Die Verarbeitung dieses Dokuments, insbesondere die Erzeugung der Baumdarstellung des XML-Dokuments, beginnt mit der OPEN DOCUMENT-Anweisung und endet mit der CLOSE DOCUMENT-Anweisung. Positionieren in dem XML-Baum geschieht mit der START-Anweisung. Die READ-Anweisung überträgt Daten von einem oder mehreren Knoten des XML-Baums in das Programm. Die OPEN DOCUMENT-Anweisung mit der AT-Angabe (gepaart mit einer CLOSE DOCUMENT-Anweisung) erlaubt es einen Teilbaum wie ein eigenständiges Dokument zu behandeln.

Ereignisorientierte Verarbeitung

Das zu bearbeitende XML-Dokument muss im Arbeitsspeicher bereitgestellt werden. Ein Dokument, falls erforderlich, aus einer Datei einzulesen, oder den Zeichensatz zu konvertieren usw., bleibt dem Anwenderprogramm überlassen. Die Anweisung XML PARSE bearbeitet und zerlegt ein gesamtes XML-Dokument dann schrittweise. Sobald eine syntaktische Einheit des Dokuments erkannt wurde, geht die Kontrolle vorübergehend an eine vom Anwender bereitzustellende Routine zu deren Behandlung – Verarbeitungsprozedur genannt – über (vergleichbar den Ein-/Ausgabeprozeduren einer SORT-Anweisung). In der Verarbeitungsprozedur stehen die erkannte syntaktische Einheit und zusätzliche, damit verbundene Daten in Sonderregistern für eine weitere Verarbeitung zur Verfügung. Nach Ende der Verarbeitungsprozedur wird die XML-Anweisung fortgesetzt. Dieser Wechsel zwischen XML-Anweisung und Verarbeitungsprozedur wiederholt sich solange, bis das ganze XML-Dokument verarbeitet worden ist, oder auf Grund von Fehlern keine weitere Verarbeitung möglich ist, oder das Anwenderprogramm auf die weitere Zerlegung des XML-Dokuments verzichtet.

Datenübertragung bei strukturorientierter Verarbeitung

Alles verhält sich so, als ob das XML-Dokument in der Baumform in UTF-16 dargestellt wäre, unabhängig davon, wie es tatsächlich dargestellt ist. Einzelne Daten im XML-Baum werden XML-Wert genannt und sind als Datenelemente der Kategorie national anzunehmen. Die Übertragung von Daten aus dem XML-Baum zu den Empfangsfeldern im COBOL-Programm umfasst daher ggf. auch Konvertierungen. Die Art der Übertragung wird durch die Länge des XML-Werts und die Kategorie des Empfangsfeldes bestimmt.

- Länge des XML-Werts = 0:

  Das Empfangsfeld wird initialisiert entsprechend der Anweisung

  ```
  INITIALIZE empfangsfeld TO DEFAULT
  ```

- Länge des XML-Werts > 0:
  - Die Kategorie des Empfangsfeldes ist national:
    Der XML-Wert wird entsprechend den Regeln für eine MOVE-Anweisung übertragen.
  - Die Kategorie des Empfangsfeldes ist alphabetisch oder alphanumerisch:
    Der XML-Wert wird entsprechend der Standard-Funktion DISPLAY-OF (ohne zweites Argument) konvertiert, und das Resultat entsprechend den Regeln für eine MOVE-Anweisung übertragen.
  - Die Kategorie des Empfangsfeldes ist numerisch:
    Aus dem XML-Wert wird entsprechend der Standard-Funktion NUMVAL-C (ohne zweites Argument) ein numerischer Wert gebildet, und dieser entsprechend den Regeln für eine MOVE-Anweisung übertragen.

Ausnahmesituationen bei strukturorientierter Verarbeitung

Zur Erzeugung der Baumdarstellung (bei der OPEN DOCUMENT-Anweisung ohne die AT-Angabe) wird das in einer Datei oder im Speicher bereitgestellte XML-Dokument in nationale Zeichendarstellung konvertiert. Wenn

sich dabei Zeichen finden, die keine nationale (UTF-16-)Darstellung besitzen, tritt die Ausnahmesituation EC-XML-CODESET-CONVERSION auf. Diese Ausnahmesituation ist NON-FATAL. An die Stelle der betroffenen Zeichen tritt im XML-Baum das Ersatzzeichen.

Nach der Übertragung eines XML-Werts in alphanumerische oder alphabetische Empfangsfelder kann die Ausnahmesituation EC-DATA-CONVERSION auftreten. An die Stelle von nicht konvertierbaren Zeichen ist dabei das Ersatzzeichen getreten. Wenn es für diese Ausnahmesituation eine USE-Prozedur gibt, die mit RESUME AT NEXT STATEMENT verlassen wird, wird der Programmlauf mit der Übertragung des nächsten XML-Werts fortgesetzt, sofern durch die READ-Anweisung mehrere XML-Werte zu übertragen sind und noch nicht alle Übertragungen erfolgt sind.

Schachtelung von Anweisungen zur Verarbeitung von XML-Dokumenten

Sowohl für die Anweisungen zur strukturorientierten Verarbeitung, als auch für die Anweisung zur ereignisorientierten Verarbeitung von XML-Dokumenten gilt folgende Einschränkung:

Das Durchlaufen von USE-Prozeduren bzw. Verarbeitungsprozeduren **während** der Ausführung von Anweisungen zur Verarbeitung eines XML-Dokuments – insbesondere können das die OPEN-Anweisung (siehe "OPEN-Anweisung"), die READ-Anweisung (siehe "READ-Anweisung") und die XML PARSE-Anweisung (siehe "XML-Anweisung") sein – darf nicht dazu führen, dass eine weitere derartige Anweisung ausgeführt wird.

11.4.1 CLOSE-Anweisung

Funktion

Die CLOSE-Anweisung beendet die Verarbeitung einer XML-organisierten Datei.

Format

```
CLOSE {dateiname-1} ...
```

Syntaxregeln

1. Neben Namen von XML-organisierten Dateien dürfen auch Namen von Dateien mit anderer Organisation angegeben werden.

Allgemeine Regeln

1. Für nicht XML-organisierte Dateien gelten die Regeln der CLOSE-Anweisung ab "CLOSE-Anweisung".
2. Wenn die Datei dateiname-1 bereits geschlossen ist, ist die CLOSE-Anweisung nicht erfolgreich und der Ein-/Ausgabe-Zustand der Datei wird auf 42 gesetzt.
3. Wenn mehrere Dateinamen angegeben sind, ist die Wirkung die gleiche, wie wenn für jeden Dateinamen eine eigene CLOSE-Anweisung gegeben worden wäre.
4. Wenn für dateiname-1 die Verarbeitung eines XML-Dokuments eröffnet, jedoch noch nicht mit einer CLOSE DOCUMENT-Anweisung beendet wurde, wird im Rahmen der CLOSE-Anweisung auch eine implizite CLOSE DOCUMENT-Anweisung ausgeführt.

11.4.2 CLOSE DOCUMENT-Anweisung

Funktion

Die CLOSE DOCUMENT-Anweisung beendet die Verarbeitung eines ganzen XML-Dokuments oder eines Teildokuments und passt vorhandene Zuordnungen von Elementen bzw. Attributen aus dem XML-Dokument zu Datenfeldern an.

Format

```
CLOSE DOCUMENT dateiname-1
```

Syntaxregeln

1. dateiname-1 muss der Name einer XML-organisierten Datei sein.

Allgemeine Regeln

1. Wenn die Datei dateiname-1 nicht geöffnet ist, ist die CLOSE DOCUMENT-Anweisung nicht erfolgreich, und der Ein-/Ausgabe-Zustand der Datei wird auf 4B gesetzt.
2. Wenn in der Datei dateiname-1 kein XML-Dokument geöffnet ist, ist die CLOSE DOCUMENT-Anweisung nicht erfolgreich, und der Ein-/Ausgabe-Zustand der Datei wird auf 4D gesetzt.
3. Das Verhalten der Anweisung hängt von der STACK-Angabe ab, die bei der letzten für dateiname-1 erfolgreich ausgeführten OPEN DOCUMENT-Anweisung gemacht wurde:
   a. STACK angegeben:
   Der EPV wird auf den Zustand zurückgesetzt, den er vor der OPEN DOCUMENT-Anweisung hatte. Die interne Darstellung des XML-Dokuments bleibt unverändert erhalten.
   b. STACK nicht angegeben:
   Alle Positionen im EPV werden auf 'ungültig' gesetzt und die interne Darstellung des XML-Dokuments wird freigegeben.

> **i** Die CLOSE DOCUMENT-Anweisung beendet nur die Verarbeitung eines XML-Dokuments, jedoch nicht die Verarbeitung der Datei.

11.4.3 OPEN-Anweisung

Funktion

Die OPEN-Anweisung eröffnet eine XML-organisierte Datei für die Verarbeitung.

Format

```
OPEN INPUT {dateiname-1} ... [ END-OPEN ]
```

Syntaxregeln

1. Neben Namen von XML-organisierten Dateien dürfen auch Namen von Dateien mit anderer Organisation angegeben werden.

Allgemeine Regeln

1. Für nicht XML-organisierte Dateien gelten die Regeln der OPEN-Anweisung ab "OPEN-Anweisung".
2. Wenn die Datei dateiname-1 bereits geöffnet ist, ist die OPEN-Anweisung nicht erfolgreich, und der Ein-/Ausgabe-Zustand der Datei wird auf 41 gesetzt.
3. Wenn mehrere Dateinamen angegeben sind, ist die Wirkung die gleiche, wie wenn für jeden Dateinamen eine eigene OPEN-Anweisung gegeben worden wäre.
4. Nach einer erfolgreichen OPEN-Anweisung ist die durch dateiname-1 angegebene Datei verfügbar und befindet sich in geöffnetem Zustand. Wenn die Datei im Speicher liegt, stellt die OPEN-Anweisung die Verbindung zum Speicher her und wertet ggf. in der ASSIGN-Klausel angegebene Datenfelder für Zeiger und Länge aus (siehe Abschnitt „ASSIGN-Klausel").
5. Nach einer erfolgreichen OPEN-Anweisung zeigt der Dateipositionsindikator auf das erste XML-Dokument, sofern die Datei nicht leer ist. Wenn die Datei leer ist, oder wenn im Dateisteuereintrag OPTIONAL angegeben ist, und die physikalische Datei nicht existiert, zeigt der Dateipositionsindikator 'dateiende' an.

11.4.4 OPEN DOCUMENT-Anweisung

Funktion

Die OPEN DOCUMENT-Anweisung eröffnet die Verarbeitung eines XML-Dokuments. Sie erzeugt die interne Darstellung des XML-Dokuments und führt eine erste Zuordnung von Elementen bzw. Attributen aus dem XML-Dokument zu Datenfeldern in der FD durch.

Format

```
OPEN DOCUMENT dateiname-1 [ AT datenname-1 [STACK] ]
  [ RETURNING bezeichner-1 ]
  [ AT END unbedingte-anweisung-1 ]
  [ NOT AT END unbedingte-anweisung-2 ]
  [ END-OPEN ]
```

Syntaxregeln

1. dateiname-1 muss der Name einer XML-organisierten Datei sein.
2. datenname-1 darf gekennzeichnet sein.
3. datenname-1 muss mit einer IDENTIFIED-Klausel mit expliziter oder impliziter ELEMENT-Angabe beschrieben und der Datei dateiname-1 untergeordnet sein.
4. bezeichner-1 muss ein alphanumerisches oder nationales Datenfeld sein.
5. Wenn datenname-1 angegeben ist, dürfen die AT END- bzw. NOT AT END-Angaben nicht gemacht werden.

Allgemeine Regeln

1. Wenn für dateiname-1 die Verarbeitung eines XML-Dokuments eröffnet, jedoch noch nicht mit einer CLOSE DOCUMENT-Anweisung beendet wurde, wird vor der Ausführung einer OPEN DOCUMENT-Anweisung ohne Angabe von datenname-1 eine implizite CLOSE-Anweisung ausgeführt.
2. Wenn die Datei dateiname-1 nicht geöffnet ist, ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4B gesetzt.
3. Wenn datenname-1 angegeben, jedoch in der Datei dateiname-1 kein XML-Dokument geöffnet ist, ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/AusgabeZustand der Datei wird auf 4D gesetzt.
4. Wenn das zu öffnende XML-Dokument nicht wohlgeformt ist, ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand wird auf 3A gesetzt.
5. Wenn der Zeichensatz, in dem das XML-Dokument dargestellt ist, nicht ermittelt werden konnte (siehe Handbuch “COBOL2000 Benutzerhandbuch“ [1], Abschnitt 10.4 “Zeichensatzerkennung (Friedrichscher Vermutungsalgorithmus)“), ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand wird auf 3D gesetzt.
6. Wenn datenname-1 keine gültige Position hat, ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 46 gesetzt.
7. Wenn der Dateipositionsindikator 'dateiende' anzeigt, ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 10 gesetzt.
8. Wenn die Anweisung nicht erfolgreich ist, werden alle Positionen im EPV auf 'ungültig' gesetzt.
9. Wenn datenname-1 nicht angegeben ist, wird die interne Darstellung des XML-Dokuments erzeugt und der Dateipositionsindikator auf 'dateiende' gesetzt.
10. Wenn datenname-1 angegeben ist, wird die folgende Verarbeitung des XML-Dokuments eingeschränkt auf den Unterbaum, dessen Wurzel der datenname-1 zugeordnete Knoten ist.

   a. Wenn STACK angegeben ist, wird der Zustand des EPV von vor der OPEN DOCUMENT-Anweisung sichergestellt, so dass er durch die nächste CLOSE DOCUMENT-Anweisung für dateiname-1 wieder hergestellt werden kann. Vor Ausführung einer solchen CLOSE DOCUMENT-Anweisung kann jedoch nicht auf den gesicherten EPV-Zustand zugegriffen werden.
   b. Wenn STACK nicht angegeben ist, geht der Zustand des EPV von vor der OPEN DOCUMENT-Anweisung verloren.

11. Die OPEN DOCUMENT-Anweisung führt eine Zuordnung aus (siehe Abschnitt „Sprachelemente DATA DIVISION"). Sie überträgt jedoch – abgesehen von der RETURNING-Angabe – keine Daten.
12. Wenn die Namen in den bei der Zuordnung wirkenden IDENTIFIED-Klauseln nicht eindeutig sind (siehe Abschnitt „IDENTIFIED-Klausel“, allgemeine Regel 9 auf "IDENTIFIED-Klausel"), ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4C gesetzt.
13. Wenn sich einzelne Attribut- bzw. Elementnamen aus den bei der Zuordnung wirkenden IDENTIFIED-Klauseln nicht im Zeichensatz UTF-16 darstellen lassen, ist die OPEN DOCUMENT-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4E gesetzt.
14. Bei der Zuordnung sind im ersten Schritt auf der COBOL-Seite alle Datenfelder mit Stufennummer 01 in den Satzbeschreibungen von dateiname-1 zu betrachten. Außerdem ist im XML-Baum folgender Knoten zu betrachten:
    a. falls datenname-1 nicht angegeben ist: der Wurzelknoten des gesamten XML-Baums.
    b. falls datenname-1 angegeben ist: der dem datenname-1 zugeordnete Knoten aus dem XML-Baum.
15. Wenn RETURNING angegeben ist, wird in das Datenfeld bezeichner-1 übertragen:
    a. ohne AT-Angabe der Name des Wurzel-Elements des XML-Dokuments.
    b. mit AT-Angabe der Name des Elements im XML-Dokument das datenname-1 zugeordnet ist.

    > **i** Diese Übertragung erfolgt bei jeder erfolgreichen OPEN DOCUMENT-Anweisung, selbst wenn das Wurzelelement keinem Datenfeld in der COBOL-Beschreibung zugeordnet werden kann.

16. Die Fortsetzung des Ablaufs der OPEN DOCUMENT-Anweisung hängt davon ab, ob AT END oder NOT AT END angegeben ist (siehe Abschnitt „Ende-Bedingung").

11.4.5 READ-Anweisung

Funktion

Ausgehend von einer bereits existierenden Zuordnung positioniert die READ-Anweisung auf Elemente bzw. Attribute aus dem XML-Dokument, ordnet sie den Datenfeldern in der FD zu und überträgt damit verbundene Werte.

Format

```
READ dateiname-1 {ATTRIBUTE | [ONLY] ELEMENT} datenname-1
   [AT END unbedingte-anweisung-1]
   [NOT AT END unbedingte-anweisung-2]
   [END-READ]
```

Syntaxregeln

1. dateiname-1 muss der Name einer XML-organisierten Datei sein.
2. datenname-1 darf gekennzeichnet sein.
3. datenname-1 muss mit einer IDENTIFIED-Klausel beschrieben und der Datei dateiname-1 untergeordnet sein.
4. Wenn in der READ-Anweisung ATTRIBUTE angegeben ist, muss die IDENTIFIED-Klausel in der Datenerklärung von datenname-1 die ATTRIBUTE-Angabe haben.
5. Wenn in der READ-Anweisung ELEMENT angegeben ist, muss die IDENTIFIED-Klausel in der Datenerklärung von datenname-1 explizit oder implizit die ELEMENT-Angabe haben.

Allgemeine Regeln

1. Wenn die Datei dateiname-1 nicht geöffnet ist, ist die READ-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 47 gesetzt.
2. Wenn in der Datei dateiname-1 kein XML-Dokument geöffnet ist, ist die READ-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4D gesetzt.
3. Wenn datenname-1 keine gültige Position hat, ist die READ-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 46 gesetzt.
4. Die READ-Anweisung führt zuerst eine Zuordnung aus (siehe Abschnitt "Sprachelemente DATA DIVISION"), und überträgt anschließend ggf. Daten. Wenn im **ersten Schritt** dieses Zuordnungsvorgangs kein Knoten aus dem XML-Baum zugeordnet werden kann, ist die READ-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand wird auf 10 gesetzt, d.h. eine Ende-Bedingung existiert.
5. Wenn die Namen in den bei der Zuordnung wirkenden IDENTIFIED-Klauseln nicht eindeutig sind (siehe "IDENTIFIED-Klausel", allgemeine Regel 9 auf "IDENTIFIED-Klausel"), ist die READ-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4C gesetzt.
6. Wenn sich einzelne Attribut- bzw. Elementnamen aus den bei der Zuordnung wirkenden IDENTIFIED-Klauseln nicht im Zeichensatz UTF-16 darstellen lassen, ist die READ-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4E gesetzt.
7. Bei der Zuordnung ist im ersten Schritt auf der COBOL-Seite nur das Datenfeld datenname-1 zu betrachten; dazu dann im XML-Baum folgende Knoten:
   a. falls datenname-1 eine IDENTIFIED **BY**-Klausel mit der ATTRIBUTE-Angabe hat: der datenname-1 zugeordnete Knoten aus dem XML-Baum und **alle** seine Geschwister. Da in XML-Dokumenten Attribute eines Elements eindeutig sein müssen, spielt hier die Reihenfolge, in der diese Knoten zu betrachten sind, keine Rolle.
   b. andernfalls:

- wenn der Knoten durch eine READ-Anweisung zu datenname-1 zugeordnet wurde: alle **jüngeren** Geschwister dieses Knotens in der Reihenfolge vom ältesten jüngeren Geschwister bis zum jüngsten jüngeren Geschwister.
- wenn der Knoten durch eine OPEN DOCUMENT- oder START-Anweisung zu datenname-1 zugeordnet wurde: dieser Knoten selbst und anschließend alle seine jüngeren Geschwister in der Reihenfolge vom ältesten jüngeren Geschwister bis zum jüngsten jüngeren Geschwister.

8. Wenn ONLY angegeben ist, sind für nachfolgende READ-Anweisungen die gemachten Zuordnungen für alle datenname-1 untergeordneten Datenfelder, die explizit oder im-plizit die ELEMENT-Angabe haben, so zu interpretieren, als ob sie durch eine START-Anweisung erfolgt wären (und nicht durch die READ-Anweisung).
9. Wenn es in dem Teilbaum des XML-Dokuments, dessen Wurzel derjenige Knoten ist, der datenname-1 zugeordnet wurde, auch Knoten gibt, die keinem Datenfeld aus dem COBOL-Programm zugeordnet werden konnten, und wenn ONLY nicht angegeben ist, ist die READ-Anweisung erfolgreich. Der Ein-/Ausgabe-Zustand wird auf 08 gesetzt.
10. Als Ergebnis der READ-Anweisung bleiben Datenfelder in den Satzbeschreibungen von dateiname-1 entweder unverändert oder werden initialisiert, oder die Anweisung überträgt Daten aus dem XML-Baum. Dies ist im Einzelfall abhängig von der aktuell durchgeführten Zuordnung und der Lage der Datenfelder in den Satzbeschreibungen relativ zu datenname-1:

Datenfelder	**datenname-1 wurde ein Knoten zugeordnet**	**datenname-1 wurde kein Knoten zugeordnet**
datenname-1	Übertragung	unverändert
Datenfeld ist datenname-1 untergeordnet; ein Knoten wurde zugeordnet.	Übertragung	–
Datenfeld ist datenname-1 untergeordnet; es wurde kein Knoten zugeordnet.	Initialisierung	unverändert
Andere Datenfelder	unverändert	unverändert

11. Wenn ONLY angegeben ist, beschränkt sich die Übertragung auf Daten zu datenname-1 und allen ihm direkt untergeordneten Datenfeldern, die in ihrer IDENTIFIED-Klausel die ATTRIBUTE-Angabe haben. Alle anderen Datenfelder bleiben unverändert.
12. Übertragung' bzw. 'Initialisierung' im Zusammenhang mit einem Datenfeld bedeutet die Zuweisung von Werten an folgende Datenfelder (sofern diese definiert sind):

	Übertragung	**Initialisierung**
Datenfeld aus der USING-Angabe der IDENTIFIED-Klausel	Name des Knotens aus dem XML-Baum	Leerzeichen
Datenfeld aus der NAMESPACE USING-Angabe	Wert des Namensraums aus dem XML-Baum	Leerzeichen
Datenfeld für den Wert	Wert des Knotens aus dem XML-Baum	INITIALIZE datenfeld TO DEFAULT
Datenfeld aus der COUNT-Klausel	1	0

Daten aus dem XML-Baum werden für die Übertragung als XML-Wert aufgefasst und entsprechend behandelt (siehe "Datenübertragung bei strukturorientierter Verarbeitung").

Das Datenfeld für den Wert eines Knoten entspricht dem Datenfeld mit der IDENTIFIED-Klausel, sofern dieses elementar ist. Andernfalls muss es dem Datenfeld mit der IDENTIFIED-Klausel direkt untergeordnet

sein und darf als einziges der direkt untergeordneten Datenfelder weder eine IDENTIFIED-Klausel haben, noch durch eine IDENTIFIED-Klausel angesprochen werden.

i
- Mixed Content-Werte sind zu einem einzigen Wert zusammengefasst. Der Inhalt von CDATA-Abschnitten wird als Bestandteil der 'normalen Werte' geliefert.
- Es werden auch Werte zu Attributen geliefert, die im XML-Dokument nicht angegeben sind, sofern diese in der DTD mit einem Vorgabewert versehen sind.

13. Die Fortsetzung des Ablaufs der READ-Anweisung hängt davon ab, ob AT END oder NOT AT END angegeben ist, siehe Abschnitt "Ende-Bedingung".

11.4.6 START-Anweisung

Funktion

Die START-Anweisung positioniert auf Elemente bzw. Attribute in dem XML-Dokument und ordnet sie Datenfeldern in der FD zu.

Format

```
START dateiname-1 {ATTRIBUTE | ELEMENT} datenname-1 [INDEX IS {bezeichner-1 |
ganzzahl-1}]
  [INVALID KEY unbedingte-anweisung-1]
  [NOT INVALID KEY unbedingte-anweisung-2]
  [END-START]
```

Syntaxregeln

1. dateiname-1 muss der Name einer XML-organisierten Datei sein.
2. datenname-1 darf gekennzeichnet sein.
3. datenname-1 muss mit einer IDENTIFIED-Klausel beschrieben und der Datei dateiname-1 untergeordnet sein.
4. bezeichner-1 muss als ganzzahliges numerisches Datenelement definiert sein.
5. Wenn in der START-Anweisung ATTRIBUTE angegeben ist, muss die IDENTIFIED-Klausel in der Datenerklärung von datenname-1 die ATTRIBUTE-Angabe haben.
6. Wenn in der START-Anweisung ELEMENT angegeben ist, muss die IDENTIFIED-Klausel in der Datenerklärung von datenname-1 explizit oder implizit die ELEMENT-Angabe haben.
7. Die INDEX-Angabe ist nicht zulässig, wenn die IDENTIFIED-Klausel in der Datenerklärung von datenname-1 die BY- und die ATTRIBUTE-Angabe hat.

Allgemeine Regeln

1. Wenn die Datei dateiname-1 nicht geöffnet ist, ist die START-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 47 gesetzt.
2. Wenn in der Datei dateiname-1 kein XML-Dokument geöffnet ist, ist die START-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4D gesetzt.
3. Wenn das datenname-1 direkt übergeordnete Datenfeld keine gültige Position hat, oder datenname-1 die Stufennummer 1 und keine gültige Position hat, ist die START-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 25 gesetzt, d.h. eine Schlüsselfehler-Bedingung existiert.
4. Die START-Anweisung führt eine Zuordnung aus (siehe Abschnitt „Sprachelemente DATA DIVISION"), überträgt jedoch keine Daten. Wenn im ersten Schritt dieses Zuordnungsvorgangs kein Knoten aus dem XML-Baum zugeordnet werden kann, ist die START-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand wird auf 23 gesetzt, d.h. eine Schlüsselfehler-Bedingung existiert.
5. Wenn die Namen in den bei der Zuordnung wirkenden IDENTIFIED-Klauseln nicht eindeutig sind (siehe "IDENTIFIED-Klausel", allgemeine Regel 9 auf "IDENTIFIED-Klausel"), ist die START-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4C gesetzt.
6. Wenn sich einzelne Attribut- bzw. Elementnamen aus den bei der Zuordnung wirkenden IDENTIFIED-Klauseln nicht im Zeichensatz UTF-16 darstellen lassen, ist die START-Anweisung nicht erfolgreich. Der Ein-/Ausgabe-Zustand der Datei wird auf 4E gesetzt.
7. Bei der Zuordnung ist im ersten Schritt auf der COBOL-Seite nur das Datenfeld datenname-1 zu betrachten. Dazu dann im XML-Baum alle Kinder – in der Reihenfolge vom ältesten zum jüngsten Kind – des Knotens, der dem datenname-1 direkt übergeordneten Datenfeld zugeordnet ist. Wenn datenname-1 die Stufennummer 1 hat, ist im XML-Baum nur der datenname-1 zugeordnete Knoten zu betrachten.

8. Wenn INDEX angegeben ist, zeigt ganzzahl-1 bzw. der Inhalt von bezeichner-1 an, der wievielte Knoten aus der Menge aller Knoten aus dem XML-Baum, die im ersten Schritt der Zuordnung prinzipiell zugeordnet werden können, durch die Anweisung dem datenname-1 tatsächlich zugeordnet werden soll. Wenn der angegebene Wert kleiner als 1 oder größer als 2^{31} - 1 ist, ist das Verhalten undefiniert.
9. Die Fortsetzung des Ablaufs der START-Anweisung hängt davon ab, ob INVALID KEY oder NOT INVALID KEY angegeben ist (siehe Abschnitt „Schlüsselfehler-Bedingung").

11.4.7 XML Generate-Anweisung

Funktion

Die XML GENERATE-Anweisung konvertiert Daten in ein XML-Dokument.

Format

```
XML GENERATE datenname-1
        FROM datenname-1
      [COUNT IN datenname-3]
  [ON EXCEPTION unbedingte-anweisung-1]
  [NOT ON EXCEPTION unbedingte-anweisung-2]
  [END-XML]
```

Syntaxregeln

1. datenname-1 ist für das generierte XML-Dokument als Ausgabeoperand festgelegt.
2. Die Gruppe oder das Basisdatenelement datenname-2 wird in das XML-Dokument konvertiert.
3. datenname-*3* enthält die Anzahl der generierten XML-Zeichen.
4. Die Operanden datenname-1, datenname-2 und datenname-3 dürfen sich nicht überlappen.

Allgemeine Regeln

1. Das Datenfeld datenname-1 muss groß genug sein, damit er das generierte XML-Dokument enthalten kann und kann eines der folgenden Elemente referenzieren:
   - ein Datenelement oder eine Gruppe der Kategorie alphanumerisch;
   - ein Datenelement oder eine Gruppe der Kategorie national.

   Wenn der Inhalt von bezeichner-1 während der Ausführung der XML PARSE-Anweisung geändert wird, ist das weitere Verhalten undefiniert.
2. datenname-2 darf nicht mit der RENAMES-Klausel beschrieben sein.
   Die folgenden durch *datenname-2* spezifizierten Datenfelder werden von der XML GENERATE-Anweisung ignoriert:
   - Jedes unbenannte oder FILLER-Datenfeld oder einem solchen untergeordnete Datenfeld.
   - Jedes mit USAGE POINTER, PROGRAM-POINTER oder OBJECT REFERENCE definierte Datenfeld.
   - Jedes datenname-2 untergeordnete Datenfeld, dessen Datenerklärung die REDEFINES-Klausel enthält oder einem solchen Datenfeld untergeordnet ist.
   - Jedes datenname-2 untergeordnete Datenfeld, dessen Datenerklärung die RENAMES-Klausel enthält.
   - Jedes datenname-2 untergeordnete Datenfeld, das innerhalb der direkt übergeordneten Gruppe einen nicht eindeutigen Namen auf gleichem Level hat oder einem solchen Datetenfeld untergeordnet ist.
   - Jede datenname-2 untergeordnete Gruppe, die nicht mindestens ein elementares Datenfeld der Kategorien alphabetisch, alphanumerisch, numerisch, national oder Index enthält.
3. datenname-3 muss ein ganzzahliges Datenelement sein und enthält nach Ausführung der XML GENERATE-Anweisung die Anzahl der Zeichen des generierten XML-Dokuments.
4. Nach der Ausführung der XML GENERATE-Anweisung enthält das Sonderregister XML-CODE entweder null, was eine erfolgreich Durchführung bedeutet oder einen Ausnahmecode der ungleich null ist..

XML-CODE	Beschreibung
0	Das Empfangsfeld enthält das erfolgreich generierte XML-Dokument. Das COUNT [IN] Datenfeld enthält die Anzahl der Zeichen im generierten XML-Dokument.
400	sofortige Beendigung der Anweisung mit 'Fehler'
2999	sofortige Beendigung der Anweisung mit 'Fehler

5. Wenn während der Generierung des Dokuments ein Fehler auftritt (XML-CODE <> 0), wird unbedingte-anweisung-1, falls angegeben, ausgeführt und zum Ende der XML GENERATE-Anweisung verzweigt.
6. Wenn die Generierung des Dokuments ohne Fehler beendet wird (XML-CODE = 0), wird unbedingte-anweisung-2, falls angegeben, ausgeführt und zum Ende der XML GENERATE-Anweisung verzweigt.
7. Beim Erstellen des XML-Dokuments befolgt XML GENERATE folgende Regeln:
   - Der Inhalt jedes nicht ignorierten elementaren Datenfeldes von datenname-2 wird in ein abdruckbares Format konvertiert. Nachgestellte Leerzeichen und führende Nullen werden eliminiert.
   - Alle übrigen Instanzen der fünf Sonderzeichen & (kaufmännisches Und-Zeichen), ’ (Apostroph), > (größer als), < (kleiner als), und “ (Anführungsstriche) werden in die äquivalente XML-Referenz ’&’, ’ '’, ’>’, ’<’ bzw. ’"’ konvertiert.
   - Der transformierte Inhalt wird als Zeichenelementinhalt in das XML-Markup eingefügt.
   - Die XML-Elementnamen werden von der Definition von datenname-2 und diesem untergeordneten Datennamen abgeleitet. Die Namen von Gruppenfeldern werden in die übergeordneten XML-Namen übernommen.
   - Wenn datenname-1 national ist, werden alle nicht nationalen Daten in das nationale (UTF16) Format konvertiert und umgekehrt, wenn datenname-1 nicht national ist werden alle nationalen Daten in das alphanumerische Format (EBCDIC) konvertiert.

11.4.8 XML Parse-Anweisung

Funktion

Die XML PARSE-Anweisung zerlegt ein XML-Dokument sequenziell in seine syntaktischen Einheiten. Daten zu erkannten Einheiten stehen in einer Verarbeitungsprozedur in Sonderregistern zur weiteren Bearbeitung zur Verfügung.

Format

```
XML PARSE bezeichner-1

  PROCESSING PROCEDURE IS prozedurname-1 [{THRU | THROUGH} prozedurname-2]

  [ON EXCEPTION unbedingte-anweisung-1]

  [NOT ON EXCEPTION unbedingte-anweisung-2]

  [END-XML]
```

Syntaxregeln

1. bezeichner-1 muss ein alphanumerisches oder nationales Datenfeld sein.
2. bezeichner-1 darf kein Funktionsbezeichner sein.
3. Wenn sowohl prozedurname-1, als auch prozedurname-2 angegeben sind, und einer davon der Name einer Prozedur innerhalb von Prozedurvereinbarungskapiteln ist, muss der andere Prozedurname innerhalb desselben Prozedurvereinbarungskapitels liegen.

Allgemeine Regeln

1. bezeichner-1 enthält während der Ausführung der XML PARSE-Anweisung das zu zerlegende Dokument.
2. Wenn der Inhalt von bezeichner-1 während der Ausführung der XML PARSE-Anweisung geändert wird, ist das weitere Verhalten undefiniert.
3. Bei der Ausführung der XML PARSE-Anweisung wird die Steuerung an den Parser übergeben. Dieser zerlegt das Dokument und stellt die syntaktischen Einheiten dem Programm als 'Ereignisse' in Sonderregistern zur Verfügung.
4. Die PROCESSING PROCEDURE-Angabe legt die Verarbeitungsprozedur fest, an die der Parser bei jedem 'Ereignis' während der Ausführung der XML-Anweisung die Steuerung übergibt.

   Die Verarbeitungsprozedur wird so ausgeführt, als würde sie vom Parser über PERFORM aktiviert.
5. Die Verarbeitungsprozedur darf nicht mit EXIT PROGRAM, EXIT METHOD oder GOBACK verlassen werden. Wenn die letzte Anweisung der Verarbeitungsprozedur durchlaufen ist, geht die Steuerung automatisch an den Parser zurück.
6. Wenn der Parser bei der Zerlegung des XML-Dokuments einen Fehler erkennt, übergibt er die Steuerung an die Verarbeitungsprozedur mit dem Ereignis 'EXCEPTION' im Sonderregister XML-EVENT. Das Sonderregister XML-CODE enthält dann den Wert des erweiterten Ein-/Ausgabe-Zustandes, siehe Handbuch "COBOL2000 Benutzerhandbuch“ [1], Abschnitt 10.6 "Erweiterter Ein-/Ausgabe-Zustand für XML-Anweisungen (CBX-Code)“. Bei allen anderen Ereignissen hat das Sonderregister XML-CODE den Wert 0.
7. Der Inhalt des Sonderregisters XML-CODE beim **Verlassen** der Verarbeitungsprozedur steuert die Fortsetzung der XML PARSE-Anweisung. Die Verarbeitungsprozedur kann durch Setzen entsprechender Werte die gewünschte Fortsetzung signalisieren:

Wert von Sonderregister XML CODE bei Ende der Verarbeitungsprozedur	Wert von Sonderregister XML-EVENT bei Beginn der Verarbeitungsprozedur

	EXCEPTION	anderes Ereignis
0	Versuch, die Anweisung fortzusetzen[1]	Fortsetzung der Anweisung
-1	sofortige Beendigung der Anweisung mit 'Fehler'	sofortige Beendigung der Anweisung mit 'Fehler'
unveränderter Fehlercode	sofortige Beendigung der Anweisung mit 'Fehler	–
anderer Wert	weiteres Verhalten undefiniert	weiteres Verhalten undefiniert

[1]Die Fortsetzung ist jedoch nur bei 'leichten' Fehlern möglich und beschränkt sich dann zumeist auf die Su-che nach weiteren Fehlern. Bei 'schweren' Fehlern wird die XML-Anweisung trotzdem immer mit 'Fehler' beendet.

> **i** Werte ungleich -1 werden künftig mit eigener Semantik belegt.
> Setzen Sie daher keine Werte ungleich -1.

8. Nach Rückkehr vom Parser enthält das Sonderregister XML-CODE den Wert, den es beim letztmaligen Verlassen der Verarbeitungsprozedur hatte.
9. Wenn die Zerlegung des Dokuments ohne Fehler beendet wird, wird nach Rückkehr vom Parser unbedingte-anweisung-2, falls angegeben, ausgeführt und zum Ende der XML PARSE-Anweisung verzweigt.
10. Wenn die XML PARSE-Anweisung mit 'Fehler' beendet wurde, wird nach Rückkehr vom Parser unbedingte-anweisung-1, falls angegeben, ausgeführt und zum Ende der XML PARSE-Anweisung verzweigt.

11.5 Sonderregister für XML PARSE-Anweisung

Die ereignisorientierte Verarbeitung eines XML-Dokuments stellt Daten zu den erkannten, syntaktischen Einheiten zu Beginn der Verarbeitungsprozedur in Sonderregistern zur Verfügung.

Datenerklärungen für die Sonderregister dürfen vom Programmierer nicht geschrieben werden. Vielmehr erzeugt sie der Compiler automatisch, wenn bei der Übersetzung des Programms die Option zur Erkennung der XML-spezifischen Schlüsselwörter eingeschaltet ist (siehe Handbuch "COBOL2000 Benutzerhandbuch“ [1], Abschnitt 10.2 „Verwenden von XML-Sprachmitteln in Programmen“). Die Sonderregister existieren nur einmal pro Ablaufeinheit und stehen auch in geschachtelten Programmen zur Verfügung.

Es gibt drei Gruppen von Sonderregistern:

- XML-EVENT, XML-CODE:

  Die darin gelieferten Daten haben immer eine feste Länge und beschreiben ein Ereignis.
- Für XML-Dokumente, die in einem alphanumerischen Datenfeld bereitgestellt sind:

  XML-TEXT, XML-NAMESPACE, XML-NAMESPACE-PREFIX.

  Die darin gelieferten Daten haben variable Länge und beschreiben die mit einem Ereignis zusammenhängenden Daten.

  Zur Ausnahme beim Sonderfall des Ereignisses CONTENT-NATIONAL-CHARACTER siehe auch Abschnitte „XML-TEXT" und „XML-NTEXT".
- Für XML-Dokumente, die in einem nationalen Datenfeld bereitgestellt sind:

  XML-NTEXT, XML-NNAMESPACE, XML-NNAMESPACE-PREFIX.

  Die darin gelieferten Daten haben variable Länge und beschreiben die mit einem Ereignis zusammenhängenden Daten.

Die Sonderregister – ausgenommen XML-CODE – werden nur durch die XML PARSE-Anweisung modifiziert. Wenn sie als Empfangsfeld im Programm auftreten, führt das zu undefiniertem Verhalten. Daher haben die Sonderregister – ausgenommen XML-CODE – auch nur während der Ausführung einer XML-Anweisung einen sinnvollen Inhalt. Die Standardfunktion FUNCTION LENGTH (sonderregister) liefert bei Bedarf die aktuelle Länge der variabel langen Daten.

XML-EVENT

Dieses Sonderregister enthält zu Beginn der Verarbeitungsprozedur den Namen des gerade erkannten Ereignisses. Die möglichen Ereignisse sind in Abschnitt „Verarbeitungsprozedur" detailliert beschrieben.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

```
USAGE DISPLAY PIC X(30).
```

XML-CODE

Dieses Sonderregister enthält zu Beginn der Verarbeitungsprozedur einen Fehlerschlüssel, wenn es sich bei dem Ereignis um 'EXCEPTION' handelt. Bei allen anderen Ereignissen enthält es den Wert 0. Die möglichen Werte für Fehlerschlüssel entnehmen Sie dem Handbuch "COBOL2000 Benutzerhandbuch“ [1], Abschnitt 10.6 "Erweiterter Ein-/Ausgabe-Zustand für XML-Anweisungen (CBX-Code)“.

Als einziges Sonderregister darf es vom Programm auch modifiziert werden, um bei der Rückkehr aus einer Verarbeitungsprozedur der XML PARSE-Anweisung die gewünschte Fortsetzung zu signalisieren (siehe Abschnitt „XML-Anweisung").

Das Sonderregister entspricht einem Datenfeld mit der Beschreibung

```
USAGE COMP-5 PICTURE S9(9).
```

XML-TEXT

Wenn das XML-Dokument in einem alphanumerischen Datenfeld steht, enthält dieses Sonderregister zu Beginn der Verarbeitungsprozedur einen Teil des XML-Dokuments, der durch das Ereignis genau beschrieben wird.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

```
USAGE DISPLAY PICTURE X ANY LENGTH,
```

wobei die aktuelle Länge der Länge des gelieferten Dokument-Teils (in Zeichen) entspricht. Wenn das XML-Dokument in einem nationalen Datenfeld steht, oder wenn das Ereignis END-OF-DOCUMENT, CONTENT-NATIONAL-CHARACTER oder
NAMESPACE-DECLARATION ist, ist die aktuelle Länge 0.

XML-NTEXT

Wenn das XML-Dokument in einem nationalen Datenfeld steht, enthält dieses Sonderregister zu Beginn der Verarbeitungsprozedur einen Teil des XML-Dokuments, der durch das Ereignis genau beschrieben wird.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

```
USAGE NATIONAL PICTURE N ANY LENGTH,
```

wobei die aktuelle Länge der Länge des gelieferten Dokument-Teils (in Zeichen) entspricht. Wenn das XML-Dokument in einem alphanumerischen Datenfeld steht – außer beim Ereignis CONTENT-NATIONAL-CHARACTER –, oder wenn das Ereignis
END-OF-DOCUMENT oder NAMESPACE-DECLARATION ist, ist die aktuelle Länge 0.

XML-NAMESPACE

Wenn das XML-Dokument in einem alphanumerischen Datenfeld steht, enthält dieses Sonderregister zu Beginn der Verarbeitungsprozedur bei den Ereignissen ATTRIBUTE-NAME, DEFAULTED-ATTRIBUTE-NAME, END-OF-ELEMENT, NAMESPACE-DECLARATION und START-OF-ELEMENT den Namen des Namensraums.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

```
USAGE DISPLAY PICTURE X ANY LENGTH,
```

wobei die aktuelle Länge die Länge des Namensraum-Namens (in Zeichen) ist. Wenn das XML-Dokument in einem nationalen Datenfeld steht oder der Namensraum leer ist oder es sich um ein anderes Ereignis handelt, ist die aktuelle Länge 0.

XML-NNAMESPACE

Wenn das XML-Dokument in einem nationalen Datenfeld steht, enthält dieses Sonderregister zu Beginn der Verarbeitungsprozedur bei den Ereignissen ATTRIBUTE-NAME, DEFAULTED-ATTRIBUTE-NAME, END-OF-ELEMENT, NAMESPACE-DECLARATION und START-OF-ELEMENT den Namen des Namensraums.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

```
USAGE NATIONAL PICTURE N ANY LENGTH,
```

wobei die aktuelle Länge die Länge des Namensraum-Namens (in Zeichen) ist. Wenn das XML-Dokument in einem alphanumerischen Datenfeld steht oder der Namensraum leer ist oder es sich um ein anderes Ereignis handelt, ist die aktuelle Länge 0.

XML-NAMESPACE-PREFIX

Wenn das XML-Dokument in einem alphanumerischen Datenfeld steht, enthält dieses Sonderregister zu Beginn der Verarbeitungsprozedur bei den Ereignissen ATTRIBUTE-NAME, DEFAULTED-ATTRIBUTE-NAME, END-OF-ELEMENT, NAMESPACE-DECLARATION und START-OF-ELEMENT das Präfix des Namensraums.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

USAGE DISPLAY PICTURE X ANY LENGTH,

wobei die aktuelle Länge die Länge des Präfix (in Zeichen) ist. Wenn das XML-Dokument in einem nationalen Datenfeld steht oder der Namensraum kein Präfix hat oder es sich um ein anderes Ereignis handelt, ist die aktuelle Länge 0.

XML-NNAMESPACE-PREFIX

Wenn das XML-Dokument in einem nationalen Datenfeld steht, enthält dieses Sonderregister zu Beginn der Verarbeitungsprozedur bei den Ereignissen ATTRIBUTE-NAME, DEFAULTED-ATTRIBUTE-NAME, END-OF-ELEMENT, NAMESPACE-DECLARATION und START-OF-ELEMENT das Präfix des Namensraums.

Das Sonderregister entspricht einem Datenelement mit der Beschreibung

USAGE DISPLAY PICTURE N ANY LENGTH,

wobei die aktuelle Länge die Länge des Präfix (in Zeichen) ist. Wenn das XML-Dokument in einem alphanumerischen Datenfeld steht oder der Namensraum kein Präfix hat oder es sich um ein anderes Ereignis handelt, ist die aktuelle Länge 0.

12 Allgemeine Konzepte

In diesem Kapitel werden folgende Themen behandelt:

12.1 Dateiverarbeitung

Eine Datei ist eine Sammlung von Datensätzen, die auf einen Datenträger übertragen bzw. von dort gelesen werden kann. Der Benutzer bestimmt die Dateiorganisation, die Art und die Reihenfolge der Verarbeitung der Datensätze.

Die Organisation einer Datei beschreibt ihre logische Struktur. Es gibt sequenzielle, indizierte und relative Dateiorganisation. Die zum Zeitpunkt der Dateierstellung festgelegte Dateiorganisation kann später nicht mehr verändert werden.

Eine sequenzielle Datei kann nur sequenziell verarbeitet werden, d.h. die Datensätze werden in der durch die Datei vorgegebenen Reihenfolge gelesen oder geschrieben. Bei sequenziellen Dateien auf Plattenspeicher können Datensätze auch aktualisiert werden. Voraussetzung ist ein Lesen (READ), dem unmittelbar ein Rückschreiben (REWRITE) folgt.

12.1.1 Sequenzielle Dateiorganisation

In diesem Kapitel werden folgende Themen behandelt:

- Ein-/Ausgabe-Zustand
- Satzsequenzielle Organisation
- Zeilensequenzielle Organisation

12.1.1.1 Ein-/Ausgabe-Zustand

Der Ein-/Ausgabe-Zustand ist ein Wert, mit dem in einem COBOL-Programm der Zustand einer Ein-/Ausgabe-Operation abgefragt werden kann. Dazu muss die FILE STATUS-Klausel im FILE-CONTROL-Paragrafen der ENVIRONMENT DIVISION angegeben werden. Der Wert wird dann in ein zwei Zeichen langes Datenfeld übertragen, und zwar

- während der Ausführung einer CLOSE-, OPEN-, READ-, REWRITE- oder WRITE-Anweisung,
- vor Ausführung einer jeden damit zusammenhängenden unbedingten Anweisung,
- vor jeder entsprechenden USE AFTER STANDARD EXCEPTION-Prozedur.

Nachfolgend sind die Werte des Ein-/Ausgabe-Zustands und deren Bedeutung aufgeführt:

Ein-/Ausgabe Zustand	Bedeutung
	Erfolgreiche Ausführung
00	Die Ein-/Ausgabe-Anweisung wurde erfolgreich ausgeführt. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
04	Satzlängenkonflikt: Eine READ-Anweisung wurde erfolgreich ausgeführt. Die Länge des gelesenen Datensatzes liegt jedoch nicht in den Grenzen, die durch die Satzbeschreibungen der Datei festgelegt wurden.
05	Erfolgreicher OPEN INPUT/I-O/EXTEND auf eine Datei mit OPTIONAL-Angabe, die zum Zeitpunkt der Ausführung der OPEN-Anweisung nicht vorhanden war
07	1. Erfolgreiche OPEN-Anweisung mit NO REWIND-Klausel auf eine Datei auf UNIT-RECORD-Datenträger 2. Erfolgreiche CLOSE-Anweisung mit NO REWIND-, REEL/UNIT- oder FOR REMOVAL-Klausel auf eine Datei auf UNIT-RECORD-Datenträger
	Erfolglose Ausführung: Endebedingung
10	Es wurde versucht, eine READ-Anweisung auszuführen. Es war jedoch kein nächster logischer Datensatz vorhanden, da das Dateiende erreicht war. Es wurde zum ersten Mal versucht, eine READ-Anweisung für eine nicht vorhandene Datei mit OPTIONAL-Angabe auszuführen.
	Erfolglose Ausführung: Permanenter Fehler
30	1. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar (der erweiterte Ein-/Ausgabezustand liefert weitere Informationen). 2. Bei zeilensequenzieller Verarbeitung: erfolgloser Zugriff auf PLAM-Element
34	Es wurde versucht, außerhalb der vom System festgelegten Bereichsgrenzen einer sequenziellen Datei zu schreiben.
35	Es wurde versucht, eine OPEN-Anweisung mit INPUT-/ I-O-/ EXTEND-Angabe für eine nicht vorhandene Datei auszuführen.

37	OPEN-Anweisung auf eine Datei, die auf folgende Weise nicht eröffnet werden kann: 1. OPEN OUTPUT / I-O / EXTEND auf eine schreibgeschützte Datei (Passwort, RETENTION-PERIOD, ACCESS=READ im Katalog) 2. OPEN I-O auf eine Banddatei 3. OPEN INPUT auf eine lesegeschützte Datei (Passwort)
38	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die vorher mit der LOCK-Angabe geschlossen wurde.
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im ADD-FILE-LINK-Kommando wurden einer oder mehrere der Operanden ACCESS-METHOD, RECORD-FORMAT bzw. RECORD-SIZE mit Werten angegeben, die von den entsprechenden expliziten oder impliziten Programmangaben abweichen. 2. Bei Eingabedateien traten Satzlängenfehler auf (Katalogüberprüfung, falls RECFORM=F). 3. Die Satzlänge ist größer als die BLKSIZE im Katalog bei Eingabedateien 4. Für eine Eingabedatei stimmt der Katalogeintrag eines der Operanden FCBTYPE, RECFORM oder RECSIZE (falls RECFORM=F) nicht mit den entsprechenden expliziten oder impliziten Programmangaben bzw. mit denentsprechenden Angaben im ADD-FILE-LINK-Kommando überein.
	Erfolglose Ausführung: Logischer Fehler
41	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die bereits eröffnet ist.
42	Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht eröffnet ist.
43	Bei Zugriff auf eine Plattenspeicherdatei, die mit OPEN I-O eröffnet wurde: Die letzte vor Ausführung einer REWRITE-Anweisung ausgeführte Ein-/Ausgabe- Anweisung war keine erfolgreich ausgeführte READ-Anweisung.
44	Überschreiten der Bereichsgrenzen: 1. Es wurde versucht, eine WRITE-Anweisung auszuführen. Die Länge des Datensatzes liegt jedoch nicht in dem für diese Datei zulässigen Bereich. 2. Es wurde versucht, eine REWRITE-Anweisung auszuführen. Der zurückzuschreibende Datensatz hat jedoch nicht die gleiche Länge wie der zu ersetzende Datensatz.
46	Es wurde versucht, eine READ-Anweisung für eine Datei auszuführen, die sich im Eröffnungsmodus INPUT oder I-O befindet, ein nächster gültiger Datensatz steht aber nicht zur Verfügung. Grund: 1. Die vorhergehende READ-Anweisung war erfolglos, ohne eine Ende-Bedingung zu verursachen, oder 2. Die vorhergehende READ-Anweisung hat eine Ende-Bedingung verursacht.
47	Es wurde versucht, eine READ-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus INPUT oder I-O befindet.
48	Es wurde versucht, eine WRITE-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus OUTPUT oder EXTEND befindet.
49	Es wurde versucht, eine REWRITE-Anweisung für eine Datei auszuführen, die sich nicht im Modus I-O befindet.

	Sonstige erfolglose Ausführungen
90	Systemfehler; es ist keine weitere Information über die Ursache vorhanden.
91	Systemfehler; ein Systemaufruf war nicht erfolgreich; entweder OPEN-Fehler oder kein freies Gerät; die eigentliche Ursache ist aus dem DVS-Code ersichtlich (siehe „FILE STATUS-Klausel").
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des ADD-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers

12.1.1.2 Satzsequenzielle Organisation

Bei Verwendung sequenzieller Organisation für eine Datei werden die logischen Datensätze in der Reihenfolge der Erzeugung sequenziell angeordnet und in der Erzeugungsreihenfolge sequenziell vorwärts oder rückwärts (REVERSED) gelesen.

Diese Art der Dateiorganisation muss für Magnetband- oder Einheitsdatensatzdateien verwendet werden und kann für Plattenspeicherdateien verwendet werden. Sequenziell organisierte Dateien benötigen zur Satzverarbeitung keine Schlüssel.

12.1.1.3 Zeilensequenzielle Organisation

Die zeilensequenzielle Organisation von COBOL-Dateien ist ein Sprachmittel des X/Open-Standards. Sie dient dazu, Textdateien zu erzeugen bzw. zu verarbeiten, die von den Texteditoren des Betriebssystems verarbeitet werden können bzw. erzeugt wurden.

12.1.2 Relative Dateiorganisation

In diesem Kapitel werden folgende Themen behandelt:

- Relative Organisation
- Sequenzieller Zugriff auf Datensätze
- Wahlfreier Zugriff auf Datensätze
- Dynamischer Zugriff auf Datensätze
- Ein-/Ausgabe-Zustand

12.1.2.1 Relative Organisation

Bei Verwendung von relativer Organisation ist die Lage eines jeden Datensatzes in einer relativen Datei anhand einer relativen Satznummer festgelegt, d.h. einem ganzzahligen Wert größer Null, der die Lage des Datensatzes innerhalb der logischen Reihenfolge der Datei angibt. Die Satznummer wird vom Benutzer in einem relativen Schlüsselfeld vorgegeben. Man kann sich die Datei als eine serielle Folge von Bereichen vorstellen, von denen jeder einen logischen Datensatz enthält. Jeder dieser Bereiche wird von einer relativen Satznummer bezeichnet. Anhand dieser Nummer werden die Datensätze abgespeichert und aufgefunden. Zum Beispiel wird der zehnte Datensatz über die relative Satznummer 10 adressiert und liegt im zehnten Satzbereich unabhängig davon, ob Datensätze in den ersten bis neunten Satzbereich geschrieben wurden. Relative Dateiorganisation ist nur für Plattenspeicherdateien zulässig.

12.1.2.2 Sequenzieller Zugriff auf Datensätze

Datensätze einer relativen Datei können sequenziell erzeugt, gelesen und aktualisiert werden.

Eine relative Datei kann sequenziell erstellt werden; in diesem Fall muss der RELATIVE KEY nicht angegeben werden, da die Datensätze physisch fortlaufend in die Ausgabedatei geschrieben werden.

Beim sequenziellen Lesen von Datensätzen aus einer relativen Datei werden die Datensätze in der Reihenfolge ihrer relativen Satznummern verarbeitet, d.h. eine READ-Anweisung stellt den nächsten oder vorgehenden existierenden logischen Datensatz der Datei zur Verfügung. Der erste gelesene Datensatz ist entweder der erste in der Datei zur Verfügung stehende Datensatz oder ein Datensatz, auf den mit einer START-Anweisung positioniert wurde; in diesem Fall muss RELATIVE KEY angegeben sein.

Ist RELATIVE KEY für eine relative Datei angegeben, so wird bei Ausführung einer READ- oder WRITE-Anweisung das RELATIVE KEY-Datenfeld auf die relative Satznummer des zur Verfügung gestellten Datensatzes gesetzt.

Eine bereits existierende relative Datei kann mit Hilfe von READ-, REWRITE- oder DELETE-Anweisungen aktualisiert werden. Der zuletzt gelesene Datensatz kann aktualisiert und dann an seinen ursprünglichen Platz in der Datei zurückgeschrieben oder logisch gelöscht werden.

12.1.2.3 Wahlfreier Zugriff auf Datensätze

Datensätze einer relativen Datei können wahlfrei erzeugt, gelesen und aktualisiert werden. Bei dieser Zugriffsmethode ist die RELATIVE KEY-Angabe immer erforderlich. Vor Ausführung jeder Ein- oder Ausgabe-Anweisung muss das in der RELATIVE KEY-Angabe genannte Datenfeld mit dem Wert der erforderlichen relativen Satznummer versorgt werden.

Eine relative Datei kann wahlfrei erstellt werden; in diesem Fall belegt der in die Datei ausgegebene Datensatz den Satzbereich mit der relativen Satznummer, die im RELATIVE KEY-Datenfeld angegeben ist.
Wird eine relative Datei wahlfrei gelesen, so wird derjenige Datensatz zur Verfügung gestellt, dessen relative Satznummer im Datenfeld der zu dieser Datei gehörigen RELATIVE KEY-Angabe steht.

Eine bereits existierende relative Datei kann mit Hilfe von REWRITE- oder DELETE-Anweisungen aktualisiert werden. Eine explizite wahlfreie READ-Anweisung kann wahlweise vor Ausführung einer REWRITE- oder DELETE-Anweisung für den zu aktualisierenden Datensatz gegeben werden, da diese Anweisungen denjenigen Datensatz ändern, der durch die relative Satznummer im RELATIVE KEY-Datenfeld angegeben ist.

12.1.2.4 Dynamischer Zugriff auf Datensätze

Mischung aus sequenzieller und wahlfreier Zugriffsart.

12.1.2.5 Ein-/Ausgabe-Zustand

Der Ein-/Ausgabe-Zustand ist ein Wert, mit dem in einem COBOL-Programm der Zustand einer Ein-/Ausgabe-Operation abgefragt werden kann. Dazu muss die FILE STATUS-Klausel im FILE-CONTROL-Paragrafen der ENVIRONMENT DIVISION angegeben werden. Der Wert wird dann in ein zwei Zeichen langes Datenfeld übertragen, und zwar

- während der Ausführung einer CLOSE-, DELETE-, OPEN-, READ-, REWRITE-, START- oder WRITE-Anweisung,
- vor Ausführung einer jeden damit zusammenhängenden unbedingten Anweisung,
- vor jeder entsprechenden USE AFTER STANDARD EXCEPTION-Prozedur.

Nachfolgend sind die Werte des Ein-/Ausgabe-Zustands und deren Bedeutung aufgeführt:

Ein-/Ausgabez-ustand	Bedeutung
	Erfolgreiche Ausführung
00	Die Ein-/Ausgabe-Anweisung wurde erfolgreich ausgeführt. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
04	Satzlängenkonflikt: Eine READ-Anweisung wurde erfolgreich ausgeführt. Die Länge des gelesenen Datensatzes liegt jedoch nicht in den Grenzen, die durch die Satzbeschreibungen der Datei festgelegt wurden. Erfolgreicher OPEN INPUT/I-O/EXTEND auf eine Datei mit OPTIONAL-Angabe in der SELECT-Klausel, die zum Zeitpunkt der Ausführung der OPEN-Anweisung nicht vorhanden war
	Erfolglose Ausführung: Endebedingung
10	Es wurde versucht, eine READ-Anweisung auszuführen. Es war jedoch kein nächster logischer Datensatz vorhanden, da das Dateiende erreicht war. Es wurde zum ersten Mal versucht, eine READ-Anweisung für eine nicht vorhandene Datei mit OPTIONAL-Angabe auszuführen.
14	Es wurde versucht, eine READ-Anweisung auszuführen. Das durch RELATIVE KEY beschriebene Datenfeld ist aber zu klein, um die relative Satznummer aufzunehmen (sequenzielles READ).
	Erfolglose Ausführung: Schlüsselfehlerbedingung
22	Doppelter Schlüssel: Es wurde versucht, eine WRITE-Anweisung mit einem Schlüssel auszuführen, für den in der Datei bereits ein Satz vorhanden ist.
23	Datensatz nicht gefunden oder Satzschlüssel Null: Es wurde versucht, anhand eines Schlüssels mit einer READ-, START-, DELETE- oder REWRITE-Anweisung auf einen Datensatz zuzugreifen, der in der Datei nicht vorhanden ist, oder der Zugriff erfolgte mit Satzschlüssel Null
24	Überschreiten der Bereichsgrenzen: Es wurde versucht, eine WRITE-Anweisung außerhalb der vom System festgelegten Bereichsgrenzen einer relativen Datei auszuführen (unzureichende Sekundärzuweisung im FILE-Kommando) oder eine WRITE-Anweisung im sequenziellen Zugriffsmodus zu geben,

	bei der die relative Satznummer so groß ist, dass sie nicht in das mit der RELATIVE KEY-Klausel beschriebene Datenfeld passt.
	Erfolglose Ausführung: Permanenter Fehler
30	Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar (der DVS-Code liefert weitere Informationen).
35	Es wurde versucht, eine OPEN-Anweisung mit INPUT-/I-O-/EXTEND-Angabe für eine nicht vorhandene Datei auszuführen.
37	OPEN-Anweisung auf eine Datei, die auf folgende Weise nicht eröffnet werden kann: 1. OPEN OUTPUT/I-O/EXTEND auf eine schreibgeschützte Datei (Passwort, RETENTION-PERIOD, ACCESS=READ im Katalog) 2. OPEN INPUT auf eine lesegeschützte Datei (Passwort)
38	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die vorher mit der LOCK-Angabe geschlossen wurde.
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im ADD-FILE-LINK-Kommando wurden einer oder mehrere der Operanden ACCESS-METHOD, RECORD-FORMAT bzw. RECORD-SIZE mit Werten angegeben, die von den entsprechenden expliziten oder impliziten Programmangaben abweichen. 2. Für eine Eingabedatei stimmt der Katalogeintrag des Operanden FCBTYPE nicht mit der entsprechenden expliziten oder impliziten Programmangabe bzw. mit der entsprechenden Angabe im ADD-FILE-LINK-Kommando überein. 3. Für eine Datei, die mit der DVS-Zugriffsmethode UPAM verarbeitet werden soll, wurde variable Satzlänge vereinbart.
	Erfolglose Ausführung: Logischer Fehler
41	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die bereits eröffnet ist.
42	Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht eröffnet ist.
43	Bei ACCESS MODE IS SEQUENTIAL: Die letzte vor Ausführung einer DELETE- oder REWRITE-Anweisung ausgeführte Ein-/Ausgabe-Anweisung war keine erfolgreich ausgeführte READ-Anweisung.
44	Überschreiten der Satzlängengrenzen: Es wurde versucht, eine WRITE- oder REWRITE-Anweisung auszuführen. Die Länge des Datensatzes liegt jedoch nicht in dem für diese Datei zulässigen Bereich.
46	Es wurde versucht, eine sequenzielle READ-Anweisung für eine Datei auszuführen, die sich im Eröffnungsmodus INPUT oder I-O befindet; ein nächster gültiger Datensatz steht aber nicht zur Verfügung. Grund: 1. Die vorhergehende START-Anweisung war erfolglos, oder 2. die vorhergehende READ-Anweisung war erfolglos, ohne eine Endebedingung zu verursachen, oder 3. die vorhergehende READ-Anweisung hat eine Ende-Bedingung verursacht.
47	Es wurde versucht, eine READ-Anweisung für eine Datei auszuführen, die sich nicht im Eröffnungsmodus INPUT oder I-O befindet.

48	Es wurde versucht, eine WRITE-Anweisung für eine Datei auszuführen, die sich • bei sequenziellem Zugriff nicht im Eröffnungsmodus OUTPUT oder EXTEND, • bei wahlfreiem oder dynamischem Zugriff nicht im Eröffnungsmodus OUTPUT oder I-O befindet.
49	Es wurde versucht, eine DELETE- oder REWRITE-Anweisung für eine Datei auszuführen, die sich nicht im Modus I-O befindet.
	Sonstige erfolglose Ausführungen
90	Systemfehler; es ist keine weitere Information über die Ursache vorhanden.
91	Systemfehler; OPEN-Fehler
93	Nur bei Simultanverarbeitung (siehe „Simultanverarbeitung“ im Handbuch „COBOL2000 Benutzerhandbuch“ [1]): Die Ein-/Ausgabe-Anweisung konnte nicht erfolgreich durchgeführt werden, weil ein anderer Prozess auf dieselbe Datei zugreift und die Zugriffe nicht vereinbar sind.
94	Nur bei Simultanverarbeitung (siehe „Simultanverarbeitung“ im Handbuch „COBOL2000 Benutzerhandbuch“ [1]): Die Aufruffolge READ - REWRITE/DELETE wurde nicht eingehalten.
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des ADD-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers
96	READ PREVIOUS wird nicht unterstützt für ein Modul, der mit COBRUN ENABLE-UFS-ACCESS=YES übersetzt wurde, oder die Datei soll mit der DVS-Zugriffsmethode UPAM bearbeitet werden.

12.1.3 Indizierte Dateiorganisation

In diesem Kapitel werden folgende Themen behandelt:

- Indizierte Organisation
- Sequenzieller Zugriff auf Datensätze
- Wahlfreier Zugriff auf Datensätze
- Dynamischer Zugriff auf Datensätze
- Ein-/Ausgabe-Zustand

12.1.3.1 Indizierte Organisation

Bei Verwendung indizierter Organisation für eine Datei wird die Position jedes Datensatzes in der Datei durch Indizes bestimmt, die mit der Datei erzeugt und vom System gewartet werden. Diese Indizes stützen sich auf Schlüssel, die vom Anwender in den Sätzen bereitzustellen sind. Indizierte Dateien müssen Plattenspeichergeräten zugeordnet sein.

Beim Erzeugen einer indizierten Datei muss die RECORD KEY-Klausel angegeben werden. Mit ihr wird definiert, welches Datenfeld innerhalb des Datensatzes als Primärschlüsselfeld verwendet werden soll.

Mit der ALTERNATE RECORD KEY-Klausel können ein oder mehrere zum Primärschlüssel alternative Satzschlüssel definiert werden (Sekundärschlüssel).

Die START-Anweisung kann verwendet werden, um einen Startpunkt innerhalb einer indizierten Datei für eine Reihe nachfolgender sequenzieller Zugriffsoperationen zu bestimmen.

12.1.3.2 Sequenzieller Zugriff auf Datensätze

Datensätze einer indizierten Datei können sequenziell gelesen, aktualisiert oder neu erstellt werden.

Die Datensätze werden in der Reihenfolge aufsteigender oder absteigender Schlüssel gelesen.

12.1.3.3 Wahlfreier Zugriff auf Datensätze

Datensätze von indizierten Dateien können wahlfrei erzeugt, gelesen und aktualisiert werden.

Das als Schlüssel definierte Datenfeld ist der in der RECORD KEY-Klausel angegebene Datenname.

Beim Neuerstellen eines Datensatzes darf der Wert des RECORD KEY mit keinem Wert eines bereits existierenden Schlüsselfeldes übereinstimmen.

12.1.3.4 Dynamischer Zugriff auf Datensätze

Im dynamischen Zugriffsmodus darf der Benutzer beliebig vom sequenziellen in den wahlfreien Zugriff wechseln, indem er entsprechende Ein-/Ausgabe-Anweisungen verwendet.

12.1.3.5 Ein-/Ausgabe-Zustand

Der Ein-/Ausgabe-Zustand ist ein Wert, mit dem in einem COBOL-Programm der Zustand einer Ein-/Ausgabe-Operation abgefragt werden kann. Dazu muss die FILE STATUS-Klausel im FILE-CONTROL-Paragrafen der ENVIRONMENT DIVISION angegeben werden. Der Wert wird dann in ein zwei Zeichen langes Datenfeld übertragen, und zwar

- während der Ausführung einer CLOSE-, OPEN-, READ-, REWRITE- oder WRITE-Anweisung,
- vor Ausführung einer jeden damit zusammenhängenden unbedingten Anweisung,
- vor jeder entsprechenden USE AFTER STANDARD EXCEPTION-Prozedur.

Nachfolgend sind die Werte des Ein-/Ausgabe-Zustands und deren Bedeutung aufgeführt:

Ein-/Ausgabe-zustand	Bedeutung
	Erfolgreiche Ausführung
00	Die Ein-/Ausgabe-Anweisung wurde erfolgreich ausgeführt. Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfügbar.
02	Ein Satz wurde über ALTERNATE KEY gelesen, und es existiert bei sequenzi ellem Weiterlesen über denselben Schlüssel noch mindestens ein Nachfolge satz mit identischem Schlüsselwert. Ein Satz mit ALTERNATE KEY WITH DUPLICATES wurde geschrieben, und es gibt bereits für mindestens einen Alternativschlüssel einen Satz mit identischem Schlüsselwert.
04	Satzlängenkonflikt: Eine READ-Anweisung wurde erfolgreich ausgeführt. Die Länge des gelesenen Datensatzes liegt jedoch nicht in den Grenzen, die durch die Satzbeschreibungen der Datei festgelegt wurden.
05	OPEN-Anweisung auf eine nicht vorhandene OPTIONAL-Datei
	Erfolglose Ausführung: Endebedingung
10	Es wurde versucht, eine READ-Anweisung auszuführen. Es war jedoch kein nächster logischer Datensatz vorhanden, da das Dateiende erreicht war.
	Erfolglose Ausführung: Schlüsselfehlerbedingung
21	Reihenfolgefehler für eine Datei bei ACCESS MODE IS SEQUENTIAL: 1. Der Wert des Satzschlüssels wurde zwischen der erfolgreichen Ausführung einer READ-Anweisung und der Ausführung der nachfolgenden REWRITE-Anweisung geändert. 2. Bei aufeinanderfolgenden WRITE-Anweisungen wurde die aufsteigende Folge von Satzschlüsseln nicht eingehalten.
22	Doppelter Schlüssel: Es wurde versucht, eine WRITE-Anweisung mit einem Schlüssel auszuführen, für den in der Datei bereits ein Satz vorhanden ist. Es wurde versucht, einen Satz mit ALTERNATE KEY ohne WITH DUPLICATES-Angabe zu erstellen, obwohl in der Datei bereits ein Alternativschlüssel mit identischem Schlüsselwert vorhanden ist.
23	

	Datensatz nicht gefunden: Es wurde versucht, anhand eines Schlüssels mit einer READ-, START-, DELETE- oder REWRITE-Anweisung auf einen Datensatz zuzugreifen, der in der Datei nicht vorhanden ist.
24	Überschreiten der Bereichsgrenzen: Es wurde versucht, eine WRITE-Anweisung außerhalb der vom System festge legten Bereichsgrenzen einer relativen Datei auszuführen (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).
	Erfolglose Ausführung: Permanenter Fehler
30	Es ist keine weitere Information bezüglich der Ein-/Ausgabe-Operation verfüg bar (der DVS-Code liefert weitere Informationen).
35	Es wurde versucht, eine OPEN INPUT, I-O- oder EXTEND-Anweisung für eine nicht optionale Datei auszuführen, die nicht vorhanden war.
37	OPEN-Anweisung auf eine Datei, die wegen folgender Bedingungen nicht eröffnet werden kann: 1. OPEN OUTPUT/I-O/EXTEND auf eine schreibgeschützte Datei (Passwort, RETENTION-PERIOD, ACCESS=READ im Katalog) 2. OPEN INPUT auf eine lesegeschützte Datei (Passwort)
38	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die vor her mit der LOCK-Angabe geschlossen wurde.
39	Die OPEN-Anweisung war aus einem der folgenden Gründe erfolglos: 1. Im ADD-FILE-LINK-Kommando wurden einer oder mehrere der Operanden ACCESS-METHOD, RECORD-FORMAT bzw. RECORD-SIZE mit Werten angegeben, die von den entsprechenden expliziten oder impliziten Programmangaben abweichen. 2. Bei einer Eingabedatei trat ein Satzlängenfehler auf (Katalogüberprüfung, falls RECFORM=F). 3. Die Satzlänge ist größer als die BLKSIZE-Angabe im Katalog einer Eingabedatei. 4. Für eine Eingabedatei stimmt der Katalogeintrag eines der Operanden FCBTYPE, RECFORM, RECSIZE (falls RECFORM=F), KEYPOS oder KEYLEN nicht mit den entsprechenden expliziten oder impliziten Programmangaben bzw. mit den entsprechenden Angaben im ADD-FILE-LINK-Kommando überein. 5. Es wurde versucht, eine Datei zu eröffnen, deren Alternativschlüssel nicht mit den im Programm angegebenen Schlüsselwerten der ALTERNATERECORDKEY-Klausel übereinstimmen.
	Erfolglose Ausführung: Logischer Fehler
41	Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die bereits eröffnet ist.
42	Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht eröffnet ist.
43	Bei ACCESS MODE IS SEQUENTIAL war die letzte vor Ausführung einer DELETE- oder REWRITE-Anweisung ausgeführte Ein-/Ausgabe-Anweisung keine erfolgreich ausgeführte READ-Anweisung.
44	Überschreiten der Satzlängengrenzen: Es wurde versucht, eine WRITE- oder REWRITE-Anweisung auszuführen. Die Länge des Datensatzes liegt jedoch nicht in dem für diese Datei zulässigen Bereich.

46	Es wurde versucht, eine sequenzielle READ-Anweisung für eine Datei auszuführen, die sich im Eröffnungsmodus INPUT oder I-O befindet; ein nächster gültiger Datensatz steht aber nicht zur Verfügung. Grund: 1. Die vorhergehende START-Anweisung war erfolglos, oder 2. die vorhergehende READ-Anweisung war erfolglos, ohne eine Endebedingung zu verursachen, oder 3. es wurde versucht, nach bereits erkannter AT END-Bedingung eine READ-Anweisung auszuführen.
47	Es wurde versucht, eine READ- oder START-Anweisung für eine Datei auszu führen, die sich nicht im Eröffnungsmodus INPUT oder I-O befindet.
48	Es wurde versucht, eine WRITE-Anweisung für eine Datei auszuführen, die sich 1. bei sequenziellem Zugriff nicht im Eröffnungsmodus OUTPUT oder EXTEND, 2. bei wahlfreiem oder dynamischem Zugriff nicht im Eröffnungsmodus OUTPUT oder I-O befindet.
49	Es wurde versucht, eine DELETE- oder REWRITE-Anweisung für eine Datei auszuführen, die sich nicht im Modus I-O befindet.
	Sonstige erfolglose Ausführungen
90	Systemfehler; es ist keine weitere Information über die Ursache vorhanden.
91	Systemfehler; OPEN-Fehler; die eigentliche Ursache ist aus dem DVS-Code ersichtlich (siehe „FILE-STATUS-Klausel“ mit Angabe von datenname-2).
93	Nur bei Simultanverarbeitung (siehe „Simultanverarbeitung“ im Handbuch „COBOL2000 Benutzerhandbuch“ [1]): Die Ein-/Ausgabe-Anweisung konnte nicht erfolgreich durchgeführt werden, weil ein anderer Prozess auf dieselbe Datei zugreift und die Zugriffe nicht vereinbar sind.
94	Nur bei Simultanverarbeitung (siehe „Simultanverarbeitung“ im Handbuch „COBOL2000 Benutzerhandbuch“ [1]): 1. Die Aufruffolge READ - REWRITE/DELETE wurde nicht eingehalten. 2. Die Satzlänge ist größer als die Blocklänge.
95	Unverträglichkeit zwischen den Angaben im BLOCK-CONTROL-INFO- oder BUFFER-LENGTH-Operanden des ADD-FILE-LINK-Kommandos und dem Dateiformat, der Blockgröße oder dem Format des verwendeten Datenträgers
96	READ PREVIOUS wird nicht unterstützt für ein Modul, der mit COBRUN ENABLE-UFS-ACCESS=YES übersetzt wurde.

12.1.4 Ein-/Ausgabe-Anweisungen

Die Ein- und Ausgabe in COBOL ist satzorientiert. Daher verarbeiten die READ-, WRITE-, DELETE- und REWRITE-Anweisungen Datensätze. Der COBOL-Anwender muss sich also nur mit der Verarbeitung einzelner Datensätze befassen. Die folgenden Operationen werden automatisch ausgeführt:

- Übertragung von Daten in Ein-/Ausgabe-Bereiche oder in den Internspeicher,
- Gültigkeitsüberprüfung,
- Fehlerkorrekturen (wo dies möglich ist),
- Blockung, Entblockung,
- Bandwechsel (nur für **sequenzielle Dateiorganisation**).

Bemerkung

In der Beschreibung von Ein-/Ausgabe-Anweisungen für **sequenziell organisierte Dateien** werden die Ausdrücke Band und Spule verwendet. Band kann auf alle Ein-/Ausgabe-Dateien angewendet werden. Spule bezieht sich nur auf Magnetbanddateien. Die Behandlung von Plattenspeicherdateien beim sequenziellen Zugriff ist logisch gleichbedeutend mit der Behandlung von Magnetbanddateien.

Übersicht

Anweisung	Funktion
CLOSE	Beenden der Verarbeitung einer Datei
DELETE	Löschen eines Datensatzes
OPEN	Eröffnen einer Datei für die Verarbeitung
READ	Lesen eines Datensatzes
REWRITE	Ersetzen eines Datensatzes
START	Positionieren innerhalb einer Datei
USE	Zusätzlich zu Ein-/Ausgabe-Anweisungen können USE-Anweisungen zur Angabe von Fehlerbehandlungs-Prozeduren angegeben werden (siehe „DECLARATIVES")
WRITE	Schreiben eines Datensatzes

DELETE und START gelten nur für **relative und indizierte Dateiorganisation**.

12.1.5 Schlüsselfehler-Bedingung

Die Schlüsselfehler-Bedingung kann nach der Ausführung einer DELETE-, READ-, REWRITE-, START- oder WRITE-Anweisung auftreten.

Der weitere Ablauf nach Ausführung der Ein-/Ausgabe-Anweisung hängt von dem angezeigten Ein-/Ausgabe-Zustand ab. Die folgende Tabelle zeigt, wie/wo der Programmlauf jeweils fortgesetzt wird. '-' steht für Angaben, die für die Auswahl der Fortsetzung nicht relelvant sind.

Ein-/Ausgabe-Zustand nach der Anweisung zeigt an	**INVALID KEY unbedingte-anweisung-1**		**NOT INVALID KEY unbedingte-anweisung-1**		**USE-Prozedur**	
	angegeben	**nicht angegeben**	**angegeben**	**nicht angegeben**	**vereinbart**	**nicht vereinbart**
erfolgreich	-	-	unbedingte-anweisung-2	Ende der Anweisung	-	-
Schlüsselfehler-Bedingung	unbedingte-anweisung-1	Fortsetzung wie in Spalte 'USE-Prozedur' für andere Fehler-Bedingung	-	-	-	-
andere Fehler-Bedingung	-	-	-	-	use-procedure	Programm-abbruch

unbedingte-anweisung-1, unbedingte-anweisung-2 bzw. use-procedure wird gemäß den Regeln für die dort angegebenen Anweisungen ausgeführt. Der Ablauf wird, je nach Art der Anweisung, entweder in einem anderen Teil des Programms oder am Ende der Ein-/Ausgabe-Anweisung fortgesetzt.

12.1.6 Ende-Bedingung

Die Ende-Bedingung kann nach der Ausführung einer OPEN DOCUMENT- oder READ-Anweisung auftreten.

Der weitere Ablauf nach Ausführung der Ein-/Ausgabe-Anweisung hängt von dem angezeigten Ein-/Ausgabe-Zustand ab. Die folgende Tabelle zeigt, wie/wo der Programmlauf jeweils fortgesetzt wird. '-' steht für Angaben, die für die Auswahl der Fortsetzung nicht relelvant sind.

Ein-/Ausgabe-Zustand nach der Anweisung zeigt an	**AT END unbedingte-anweisung-1**		**NOT AT END unbedingte-anweisung-2**		**USE-Prozedur**	
	angegeben	**nicht angegeben**	**angegeben**	**nicht angegeben**	**vereinbart**	**nicht vereinbart**
erfolgreich	-	-	unbedingte-anweisung-2	Ende der Anweisung	-	-
Ende-Bedingung	unbedingte-anweisung-1	Fortsetzung wie in Spalte 'USE-Prozedur' für andere Fehler-Bedingung	-	-	-	-
andere Fehler-Bedingung	-	-	-	-	use-procedure	Programm-abbruch

unbedingte-anweisung-1, unbedingte-anweisung-2 bzw. use-procedure wird gemäß den Regeln für die dort angegebenen Anweisungen ausgeführt. Der Ablauf wird, je nach Art der Anweisung, entweder in einem anderen Teil des Programms oder am Ende der Ein-/Ausgabe-Anweisung fortgesetzt.

12.2 Ausnahmesituationen und Ausnahmezustände

Eine Ausnahmesituation kann zur Laufzeit als Abweichung zur normalen Ausführung einer COBOL-Anweisung auftreten.
Um auf eine Ausnahmesituation in einem Programm bzw. einer Methode gezielt reagieren zu können, gibt es neben der Angabe von expliziten Klauseln (z. B. SIZE ERROR-Angabe) und der Abfrage von Ein- und Ausgabezuständen bei der Dateibearbeitung für einige Ausnahmesituationen die Möglichkeit, Ausnahmezustände auszulösen und diese in USE-Prozeduren zu behandeln. Diesen Ausnahmesituationen sind Ausnahmesituationsnamen zugeordnet, durch die sie angesprochen werden können. Eine Aufstellung dieser Namen befindet sich in Tabelle 45.
Um für eine aufgetretene Ausnahmesituation den zugehörigen Ausnahmezustand auszulösen, muss deren Überprüfung durch die >>TURN-Direktive eingeschaltet sein.

Ist für eine Ausnahmesituation die Überprüfung eingeschaltet, gleichzeitig aber eine Klausel (ohne NOT) angegeben, mit der die Ausnahmesituation behandelt werden kann (z.B. INVOKE ... ON EXCEPTION für EC-OO-NULL), so hat die Klausel Vorrang, d.h. die Ausnahmesituation wird zwar (aufgrund der Klausel) überprüft, der zugehörige Ausnahmezustand wird aber **nicht** ausgelöst.

Zusätzlich können Ausnahmezustände direkt durch die RAISE-Anweisung ausgelöst werden.

Wird ein Ausnahmezustand ausgelöst, so wird der letzte Ausnahmezustand aktualisiert. Dieser gilt für die gesamte Ablaufeinheit solange, bis ein neuer Ausnahmezustand ausgelöst wird oder die Anweisung SET LAST EXCEPTION TO OFF ausgeführt wird. Der zugehörige Name kann über die Funktion EXCEPTION-STATUS abgefragt werden.
Der weitere Ablauf hängt von der Schwere (Kategorie) der Ausnahmesituation ab. Ausnahmesituationen sind entweder FATAL oder NON-FATAL.

Auslösen einer FATAL Ausnahmesituation

1. Ist für die Ausnahmesituation eine USE-Prozedur angegeben, so wird diese aktiviert.
2. Wird das Ende der USE-Prozedur erreicht, ohne dass die USE-Prozedur zuvor explizit verlassen wurde, so wird der Programmablauf abgebrochen.
3. Ist keine USE-Prozedur angegeben, so wird der Programmablauf abgebrochen.

Auslösen einer NON-FATAL Ausnahmesituation

1. Ist für die Ausnahmesituation eine USE-Prozedur angegeben, so wird diese aktiviert.
2. Wird das Ende der USE-Prozedur erreicht, ohne dass die USE-Prozedur zuvor explizit verlassen wurde, so wird der Programmlauf abgebrochen.
3. Ist keine USE-Prozedur angegeben, so wird der Programmlauf fortgesetzt, als wäre die Überprüfung der Ausnahmesituation nicht eingeschaltet.

Ausnahmesituationsname

Die folgende Tabelle enthält eine Aufstellung der Ausnahmesituationsnamen, die von der aktuellen Compiler-Version unterstützt werden.

Bedeutung der einzelnen Spalten

Name:	Name der Ausnahmesituationen.
Kat:	Kategorie der Ausnahmesituation: *Fatal*, *Non-Fatal* (NF).
Kurzbeschreibung:	Umstand, unter dem die Ausnahmesituation auftritt
Klauseln:	Die Behandlung der Ausnahmesituation ist durch die angegebenen Klauseln möglich.

Name	Kat	Kurzbeschreibung	Klauseln

EC-DATA-CONVERSION	NF	Ersatzzeichen bei Konvertierung	-
EC-OO-CONFORMANCE	Fatal	Fehler bei Objektsicht	-
EC-OO-METHOD	Fatal	Methode nicht gefunden	ON EXCEPTION
EC-OO-NULL	Fatal	Aufruf einer Methode mit einer NULL-Objektreferenz	ON EXCEPTION
EC-OO-RESOURCE	Fatal	Nicht genügend Speicherplatz verfügbar, um ein Objekt zu erzeugen	-
EC-OO-UNIVERSAL	Fatal	Konformitätsregeln bei Methodenaufruf über universelle Objektreferenz sind nicht erfüllt	ON EXCEPTION
EC-STORAGE-NOT-ALLOC	NF	Zeiger in der FREE-Anweisung zeigt nicht auf einen durch ALLOCATE zugewiesenen Speicherbereich	-
EC-STORAGE-NOT-AVAIL	NF	Der in einer ALLOCATE-Anweisung angeforderte Speicherplatz ist nicht verfügbar	-
EC-XML-CODESET-CONVERSION	NF	Ersatzzeichen bei Konvertierung eines XML-Dokuments	-

Tabelle 45: Ausnahmesituationsnamen

12.3 Initial- und „last used"-Zustand

Initial-Zustand bedeutet für

- *Indizes:* undefiniert
- *Objektreferenzen/Zeiger*: Initialwert NULL
- *sonstigen Daten*: den Wert aus der VALUE-Klausel, wenn sie angegeben ist; sonst undefiniert
- *Dateien*: Datei im nicht geöffneten Zustand.

„Last used“-Zustand bedeutet bei

- *Daten*: Zustand wie nach der letzten, vorhergegangenen Änderung
- *Dateien*: Zustand wie nach der letzten, vorhergegangenen Operation.

In welchem Zustand sich Daten bzw. Dateien beim Aufruf einer Methode bzw. eines Programms befinden, hängt ab von

- der Section, in der sie definiert sind und von weiteren Klauseln der Definition
- dem bisherigen Geschehen beim Ablauf der Ablaufeinheit.

Section der Definition	**Bisheriger Ablauf**					
	Methode		**Programm**			
	erster Aufruf	**weiterer Aufruf**	**mit INITIAL** [1)]	**ohne INITIAL**		
			erster /weiterer Aufruf	**aller erster Aufruf**	**erster Aufruf nach Cancel** [2]	**Aufruf sonst**
Working-Storage mit EXTERNAL	initial last used	last used last used	initial last used	initial last used	initial last used	last used last used
Local-Storage	initial	initial	initial	initial	initial	initial
Linkage [3)]	last used	last used	last used	last used	last used	last used

[1)] das Programm selbst besitzt die Klausel oder ist in einem solchen Programm enthalten
[2)] Cancel auf das Programm selbst oder eines, in dem das Programm enthalten ist
[3)] bezieht sich auf die aktuell übergebenen Parameter

12.4 Programmkommunikation

Die Programmkommunikation mit COBOL2000 umfasst:

- Wechsel der Steuerung von einem Programm zu einem anderen mit der Möglichkeit, zwischen den einzelnen Programmen Parameter zu übergeben, mit denen einem aufgerufenen Programm Daten aus dem aufrufenden Programm verfügbar gemacht werden können,
- die Verwendung gemeinsamer Daten und Dateien durch die verschiedenen Programme einer Ablaufeinheit.

12.4.1 Begriffe

Getrennt übersetztes Programm (separately compiled program)

Ein vollständiges COBOL-Programm, das in einem eigenen Compilerlauf übersetzt wurde, wird als getrennt übersetztes Programm bezeichnet. Es kann sowohl ein einzelnes Programm als auch das äußerste Programm eines geschachtelten Programms sein.

Geschachteltes Programm (nested program)

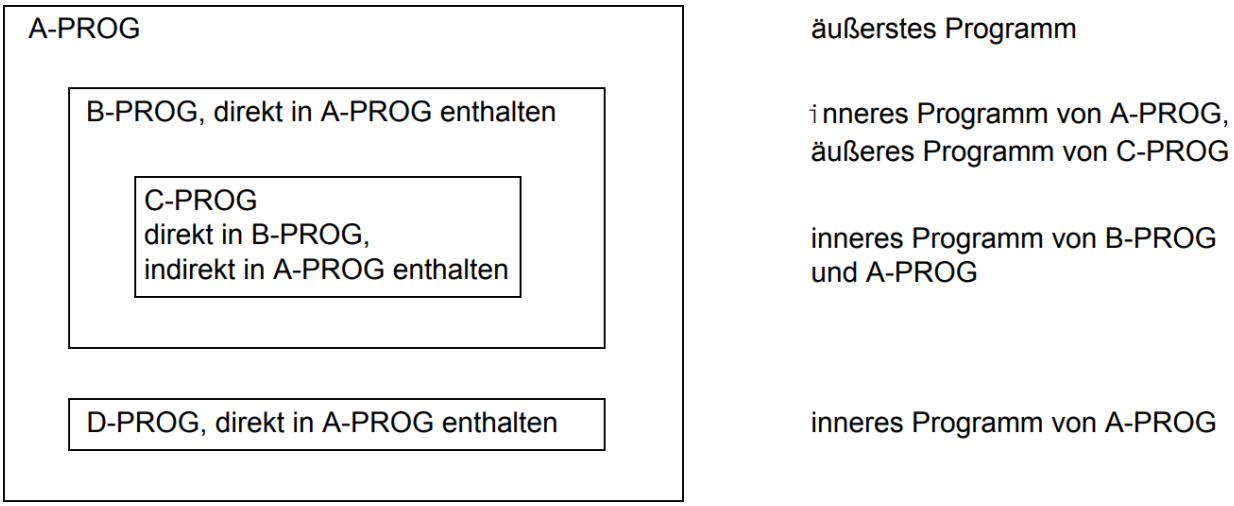
Ein COBOL-Programm, das aus mehreren ineinander geschachtelten, vollständigen Programmen besteht, wird als „geschachteltes Programm“ bezeichnet.
Ein Programm, das weitere Programme enthält, wird als „äußeres Programm“(containing program) bezeichnet.
Ein Programm, das in einem anderen Programm enthalten ist, wird als „inneres Programm“ (contained program) bezeichnet.
Die Programme eines geschachtelten Programms können sowohl äußere als auch innere Programme sein, ausgenommen das „äußerste“ Programm, das innerhalb der Ablaufeinheit genauso behandelt wird wie ein getrennt übersetztes Programm.

Ein inneres Programm kann in einem anderen Programm direkt oder indirekt enthalten sein.

Bezogen auf die unmittelbar übergeordnete Schachtelungsebene ist ein inneres Programm *direkt* enthalten, bezogen auf weitere übergeordnete Schachtelungsebenen ist es *indirekt* enthalten.

Beispiel 12-1

für die Struktur eines geschachtelten Programms



„Geschwisterprogramm“

Die Programme eines geschachtelten Programms, die auf derselben Schachtelungsebene in einem Programm enthalten sind, werden im Folgenden als „Geschwisterprogramme“ bezeichnet.

„Abkömmling“

Jedes direkt oder indirekt in einem „Geschwisterprogramm“ enthaltene Programm wird als „Abkömmling“ dieses „Geschwisterprogramms“ bezeichnet.

Ablaufeinheit (run unit)

Eine Ablaufeinheit ist eine bestimmte Anzahl von ablauffähigen Programmen, die zum Ablaufzeitpunkt als logische Einheit wirken.

Eine Ablaufeinheit kann bestehen

- aus einem oder mehreren Einzelprogrammen,
- aus einem oder mehreren geschachtelten Programmen,
- aus einer Kombination von Einzelprogrammen und geschachtelten Programmen.

Das auf Systemebene gestartete Programm wird als „Hauptprogramm“ bezeichnet, alle weiteren Programme der Ablaufeinheit als „Unterprogramme“.

12.4.2 Steuerung der Programmkommunikation

In diesem Kapitel werden folgende Themen behandelt:

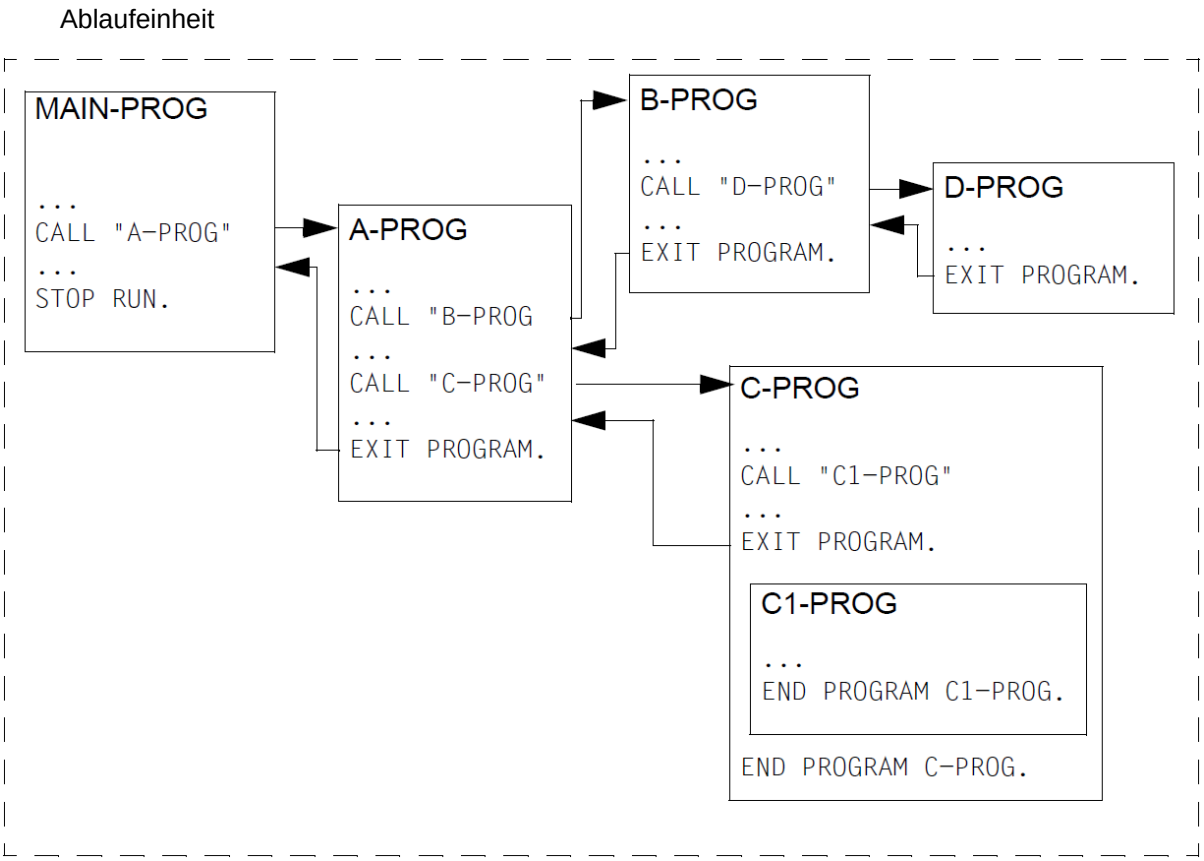
- Ablaufsteuerung

12.4.2.1 Ablaufsteuerung

Die Steuerung einer Ablaufeinheit beginnt bei dem Programm, das auf Systemebene aufgerufen wird. Jedes weitere Programm der Ablaufeinheit wird mit der CALL-Anweisung aufgerufen.
Die CALL-Anweisung übergibt die Programmsteuerung an das aufgerufene Programm. Von dort wird die Steuerung mittels der EXIT PROGRAM-Anweisung wieder an das aufru-fende Programm zurückgegeben. Das aufrufende Programm wird mit der auf die CALL-Anweisung folgenden Anweisung fortgesetzt.

Das folgende Beispiel zeigt die logische Struktur einer Ablaufeinheit, die aus fünf getrennt übersetzten Programmen, darunter einem geschachtelten Programm, besteht:

Beispiel 12-2



12.4.3 Regeln für Programmnamen

Der Aufruf eines Programms erfolgt über den Namen des Programms, der im PROGRAM-ID-Paragrafen der IDENTIFICATION DIVISION vereinbart wurde.

Für die Programmnamen gelten folgende grundsätzliche Regeln:

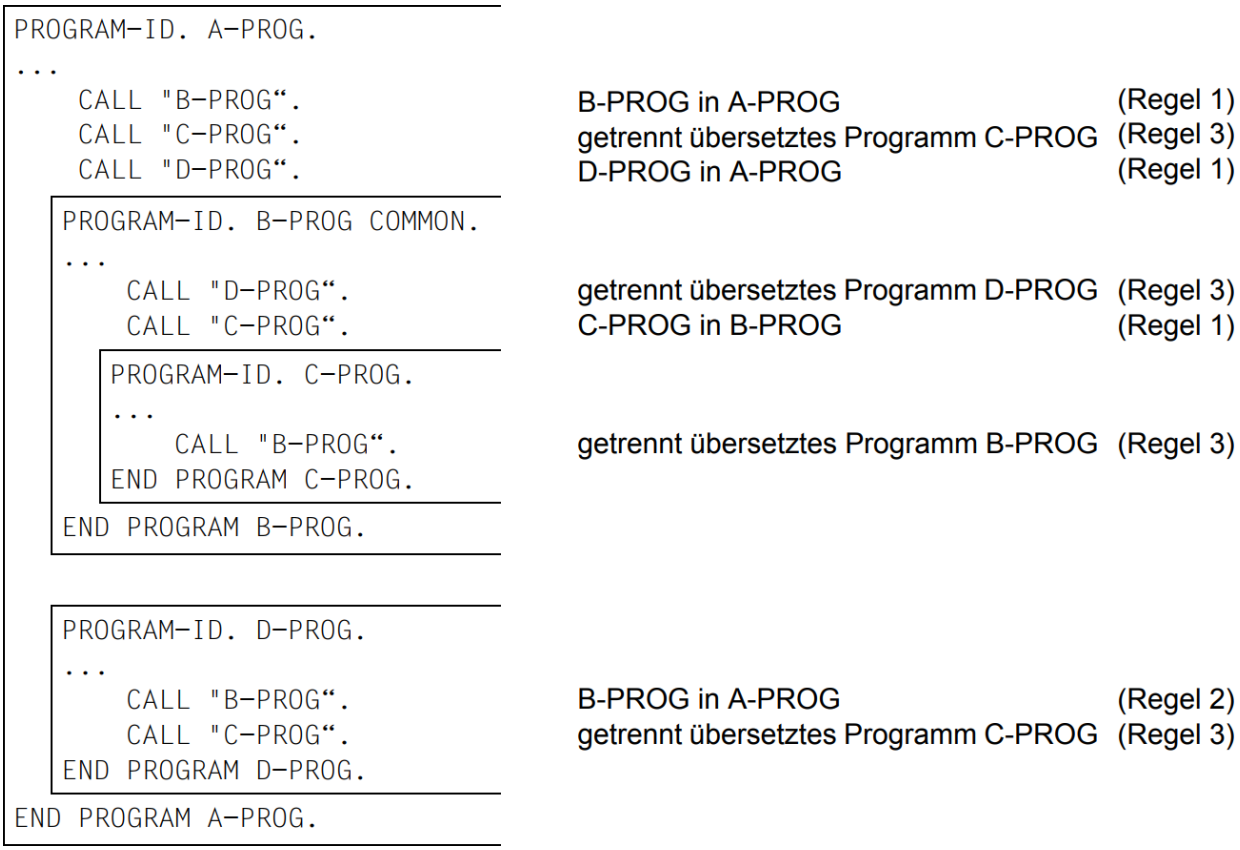
1. Auf einen Programmnamen dürfen sich nur die CALL-Anweisung, die CANCEL-Anweisung und der END PROGRAM-Eintrag beziehen.
2. Alle getrennt übersetzten Programme einer Ablaufeinheit müssen unterschiedliche Programmnamen haben.
3. Alle Programme eines geschachtelten Programms müssen unterschiedliche Programmnamen haben.
4. Die inneren Programme eines geschachtelten Programms sind für alle getrennt übersetzten Programme einer Ablaufeinheit nicht „sichtbar“; d.h. ein getrennt übersetztes Programm kann kein inneres Programm eines anderen getrennt übersetzten Programms aufrufen.
5. Die inneren Programme eines geschachtelten Programms können namensgleich mit den getrennt übersetzten Programmen der Ablaufeinheit sein. Das für diesen Fall vorgesehene Verfahren zur Bestimmung des gültigen Programmnamens ist im nächsten Abschnitt („Auswahl des gültigen Programmnamens“) dargestellt.
6. Innerhalb eines geschachtelten Programms kann ein Programm im Standardfall nur ein Programm aufrufen, das direkt in ihm enthalten ist.
7. Die Aufrufmöglichkeiten in einem geschachtelten Programm lassen sich gegenüber dem Standardfall erweitern, wenn einem inneren Programm mit der COMMON-Klausel im PROGRAM-ID-Paragrafen das COMMON-Attribut verliehen wird.
   Ein Programm, das mit dem Attribut COMMON versehen ist, kann nicht nur von dem direkt übergeordneten Programm aufgerufen werden, sondern auch von jedem „Geschwisterprogramm“ und dessen „Abkömmlingen“ (siehe „COMMON-Klausel“, "PROGRAM-ID-Paragraf ").

Auswahl des gültigen Programmnamens

Wird ein Programm in einem geschachtelten Programm aufgerufen, so wird der jeweils gültige Programmname aus der Menge aller in der Ablaufeinheit vorhandenen Programmnamen nach folgenden Präzedenzregeln ausgewählt:

1. Der Aufruf gilt dem Namen eines Programms, das direkt in dem aufrufenden Programm enthalten ist.
2. Wenn 1. nicht zutrifft, gilt der Aufruf einem COMMON-Programm, d.h. einem Programm, das von seinen „Geschwisterprogrammen“ und deren „Abkömmlingen“ aufgerufen werden kann.
3. Wenn weder 1. noch 2. zutrifft, gilt der Aufruf einem getrennt übersetzten Programm der Ablaufeinheit.

Beispiel 12-3



Dieses Beispiel zeigt die Auswahl des jeweils gültigen Programmnamens.

Eine CALL-Anweisung auf ein Programm „A-PROG“ innerhalb dieses geschachtelten Programms wäre unzulässig, da dieser Aufruf einem weiteren getrennt übersetzten Programm namens „A-PROG“ gelten würde, das in der Ablaufeinheit nicht vorhanden sein darf.

12.4.4 Initialzustand bei der Programmkommunikation

Das Attribut INITIAL wird durch Angabe der INITIAL-Klausel im PROGRAM-ID-Paragrafen des Programms erzeugt („INITIAL-Klausel“, "PROGRAM-ID-Paragraf ").

Der Initialzustand bedeutet Folgendes:

- Die internen Daten des Programms, die in der WORKING-STORAGE SECTION bzw. in der LOCAL-STORAGE SECTION definiert sind, sind initialisiert. Wird für ein Datenfeld die VALUE-Klausel verwendet, wird das Datenfeld mit dem definierten Wert versehen. Ist keine VALUE-Klausel angegeben, ist der Anfangswert des Datenfeldes unbestimmt.
- Zum Programm gehörende (interne) Dateien sind geschlossen.
- Die Steuerungsmechanismen für alle im Programm enthaltenen PERFORM-Anweisungen befinden sich im Initialzustand.
- Eine GO TO-Anweisung, auf die sich eine im selben Programm enthaltene ALTER-Anweisung bezieht, befindet sich im Initialzustand.

Ein Programm befindet sich im Initialzustand,

1. wenn es das erste Mal in einer Ablaufeinheit aufgerufen wird,
2. wenn es das erste Mal aufgerufen wird, nachdem es selbst oder ein Programm, in dem es direkt oder indirekt enthalten ist, Objekt einer CANCEL-Anweisung war,
3. bei jedem Aufruf, wenn es das INITIAL-Attribut besitzt.
4. wenn es direkt oder indirekt in einem Programm enthalten ist, das das INITIAL-Attribut besitzt, und wenn es nach dem Aufruf dieses Programms das erste Mal aufgerufen wird.

Beispiel 12-4

```
PROGRAM-ID. A-PROG.
...
    CALL "B-PROG".                  Anweisung 1
    CALL "B-PROG".                  Anweisung 2
...
  PROGRAM-ID. B-PROG INITIAL.
  ...
      CALL "C-PROG".                Anweisung 3
      CALL "D-PROG".                Anweisung 4
  ...
    PROGRAM-ID. C-PROG COMMON.
    ...
    END PROGRAM C-PROG.

    PROGRAM-ID. D-PROG.
    ...
        CALL "C-PROG".              Anweisung 5
        CANCEL "C-PROG".
        CALL "C-PROG".              Anweisung 6
    ...
    END PROGRAM D-PROG.
  END PROGRAM B-PROG.
END PROGRAM A-PROG.
```

Anweisung 1: Programm B-PROG befindet sich im Initialzustand (Regel 1 und 3).

Anweisung 2: Programm B-PROG befindet sich im Initialzustand (Regel 3).

Anweisung 3: Programm C-PROG befindet sich im Initialzustand (Regel 1 und 4).

Anweisung 4: Programm D-PROG befindet sich im Initialzustand (Regel 1 und 4)

Anweisung 5: Programm C-PROG befindet sich nicht im Initialzustand.

Anweisung 6: Programm C-PROG befindet sich im Initialzustand (Regel 2).

12.4.5 Verwendung gemeinsamer Daten

In diesem Kapitel werden folgende Themen behandelt:

- Externe und interne Daten
- Lokale und globale Namen

12.4.5.1 Externe und interne Daten

In einem Programm definierte Datenfelder und Dateien können interne oder externe Daten sein.

Auf ein internes Datum kann nur innerhalb des Programms zugegriffen werden, in dem das Datum definiert ist. Auf ein externes Datum kann von jedem Programm der Ablaufeinheit zugegriffen werden.

Ein internes Datum belegt einen Speicherbereich, der ausschließlich mit demjenigen Programm verbunden ist, in dem das Datum definiert wurde. Ein externes Datum dagegen belegt einen Speicherbereich, der mit der Ablaufeinheit verbunden ist. Auf Daten, die in den Programmen einer Ablaufeinheit als extern vereinbart sind, kann von allen Programmen der Ablaufeinheit zugegriffen werden.

Externe Dateien und Datenfelder werden durch die Angabe der EXTERNAL-Klausel in der FILE SECTION bzw. WORKING-STORAGE SECTION erzeugt (siehe „EXTERNAL-Klausel").

Beispiel 12-5

Eine Ablaufeinheit besteht aus drei Programmen, in denen einige Daten als extern definiert sind. Die Speicherbelegung gestaltet sich - schematisch dargestellt - folgendermaßen:

Speicherbereich Programm A
Speicherbereich Programm B
Speicherbereich Programm C
Speicherbereich Externe Daten

Alle internen Daten, d.h. solche, die nicht als extern definiert sind, stehen nur im Speicherbereich des Programms, in dem sie definiert sind. Alle als extern definierten Daten dagegen stehen nur im Speicherbereich für externe Daten.

12.4.5.2 Lokale und globale Namen

Die in einem *geschachtelten* Programm verwendeten Namen lassen sich unterscheiden in „lokale“ und „globale“ Namen.

Lokale Namen

Lokale Namen sind nur innerhalb des Programms gültig, in dem die zugehörigen Daten beschrieben sind. Die meisten standardmäßig lokalen Namen können als global vereinbart werden. Stets lokale Namen sind:

- Paragrafennamen
- Kapitelnamen

Globale Namen

Globale Namen sind nicht nur in dem Programm gültig, in dem sie definiert sind, sondern darüberhinaus auch in dessen inneren Programmen. Stets globale Namen sind:

- Namen, die in der CONFIGURATION SECTION vereinbart werden:

  Rechenanlagenname
  Alphabetnamen
  Klassennamen
  Bedingungsnamen (SPECIAL-NAMES-Paragraf)
  Merknamen
  symbolische Zeichen
  CURRENCY SIGN-Zeichen
  DECIMAL-POINT
- COBOL-Sonderregister:

  TALLY
  PRINT-SWITCH
  SORT-Register
  RETURN-CODE

Die folgenden Namenstypen sind standardmäßig lokal und können in geschachtelten Programmen explizit als global vereinbart werden:

- Bedingungsnamen
- Datennamen
- Dateinamen
- Indexnamen
- Datensatznamen
- Listennamen
- Typnamen

Die Vereinbarung eines Namens als global erfolgt mit der GLOBAL-Klausel (siehe "GLOBAL-Klausel " und "GLOBAL-Klausel ").

Bei Typnamen bezieht sich die Angabe GLOBAL auf die Definition des Typs und nicht auf die mit Hilfe der Typdefinition erzeugten Datenbeschreibung.

Bestimmung des gültigen Namens

Wird ein Name referenziert, so wird der gültige Name aus der Menge aller in dem geschachtelten Programm definierten Namen nach folgenden Präzedenzregeln ausgewählt:

1. Der referenzierte Name ist im selben Programm definiert.

2. Wenn 1. nicht zutrifft, ist der referenzierte Name im *direkt* übergeordneten (nächstäußeren) Programm als *globaler* Name definiert.
3. Wenn weder 1. noch 2. zutrifft, ist der referenzierte Name im *indirekt* übergeordneten Programm als *globaler* Name definiert.

   Bedingung 3. wird solange geprüft, bis die Datenerklärung des referenzierten Namens in einem der weiteren indirekt übergeordneten Programme gefunden ist, ggf. erst im äußersten Programm.

Beispiel 12-6

```
PROGRAM-ID. A-PROG.
...
01  A1   PIC X  GLOBAL.
01  B1   PIC X  GLOBAL.
01  C1   PIC X  GLOBAL.
...
  PROGRAM-ID. B-PROG.
  ...
  01  A1   PIC X  GLOBAL.
  01  B1   PIC X.
  ...
      MOVE  "A"  TO  A1.          A1 in Programm B-PROG (Regel 1)
      MOVE  "B"  TO  B1.          B1 in Programm B-PROG (Regel 1)
      MOVE  "C"  TO  C1.          C1 in Programm A-PROG (Regel 2)
  ...
    PROGRAM-ID. C-PROG.
    ...
        MOVE  "X"  TO  A1.        A1 in Programm B-PROG (Regel 2)
        MOVE  "Y"  TO  B1.        B1 in Programm A-PROG (Regel 3)
        MOVE  "Z"  TO  C1.        C1 in Programm A-PROG (Regel 3)
    ...
    END PROGRAM C-PROG.
  END PROGRAM B-PROG.
END PROGRAM A-PROG.
```

Beispiel 12-7

```
PROGRAM-ID. A-PROG.
...
01 T1 TYPEDEF STRONG GLOBAL.
   02 NAME PIC X(30).
   02 STRASSE PIC X(80).
...
01 A1 TYPE T1.                1)
01 B1 TYPE T1 GLOBAL.
...
PROGRAM-ID. B-PROG.
...
01 T1 TYPEDEF STRONG.         2)
   02 UNTERSTRUKTUR.
     05 NAME-3 PIC X(30).
     05 STRASSE-3 PIC X(80).
...
```

```
01 C1 TYPE T1.                3)
...
MOVE B1 TO C1.                4)
...
END PROGRAM B-PROG.
END PROGRAM A-PROG.
```

1. verwendet Typdefinition T1 aus A-PROG;
   Definition A1 ist lokal in A-Prog, Definition B1 ist global
2. Typ T1 ist lokal in B-PROG
3. verwendet Typdefinition T1 aus B-PROG
4. Zuweisung ist zulässig, da T1 in A-PROG äquivalent zu T1 in B-PROG

12.4.6 Sprachelemente für die Programmkommunikation

Dieses Kapitel gibt eine Übersicht über die Sprachelemente für die Programmkommunikation.

12.4.6.1 Übersicht

Die für die Programmkommunikation relevanten Sprachelemente sind in der folgenden Tabelle zusammengefasst:

Sprachelement	**Funktion**
END PROGRAM-Eintrag	Bezeichnet das Ende der benannten Übersetzungseinheit
INITIAL-Klausel	Der Initialzustand des Programms wird bei jedem Aufruf des Programms hergestellt.
COMMON-Klausel	Das Programm kann auch von seinen Geschwisterprogrammen und deren Abkömmlingen aufgerufen werden.
LINKAGE SECTION	Im aufgerufenen Programm: zur Definition der Daten, die vom aufrufenden Programm übergeben werden
EXTERNAL-Klausel	Deklarieren von Dateien und Datenfeldern als extern
GLOBAL-Klausel	Deklarieren von Namen als global
PROCEDURE DIVISION USING-Angabe	Im aufgerufenen Programm: Definieren der Standard-Einsprungstelle Liste von Datennamen; zeigt an, dass Daten vom aufrufenden Programm übergeben werden
CALL-Anweisung USING-Angabe	Im aufrufenden Programm: Aufruf eines Unterprogramms bzw. einesinneren Programms; Liste von Datennamen; zeigt an, dass Daten an das aufgerufene Programm übergeben werden
CANCEL-Anweisung	Herstellen des Initialzustands für den nächsten Aufruf des Programms
ENTRY-Anweisung USING-Angabe	Nur im Unterprogramm einer Ablaufeinheit: Definieren einer NichtStandard-Einsprungstelle Liste von Datennamen (definiert in der LINKAGE SECTION); zeigt an, dass Daten vom aufrufenden Programm übergeben werden
EXIT PROGRAM-Anweisung	Rückgabe der Steuerung an das aufrufende Programm
GOBACK-Anweisung	Kennzeichnet das logische Ende eines Programms
USE-Anweisung mitGLOBAL-Attribut	In geschachtelten Programmen: Vereinbarung globaler USE-Prozeduren

12.5 Sortieren von Datensätzen

Datensätze können auf zweierlei Weise sortiert werden:

- Sortieren von Datensätzen, die in einer Datei vorliegen (Datei-Sortieren),
- Sortieren von Datensätzen, die als Elemente einer Tabelle vorliegen (Tabellen-Sortieren).

Beide Sortierarten verwenden für den Sortiervorgang das BS2000-Dienstprogramm SORT. Das Datei-Sortieren ist im folgenden Abschnitt, das Tabellen-Sortieren im Kapitel „PROCEDURE DIVISION" im Abschnitt „SORT-Anweisung". beschrieben.

12.5.1 Sortieren und Mischen von Dateien

In diesem Kapitel werden folgende Themen behandelt:

- Ablauf eines Sortiervorgangs
- Ablauf eines Mischvorgangs
- Sortieren und Mischen ohne Ein-/Ausgabeprozeduren
- Sortieren mit Ein-/Ausgabeprozeduren
- Übersicht über die Sprachelemente

12.5.1.1 Ablauf eines Sortiervorgangs

Der Benutzer kann eine Datei nach einer Anzahl von Datenfeldern (Sortierschlüssel) sortieren. Diese Sortierschlüssel werden vom Benutzer festgelegt und sind in jedem Datensatz der Datei vorhanden. Die Datensätze können so sortiert werden, dass entweder alle Schlüssel aufsteigend oder absteigend sind oder dass einige Schlüssel absteigende und andere aufsteigende Reihenfolge haben.

Die Datei, in der die Datensätze sortiert werden, heißt Sortierdatei. Sie wird in der DATA DIVISION innerhalb einer Sortierdateierklärung (SD) und in der zugehörigen Datensatzbeschreibung definiert. Die SORT-Anweisung in der PROCEDURE DIVISION veranlasst den Sortiervorgang.

Ablauf eines Sortiervorgangs

1. Übergabe aller Datensätze an die Sortierdatei.
2. Sortieren der Datensätze in der Sortierdatei.
3. Rückgabe aller Datensätze aus der Sortierdatei.

Der Benutzer kann entweder eine Eingabeprozedur angeben, mit der Datensätze verarbeitet und an die Sortierdatei übergeben werden, oder er kann eine Eingabedatei spezifizieren, die die zu sortierenden Datensätze enthält und die es der SORT-Anweisung überlässt, diese Datensätze an die Sortierdatei zu übergeben. Dementsprechend kann der Benutzer eine Ausgabeprozedur angeben, die die Datensätze aus der Sortierdatei übernimmt und weiterverarbeitet, oder er kann eine Ausgabedatei spezifizieren, in die die SORT-Anweisung die sortierten Datensätze abliefert.

Innerhalb eines Programms können mehrere Sortiervorgänge angegeben werden.

12.5.1.2 Ablauf eines Mischvorgangs

Der Benutzer kann zwei oder mehrere (max. 16) sortierte Eingabedateien mit demselben Datensatzformat in eine Ausgabedatei mischen.

Die Datei, in der Datensätze gemischt werden, heißt Sortierdatei. Sie wird in der DATA DIVISION innerhalb einer Sortierdateierklärung (SD) und in der zugehörigen Datensatzbeschreibung definiert. Die MERGE-Anweisung in der PROCEDURE DIVISION veranlasst den Mischvorgang. Der Mischvorgang erfolgt in drei Schritten:

1. Übergabe der Datensätze aller Eingabedateien an die Sortierdatei.
2. Mischen der Datensätze in der Sortierdatei.
3. Rückgabe aller Datensätze aus der Sortierdatei.

Die MERGE-Anweisung übergibt die Datensätze aller spezifizierten Eingabedateien an die Sortierdatei. Der Benutzer kann eine Ausgabedatei spezifizieren in die die MERGE-Anweisung die gemischten Datensätze abliefert. Für die Rückgabe besteht auch die Möglichkeit, eine Ausgabeprozedur anzugeben. Diese übernimmt die Datensätze aus der Sortierdatei und verarbeitet sie weiter.
Innerhalb eines Programms können mehrere Mischvorgänge angegeben werden.

12.5.1.3 Sortieren und Mischen ohne Ein-/Ausgabeprozeduren

Sind keine Ein-/Ausgabeprozeduren angegeben, erzeugt der Compiler Prozeduren, die die Ein-/Ausgabe der Datensätze übernehmen. In diesem Fall muss der Benutzer folgende Dateien beschreiben:

1. eine oder mehrere Eingabedateien, die die zu sortierenden oder zu mischenden Datensätze enthält,
2. eine Sortierdatei und eine oder mehrere Ausgabedateien, in die die Datensätze abgeliefert werden.

Wird eine SORT- oder MERGE-Anweisung angesprochen, werden folgende Funktionen ausgeführt:

1. Eröffnen der Eingabe- und Sortierdateien.
2. Übernahme aller Datensätze aus Eingabedateien in die Sortierdatei.
3. Schließen der Eingabedateien
4. Sortieren oder Mischen der Datensätze in der Sortierdatei.
5. Eröffnen der Ausgabedateien.
6. Übergabe der Datensätze aus der Sortierdatei in die Ausgabedateien.
7. Schließen der Sortier- und Ausgabedateien.

12.5.1.4 Sortieren mit Ein-/Ausgabeprozeduren

Sind Ein-/Ausgabeprozeduren angegeben, werden folgende Schritte durchgeführt:

1. Sobald eine SORT-Anweisung ausgeführt wird, geht die Steuerung an die Eingabeprozedur über.
2. Die Eingabeprozedur muss folgende Schritte durchführen:
   a. Verarbeitung eines Datensatzes (z.B. Lesen eines Datensatzes aus einer Datei oder Erstellen eines neuen Datensatzes).
   b. Übergabe des Datensatzes an die Sortierdatei.
   c. Wiederholen der Schritte a und b, bis alle Datensätze übergeben sind.
3. Die SORT-Anweisung sortiert die Datensätze in der Sortierdatei.
4. Die Steuerung wird dann an die Ausgabeprozedur abgegeben.
5. Die Ausgabeprozedur führt folgende Schritte durch:
   a. Übernehmen eines Datensatzes aus der Sortierdatei.
   b. Verarbeitung des Datensatzes (z.B. Schreiben des Datensatzes in eine Ausgabedatei).
   c. Wiederholen der Schritte a und b, bis alle Datensätze übergeben und verarbeitet sind.
6. Die Steuerung wird an die der SORT-Anweisung folgenden Anweisung weitergegeben.

Sind die Angaben gemischt (z.B. nur eine Eingabeprozedur angegeben), so werden die oben beschriebenen Prozeduren entsprechend abgewandelt durchgeführt.

12.5.1.5 Übersicht über die Sprachelemente

Um die Sortier- und Mischfunktionen anwenden zu können, muss der Benutzer zusätzliche Informationen in der ENVIRONMENT DIVISION, DATA DIVISION und PROCEDURE DIVISION einer Übersetzungseinheit angeben. Diese Informationen sind nachfolgend tabellarisch zusammengefasst und im Einzelnen beschrieben.

<table>
<tr><th>Teil der Übersetzungseinheit</th><th colspan="2">Inhalt und Bedeutung</th></tr>
<tr><td rowspan="4">ENVIRONMENT DIVISION</td><td colspan="2">Eine SELECT-Klausel im FILE-CONTROL-Paragrafen für die Sortierdatei (sortierdateiname).</td></tr>
<tr><td colspan="2">SELECT-Klauseln im FILE-CONTROL-Paragrafen für alle Dateien, die in Ein- und Ausgabeprozeduren für die Sortierdatei benutzt werden (kapitelname-1, kapitelname-2, kapitelname-3, kapitelname-4 für SORT, kapitelname-1, kapitelname-2 für MERGE) sowie für Dateien, die in der USING-/GIVING-Angabe einer SORT- oder MERGE-Anweisung vorkommen (dateiname-1,...).</td></tr>
<tr><td colspan="2">Um einen Wiederanlauf von Programmen, die die SORT-Anweisung enthalten, zu ermöglichen, kann die RERUN-Klausel im I-O-CONTROL-Paragrafen verwendet werden.</td></tr>
<tr><td colspan="2">Die SAME SORT AREA- bzw. die SAME SORT-MERGE AREA-Klausel im I-O-CONTROL-Paragrafen ist dazu bestimmt, die Zuweisung von Arbeitsspeicher für eine Sortierdatei zu optimieren.</td></tr>
<tr><td rowspan="3">DATA DIVISION</td><td colspan="2">Eine Sortierdateierklärung (SD) und die dazugehörigen Datensatzerklärungen für eine Sortierdatei. Die Datensatzerklärungen müssen die Sortierschlüssel enthalten.</td></tr>
<tr><td colspan="2">Dateierklärungen (FD) und die dazugehörigen Datensatzerklärungen für alle Dateien, die in Ein- und Ausgabeprozeduren für die Sortierdatei benutzt werden (kapitelname-1, kapitelname-2, kapitelname-3, kapitelname-4 für SORT, kapitelname-1, kapitelname-2 für MERGE) sowie für Dateien, die in der USING- oder GIVING-Angabe einer SORT- oder MERGE-Anweisung vorkommen (dateiname-1,...).</td></tr>
<tr><td colspan="2">Datenerklärungen für die oben genannten Dateien.</td></tr>
<tr><td rowspan="5">PROCEDURE DIVISION</td><td colspan="2">Eine SORT- bzw. MERGE-Anweisung für die Sortierdatei mit folgender Information:
Namen der Sortierschlüssel.
Angaben, ob in auf- oder absteigender Reihenfolgesortiert weden soll.
Ein-/Ausgabe-Informationen, die wie folgt angegebenwerden:</td></tr>
<tr><td>Angabe</td><td>Bedeutung</td></tr>
<tr><td>INPUT PROCEDURE</td><td>Die Datensätze werden von einer Eingabeprozedur an die Sortierdatei übergeben.</td></tr>
<tr><td>USING dateiname-1,...</td><td>Angabe der Dateien, von denen Datensätze übernommen und der Sortierdatei übergeben werden sollen.</td></tr>
<tr><td>OUTPUT PROCEDURE</td><td>Die Datensätze werden einer Ausgabeprozedur aus der Sortierdatei zurückgegeben.</td></tr>
<tr><td>GIVING dateiname-3,...</td><td>Angabe der Dateien, in die die SORT- bzw. MERGE-Anweisung die sortierten bzw. gemischten Sätze übergeben soll.</td></tr>
</table>

	Falls Eingabe- und/oder Ausgabeprozeduren in der SORT- bzw. MERGE-Anweisung angegeben sind, müssen diese Prozeduren in der PROCEDURE DIVISION enthalten sein.Eine Eingabeprozedur muss eine RELEASE-Anweisung enthalten, um Sätze an die Sortierdatei zu übergeben. Eine Ausgabeprozedur muss eine RETURN-Anweisung enthalten, um Sätze von der Sortierdatei zu übernehmen.
	Besondere Sortierregister können in der PROCEDURE DIVISION angesprochen werden (siehe „Sonderregister für Dateien-SORT").

12.5.2 Sonderregister für Dateien-SORT

Im Compiler sind vier Sonderregister für die Kommunikation zwischen dem Programmierer und der SORT-Steuerroutine vorgesehen. Jedes dieser Register ist ein vier Byte langer binärer Datenbereich mit festem Namen; jedes Register wird mit PICTURE S9(8) USAGE IS COMPUTATIONAL beschrieben. Die Datenerklärungen für die Register werden automatisch vom Compiler erzeugt und dürfen vom Programmierer nicht angegeben werden.

SORT-FILE-SIZE-Sonderregister

Der Programmierer kann als Inhalt von SORT-FILE-SIZE die ungefähre Anzahl von Datensätzen angeben, die beim nächsten Sortiervorgang verarbeitet werden sollen. Dies muss vor Ausführung der SORT-Anweisung geschehen. Mit Hilfe dieser Information berechnet SORT den benötigten Speicherplatz für interne Arbeitsdateien. SORT-FILE-SIZE wird vom Compiler auf den Anfangswert Null gesetzt.
Falls der Inhalt des Sonderregisters zur Anfangszeit des Sortiervorgangs immer noch Null ist, nimmt SORT die Speicherzuweisung auf Grund eines angenommenen Wertes vor. Deshalb braucht der Programmierer in das Sonderregister keinen Wert für die Dateigröße einzutragen, obwohl die Angabe dieser Information zur Effektivität des Sortiervorganges beiträgt.
Der Wert, den der Benutzer in das Sonderregister eingetragen hat, z.B. MOVE 25 TO SORT-FILE-SIZE, wird der Steuerroutine übergeben.

SORT-CORE-SIZE-Sonderregister

Der Programmierer kann als Inhalt von SORT-CORE-SIZE die Anzahl von Bytes des Internspeichers angeben, die SORT zur Verfügung stehen sollen. Dies muss vor Ausführung der SORT-Anweisung geschehen.

SORT-MODE-SIZE-Sonderregister

SORT-MODE-SIZE kann gesetzt werden, wenn Datensätze variabler Länge sortiert werden sollen.
Der Programmierer kann in dieses Sonderregister die am häufigsten auftretende Satzlänge der zu sortierenden Sätze eintragen. Der Wert darf nicht kleiner als die kleinste und nicht größer als die größte Angabe in der Sortierdateierklärung sein. Der ungefähre Wert muss vor Ausführung der SORT-Anweisung in SORT-MODE-SIZE eingetragen werden. Der Compiler setzt das Register auf den Anfangswert Null.
Falls der Programmierer vor Ausführung der SORT-Anweisung keinen anderen Wert angegeben hat, wird die maximale Satzlänge als die am häufigsten auftretende Satzlänge angenommen.

SORT-RETURN-Sonderregister

Nachdem eine SORT-Anweisung bzw. RELEASE/RETURN-Anweisung ausgeführt ist, enthält SORT-RETURN einen Wert, der angibt, ob der Sortiervorgang erfolgreich durchgeführt wurde. Der Wert Null bedeutet, dass der Sortiervorgang erfolgreich durchgeführt wurde. Ein Wert ungleich Null bedeutet, dass der Sortiervorgang nicht ordnungsgemäß abgeschlossen werden konnte.

Zusammenfassung

Der Wert der Sonderregister (außer SORT-RETURN) wird durch Ausführung der SORT-Anweisung nicht zurückgesetzt oder auf Null gesetzt. Falls ein Programm mehrere Sortierläufe enthält, muss der Programmierer vor Ausführung eines jeden Sortiervorgangs einen ungefähren Wert in SORT-FILE-SIZE, SORT-CORE-SIZE und SORT-MODE-SIZE eintragen.

Man beachte, dass die Sonderregister nicht als direkte Operanden einer ACCEPT-Anweisung angegeben werden können, da sie binäre Felder sind.

Beispiel 12-8

```
ACCEPT MODE-SIZE FROM EINGABE.
MOVE MODE-SIZE TO SORT-MODE-SIZE.
```

Da die ACCEPT-Anweisung Eingabedaten vom Terminal ohne Konvertierung überträgt, wird die MOVE-Anweisung für die Konvertierung nach COMPUTATIONAL verwendet.

12.5.3 Sortieren zweistelliger Jahreszahlen mit Jahrhundertfenster

Allgemeine Beschreibung

Die folgende Beschreibung ist sowohl für den Dateien- als auch für den Tabellensort gültig. Das Jahrhundertfenster wird wie in der COBOL-Funktion YEAR-TO-YYYY festgelegt. Das letzte Jahr, das noch zu diesem Fenster gehört, wird relativ zum aktuellen Jahr angegeben. Der angegebene Wert bestimmt die Anzahl der zukünftigen Jahre, die zum Jahrhundertfenster gehören. So bedeutet der Wert 50 bei Ablauf in 1998 den Zeitraum von 1949 bis 2048.

Zur Festlegung des Jahrhundertfensters wird ein SORT-Sonderregister SORT-EOW (SORT-END-OF-WINDOW) definiert. Es wird in COBOL-Programmen mit der SORT- bzw. MERGE-Anweisung zur Verfügung gestellt und durch den Compiler implizit mit PIC 9(7) PACKED-DECIMAL beschrieben. Der in SORT-EOW gespeicherte Wert muss zwischen 0 und 99 liegen. Standardwert ist 50.

Um zweistellige Jahreszahlen, in Abhängigkeit von einem Jahrhundertfenster, als SORT-Schlüssel verwenden zu können, sind die Angaben ASCENDING/DESCENDING in der SORT- und MERGE-Anweisung erweitert worden.

Bei der MERGE-Anweisung wird zu Beginn der Verarbeitung die Lage des Jahrhundertfensters festgelegt (Auswertung SORT-EOW und aktuelles Jahr). Werden z.B. Dateien voher mit der SORT-Anweisung sortiert, so muss dafür dasselbe Jahrhundertfenster gewählt werden.

Beispiel 12-9

SORT mit Auswahl des Jahrhundertfensters

```
IDENTIFICATION DIVISION.
PROGRAM-ID.SORTIERY.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT EINGABE ASSIGN TO "EINGABE".
   SELECT AUSGABE ASSIGN TO "AUSGABE".
   SELECT SORTIER ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD EINGABE.
01 ESATZ.
   02 E0      PIC X.
   02 EY1     PIC 99.
   02 EY2     PIC 99 USAGE PACKED-DECIMAL.
   02 E3      PIC X(10).
FD AUSGABE.
01 ASATZ.
   02 A0      PIC X.
   02 AY1     PIC 99.
   02 AY2     PIC 99 USAGE PACKED-DECIMAL.
   02 A3      PIC X(10).
SD SORTIER.
01 SSATZ.
   02 S0       PIC X.
   02 SY1      PIC 99.
   02 SY2      PIC 99 USAGE PACKED-DECIMAL.
   02 S3       PIC X(10).
PROCEDURE DIVISION.
P1 SECTION.
SORTIEREN.
```

```
* Festlegung des Jahrhundertfensters: bei Ablauf im Jahr
* 1998 ist 2008 das letzte Jahr des Jahrhundertfensters
     MOVE 10 TO SORT-EOW
* Die Schlüssel SY1 und SY2 werden als zweistellige Jahres-
* zahlen innerhalb des Jahrhundertfensters von 1909-2008
* behandelt; 06 ist größer als 75
     SORT SORTIER ASCENDING KEY-YY SY1 SY2
                  DESCENDING KEY S3
     USING EINGABE GIVING AUSGABE.
SORTEND.
     STOP RUN.
```

12.5.4 Sortieren mit erweiterten Zeichensätzen (XHCS)

Die folgende Beschreibung ist sowohl für den Dateien- als auch für den Tabellensort gültig.

Zur Festlegung des Zeichensatzes für alphanumerische Sortierschlüssel wird ein SORT-Sonderregister SORT-CCSN („Coded Character Set Name“) definiert. Es wird in COBOL-Programmen, die eine SORT-Anweisung enthalten, zur Verfügung gestellt und durch den Compiler implizit mit PIC X(8) beschrieben. Dieses SORT-Sonderregister ist vor Aufruf der SORT-Anweisung mit dem Namen des gewählten Zeichensatzes zu versorgen (z.B. MOVE "EDF041" TO SORT-CCSN).

Soll für eine nachfolgende SORT-Anweisung diese Option nicht gelten, so muss der Wert des neuen SORT-Sonderregisters SORT-CCSN vor der Ausführung dieser SORT-Anweisung mit Leerzeichen gefüllt werden (z. B. MOVE SPACES TO SORT-CCSN).

Die Angaben zur Sortierfolge gelten in folgender Reihenfolge:

- Angabe der SORT COLLATING SEQUENCE in der SORT-Anweisung bzw. PROGRAM COLLATING SEQUENCE im Programm
- Wert im SORT-Sonderregister SORT-CCSN
- COMOPT SORT-EBCDIC-DIN bzw. SDF-Option SORTING-ORDER

Beispiel 12-10

SORT mit erweiterten Zeichensätzen (XHCS)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORTIERX.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT EINGABE ASSIGN TO "EINGABE".
SELECT AUSGABE ASSIGN TO "AUSGABE".
SELECT SORTIER ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD EINGABE LABEL RECORD STANDARD.
01 ESATZ.
02 E0 PIC X.
02 NAME     PIC X(30).
02 VORNAME  PIC X(30).
02 ORT      PIC X(30).
FD AUSGABE LABEL RECORD STANDARD.
01 ASATZ
02 A0 PIC X.
02 NAME     PIC X(30).
02 VORNAME  PIC X(30).
02 ORT      PIC X(30).
SD SORTIER LABEL RECORD STANDARD.
01 SSATZ.
02 S0 PIC X.
02 NAME     PIC X(30).
02 VORNAME  PIC X(30).
02 ORT      PIC X(30).
PROCEDURE DIVISION.
P1 SECTION.
SORTIEREN.
MOVE "EDF03IRV" TO SORT-CCSN.                    (1)
```

```
SORT SORTIER ASCENDING NAME VORNAME ORT
USING EINGABE GIVING AUSGABE.
STOP RUN.
```

(1) versorgt das SORT-Sonderregister SORT-CCSN mit dem Namen des „Extended character Code Set“; dieser Code Set wird benutzt, um die Sortierfolge festzulegen.

12.6 Zeichendarstellung durch UTF-16

Zeichen, die ein COBOL Programm verarbeitet, können durch unterschiedliche Zeichensätze dargestellt werden. Die Sprache definiert dafür die beiden Datenklassen alphanumerisch und national.

COBOL2000 stellt alphanumerische Zeichen im Zeichensatz EBCDIC dar, nationale Zeichen in UTF-16. Definition und Verwendung von Daten der Klasse national erfolgen weitgehend analog denen der Klasse alphanumerisch.

12.6.1 Nationale Daten

Ein nationales Zeichen wird in der PICTURE Klausel durch das Symbol 'N' beschrieben. Damit wird die Anzahl der Zeichen angegeben, d.h. nicht die Anzahl der belegten Bytes. Ein nationales Zeichen belegt 2 Bytes und wird in Datenstrukturen auf die Bytegrenze ausgerichtet.

Teilfeldselektion ist auch für nationale Daten erlaubt. In diesem Fall zählen für die Startposition bzw. die Länge ebenfalls die Zeichen.

Beispiel 12-11

```
01 nat    PIC N(30).
```

definiert ein Datenfeld für 30 Zeichen in UTF-16 Darstellung, d.h. 60 Bytes

```
nat(3:4)
```

selektiert das 3te bis 6te Zeichen von nat, d.h. die Bytes 5 bis 12

12.6.2 Datenstrukturen, Klauseln

Das Datenformat national kann zusätzlich zur PICTURE Klausel auch in der USAGE-Klausel explizit als NATIONAL angegeben werden.

Nationale Daten dürfen auch zu Datenstrukturen gruppiert werden. Dabei ist zu berücksichtigen, dass Datengruppen in COBOL immer von der Klasse alphanumerisch sind. Dadurch wird eine Datengruppe wie ein alphanumerisches Datenfeld behandelt, auch wenn sie nur elementare nationale Datenfelder enthält, z.B. beim Auffüllen mit Leerzeichen. Die Klausel GROUP-USAGE NATIONAL bewirkt, dass auch eine Datengruppe wie ein nationales Datenfeld behandelt wird.

Beispiel 12-12

```
01 alfanum-struktur.
 02 nat1 PIC N(10).
 02 nat2 PIC N(10).
01 national-struktur GROUP-USAGE NATIONAL.
 02 nat1 PIC N(10).
 02 nat2 PIC N(10).
```

Die erste Datenstruktur ist immer alphanumerisch. Wird ein kürzeres Sendefeld dorthin übertragen, werden alphanumerische Leerzeichen (X'40') angefügt, die in nationalen Datenfeldern nicht sinnvoll sind.
Die zweite Datenstruktur ist dagegen wegen der zusätzlichen Klausel national. Es würden nationale Leerzeichen (X'0020') angefügt werden.

12.6.3 Nationale Literale

Ein nationales Literal darf maximal 90 Zeichen beschreiben. Zur Unterscheidung von alphanumerischen Literalen besteht bei einem nationalen Literal der öffnende Literalbegrenzer aus dem Buchstaben N, gefolgt von einem Anführungszeichen. Da ein COBOL-Quelltext in EBCDIC vorliegt, können als Inhalt des Literals nur EBCDIC-Zeichen geschrieben werden. Der Compiler ersetzt sie automatisch durch die entsprechende UTF-16-Darstellung.

Zusätzlich gibt es auch die hexadezimale Variante, eingeleitet durch die beiden Buchstaben NX, gefolgt von einem Anführungszeichen. Dabei ist jedes Zeichen direkt in seiner UTF-16-Darstellung in Form von 4 hexadezimalen Ziffern zu schreiben. Auf diese Weise lassen sich auch nationale Literale angeben, die keine Entsprechung im EBCDIC-Zeichensatz haben.

Beispiel 12-13

```
N'AB '
NX"004100420020"
```

beschreiben beide die gleiche Zeichenfolge in UTF-16-Darstellung

Eine besondere Form von Literalen stellen die figurativen Konstanten dar. Auch davon gibt es nationale Formen:

- Das Schlüsselwort ALL, gefolgt von einem nationalen Literal.
- Die als Schlüsselwörter vordefinierten SPACE, QUOTE, ZERO, HIGH-VALUE und LOW-VALUE. Abhängig vom Kontext ihrer Verwendung stehen sie für die entsprechenden alphanumerischen oder nationalen Literale.

Beispiel 12-14

```
01 alfa PIC X.
01 nat PIC N.
MOVE SPACE to alfa, nat.
```

versorgt das Feld alfa mit einem alphanumerischen Leerzeichen (X'40') und das Feld nat mit einem nationale Leerzeichen (X'0020').

12.6.4 Übertragung von nationalen Datenfeldern

Nationale Datenfelder werden mit der MOVE Anweisung übertragen. Bei Operanden unterschiedlicher Klassen kann eine implizite Konvertierung des Inhalts eines alphanumerischen Feldes in die entsprechende nationale Darstellung stattfinden. Dies ist aber nicht in umgekehrter Richtung möglich. Ein ggf. längeres nationales Empfangsfeld wird bei einer Übertragung dann mit nationalen Leerzeichen (X'0020') aufgefüllt.

Beispiel 12-15

```
01 a  PIC XXX VALUE "123".
01 s.
  02 nat1  PIC N(2).
  02 nat2  PIC N(2).
01 n  PIC N(4) JUSTIFIED RIGHT.
MOVE a TO s, n.
```

nach dem MOVE haben a, s und n folgenden Inhalt (in hexadezimaler Schreibweise):

a: F1F2F3

s: F1F2F34040404040

n: 0020003100320033'

12.6.5 Nationale Datenfelder in Bedingungen

In einer Vergleichsbedingung darf ein nationales Datenfeld mit Operanden verschiedener Klassen verglichen werden. Dabei kann ebenfalls eine implizite Konvertierung von Datenfeldinhalten in die entsprechende nationale Darstellung stattfinden.

Für den Vergleich werden nicht-nationale Operanden so behandelt, als ob sie mittels MOVE in ein „gleichgroßes “ nationales Datenfeld übertragen wurden. Anschließend erfolgt dann ein Vergleich der nationalen Daten. Der kürzere Operand wird dazu rechts mit nationalen Leerzeichen auf die Länge des längeren Operanden aufgefüllt.

Beispiel 12-16

```
Beispiel 12-16
01 num     PIC 9(4) VALUE 1860.
01 alfa     VALUE "TSV MÜNCHEN".
01 nat.
  02 nat1  PIC NNN VALUE N"TSV".
  02 nat2  PIC NNNN VALUE N"1860".
IF num = nat2 THEN ...                (1)
IF alfa = nat1 THEN ...               (2)
```

(1) Das numerische Feld num wird in ein 4 Zeichen langes nationales Vergleichsfeld (definiert als NNNN) übertragen und mit nat2 verglichen. Der Vergleich ist wahr.

(2) Das alphanumerische Feld alfa ist in ein nationales Vergleichsfeld (definiert als N(11)) konvertiert zu denken. Dieses wird verglichen mit einem Vergleichsfeld (ebenfalls als N(11) definiert, dessen Inhalt nat1 entspricht, ergänzt um 8 nationalen Leerzeichen).
Der Vergleich ist falsch.

Der Vergleich einzelner nationaler Zeichen erfolgt immer auf Basis der Binär-Darstellung dieser Zeichen. Das gilt auch für die Anweisungen SORT und MERGE. Die erweiterten Möglichkeiten beim Vergleich entsprechend Unicode werden nicht unterstützt (culturally sensitive, normalization).
Zu beachten ist auch, dass eine Vergleichsrelation, die zwischen 2 alphanumerischen Zeichen besteht, nicht genauso zwischen den entsprechenden beiden nationalen Zeichen bestehen muss (z.B. "A" < "1", aber N"A" > N"1").

12.6.6 Konvertierungen zwischen EBCDIC und UTF-16-Darstellung

Neben den impliziten Konvertierungen bei Übertragungen und Vergleichsbedingungen gibt es die Möglichkeit, explizit Konvertierungen mit Hilfe von FUNCTIONs durchzuführen:

- NATIONAL-OF liefert die nationale Darstellung (UTF-16) eines alphanumerischen Arguments
- DISPLAY-OF liefert die alphanumerische Darstellung (EBCDIC) eines nationalen Arguments

Beide Funktionen erlauben als optionales zweites Argument die Angabe eines Ersatzzeichens, das an Stelle derjenigen Zeichen eingesetzt wird, die kein Gegenstück im anderen Zeichensatz haben.

Beispiel 12-17

```
01 n  PIC N(3) VALUE NX"E23A00410040".
01 a  PIC X VALUE "?".
FUNCTION DISPLAY-OF (n, a)
```

Der Funktionsaufruf liefert ein alphanumerisches Datenfeld mit Inhalt „?A@“, weil das erste nationale Zeichen kein EBCDIC Gegenstück hat. Dafür wird im Ergebnis das Ersatzzeichen „?“ eingesetzt.

12.6.7 Fehlerbehandlung bei Konvertierungen

Lässt sich bei impliziten oder bei expliziten Konvertierungen (durch Aufruf der FUNCTIONs ohne zweiten Parameter) ein Zeichen aufgrund des fehlenden Gegenstücks im anderen Zeichensatz nicht darstellen, tritt die Ausnahmesituation EC-DATA-CONVERSION auf. In diesem Fall wird das im System festgelegte Ersatzzeichen Punkt (’.’) verwendet.

Ist die Überprüfung der Ausnahmesituation durch die zugehörige TURN-Direktive eingeschaltet, so wird der Ausnahmezustand ausgelöst und kann ggf. in einer entsprechenden USE-Prozedur behandelt werden.

Beispiel 12-18

```
01 n  PIC N VALUE NX"E23A".
...
IF  FUNCTION DISPLAY-OF(n) = "a"    (1)
THEN ...
ELSE ...                            (2)
END-IF                              (3)
```

(1) Die Ausnahmesituation EC-DATA-CONVERSION tritt auf (NX'E23A' hat kein EBCDIC Gegenstück):

- wenn die Überprüfung der Ausnahmesituation durch >>TURN EC-DATA-CONVERSION CHECKING ON eingeschaltet ist und eine USE-Prozedur existiert, die mit RESUME AT NEXT STATEMENT verlassen wird, wird nach (3) fortgesetzt
- ist die Überprüfung nicht eingeschaltet oder existiert keine entsprechende USE-Prozedur, wird bei (2) fortgesetzt, weil das verwendete Ersatzzeichen Punkt (".") ungleich "a" ist.

12.7 Objektorientierte Konzepte

In diesem Kapitel werden folgende Themen behandelt:

- Grundbegriffe des objektorientierten Programmierens
- Parametrisierte Klassen und Interfaces
- Dateien in Objekten
- Konformität
  - Konformität zwischen Schnittstellen
  - Konformität von Parametern und Rückgabe-Elementen
  - Parameter
  - Rückgabe-Elemente
- Die Systemklasse BASE
  - Methode NEW
  - Methode FactoryObject
- Automatische Speicherfreigabe (Garbage Collection)

12.7.1 Grundbegriffe des objektorientierten Programmierens

Objektorientierte Software-Entwicklung stützt sich auf einige Schlüsseltechniken, die es ermöglichen, Softwaresysteme verstehbar, änderbar und wiederverwendbar zu gestalten.
Die wesentlichsten sind:

- Definieren von Objekten, die Botschaften austauschen
- Bilden von möglichst allgemein verwendbaren Klassen
- Nutzen der Vererbung als Strukturierungsmittel und zum Vermeiden von Mehrfachimplementierungen
- Verwenden von so genannten virtuellen Methoden, wie Polymorphie und „late binding“, zur konkreten Spezialisierung.

Ein klassisch programmiertes Anwendungssystem besteht grundsätzlich aus einem Algorithmus und einer Datenstruktur. Bei einem objektorientierten Anwendungssystem sind Daten nur mehr indirekt durch Funktionen modifizierbar.
Im Folgenden werden Grundideen des objektorientierten Progammierens erläutert.

Objekte

Ein Objekt wird durch eine Klasse definiert. Es besteht aus einer Kombination von Daten und Funktionen. Diese Funktionen heißen Methoden des Objektes. Objekte kommunizieren untereinander durch den Austausch von Botschaften. Botschaften werden durch das Aufrufen von Methoden gesendet.
Jedes Objekt bekommt bei seiner Erzeugung eine Objektreferenz zugewiesen, durch die es für seine gesamte Lebensdauer eindeutig identifizierbar ist.

Klassen

Eine Klasse beschreibt durch die Klassendefinition eine Menge von Objekten mit ihren Attributen und Methoden.

Objektreferenzen

Eine Objektreferenz ist ein implizit oder explizit definiertes Datenelement. Der Inhalt der Objektreferenz verweist eindeutig auf ein Objekt und seine zugehörigen Informationen.Implizit definierte Objektreferenzen sind die vordefinierten Objektreferenzen und Objektreferenzen, die von einer Objektsicht (object-view) zurückgegeben werden. Explizit definierte Objektreferenzen sind Datenelemente, die durch eine Datenerklärung definiert werden, die eine USAGE OBJECT REFERENCE-Klausel spezifiziert.

Vordefinierte Objektreferenzen

Eine vordefinierte Objektreferenz ist ein implizit erzeugtes Datenelement, auf welches einer der Bezeichner NULL, SELF oder SUPER verweist.

Objektinstanzen

Soll ein neues Objekt erzeugt werden, legt das System eine neue Kopie der Attribute der Klasse an und erzeugt damit eine Objektinstanz. Die Methoden dieser Klasse gelten dann auch für dieses neue Objekt.

Fabrikobjekte (Factory-Objects)

In OO-COBOL enthält jede Klasse ein und nur ein spezielles Objekt, das so genannte Fabrikobjekt. Es wird durch die FACTORY-Definition der Klasse festgelegt. Das Fabrikobjekt ist verantwortlich für das Erzeugen von Objektinstanzen (Objekte) einer Klasse.

Methoden

Methoden sind Operationen, Funktionen und Anweisungen, die ein Objekt ausführen kann.

Vererbung

Die zu Grunde liegende Idee besteht darin, eine Klassenhierarchie, geordnet vom „Allgemeinen“ zum „Speziellen“, so zu nutzen, dass die „unten“ angesiedelten Klassen von den darüberliegenden Klassen, deren Methoden und Attribute erben können. Das Konzept der Vererbung verhindert eine Vervielfachung des Codes für gleiche Anwendungen.Eine Klasse kann von mehreren anderen Klassen erben. Dieser Vorgang heißt **Mehrfachvererbung.**

Polymorphismus

Polymorphismus bedeutet, dass die gleiche Botschaft, die an verschiedene Objekte geschickt wird, abhängig vom Typ des Objektes, verschiedene Aktionen auslöst d.h. unterschiedliche Methoden aufrufen kann, sofern deren Schnittstelle identisch ist.

Schnittstellen

Jedes Objekt besitzt eine Schnittstelle, bestehend aus dem Namen und den Parameterspezifikationen für jede Methode eines Objektes, einschließlich der geerbten Methoden.Jede Klasse verfügt über zwei Schnittstellen: eine für das Fabrikobjekt und eine Schnittstelle für die anderen Objekte.

Interface

OO-COBOL stellt das Sprachmittel Interface bereit, um eine Schnittstelle - losgelöst von einem konkreten Objekt bzw. einer konkreten Klasse - zu beschreiben. Dabei ist es nicht erforderlich, die komplette Schnittstelle eines Objekts in einem Interface festzuhalten; vielmehr können mit Hilfe eines Interface, das nur den gemeinsamen Teil der Schnittstellen von nicht verwandten Klassen beschreibt, Objekte dieser verschiedenen Klassen einheitlich verarbeitet werden.

Konformität

Konformität ist eine nicht umkehrbare Relation zwischen zwei Schnittstellen und zwischen einem Objekt und einer Schnittstelle.
Konformität ist eine der Grundlagen für die Anwendung von wesentlichen Merkmalen, wie z.B Vererbung und Interfacedefinitionen.
Besteht Konformität zwischen zwei Klassen, so können alle Schnittstellen der einen Klasse auch in der anderen verwendet werden.
Konformität ist an bestimmte Regeln gebunden, die grundsätzlich zur Übersetzungszeit abgeprüft werden. Wird allerdings eine Objektsicht (object-view) verwendet bzw. eine Methode für eine universelle Objektreferenz aufgerufen, so findet diese Überprüfung erst zur Laufzeit statt.

Schematische Beispiele

Beispiel 12-19

für Polymorphismus, late binding

```
Klasse A                        {M(A)} Menge der verfügbaren Methoden
  Methode M
     Display ’AAA’

Klasse B inherits A             {M(B)} Menge der verfügbaren Methoden
  Methode M override
     Display ’BBB’
```

Nutzerprogramm

```
01 aref usage object reference A      a)
```

```
  invoke B 'NEW' returning aref       b)
  invoke aref 'M' c)

       Ausgabe -> BBB

  invoke A 'NEW' returning aref d)
  invoke aref 'M''

       Ausgabe -> AAA
```

Anmerkungen

1. Eine Objektreferenz, die auf Objekte der Klasse A und aller Klassen, die davon erben, zeigen kann.
2. aref zeigt auf ein Klasse B Objekt.
3. Die Methode, die wirklich aufgerufen wird, ist nicht notwendig eine der Methoden, die in der Klasse verfügbar sind, die bei der Definition der Objektrefrenz angegeben ist, denn das wäre Klasse A und damit die Methode $M_{(A)}$, sondern es kommt auf den Inhalt der Objektreferenz zum Zeitpunkt des Methodenaufrufs an: die Methode ist eine, die in der Klasse des aktuellen Objekts verfügbar ist, entweder in der Klasse selbst definiert oder von einer Oberklasse geerbt.
   In diesem Falle ist das $M_{(B)}$.
4. Hier zeigt aref jetzt auf ein Klasse A Objekt und ein „äußerlich" identischer Invoke wie bei c) ergibt eine andere Ausgabe, weil das aktuelle Objekt ein anderes ist.

Nützliche Hinweise

- Hieraus leitet sich der Begriff „late binding" ab, denn die Zuordnung der aktuellen Methode zum Methodenaufruf kann erst zur Laufzeit geschehen, nämlich dann, wenn der aktuelle Inhalt der Objektreferenz, d.h. die Klasse in der die Methode zu suchen ist, bestimmt wird.
- Dieser Effekt tritt nur dann auf, wenn in einer erbenden Klasse eine Methode der Oberklasse überschrieben wird. Andernfalls ist die gerufene Methode die aus der Oberklasse.
- Wesentlich dabei ist, dass neben den identischen Methodennamen auch die Schnittstelle der entsprechenden Methoden gleich sein muss (damit ein Unterklassenobjekt auch von Seiten der Schnittstelle wie ein Objekt seiner Oberklasse angesprochen werden kann). Hieraus ergibt sich dann auch die Konformität und eine Form des Polymorphismus.

Beispiel 12-20

für SELF

```
Klasse A                {M(A),N(A)} Menge der verfügbaren Methoden
  Methode M
     Display 'AAA'
  Methode N
     invoke SELF 'M'                  c)

Klasse B inherits A     {M(B),N(A)} Menge der verfügbaren Methoden
  Methode M override
     Display 'BBB'
```

Nutzerprogramm

```
01 aref usage object reference A

  invoke B 'NEW' returning aref      a)
```

```
  invoke aref 'N'                    b)

       Ausgabe -> BBB
```

Anmerkungen

1. Das aktuelle Objekt ist ein Klasse B Objekt.
2. Hier wird die Methode $N_{(A)}$ gerufen, weil in der erbenden Klasse B (von der das aktuelle Objekt stammt) kein eigenes lokales N definiert wurde.
3. Die Ausführung von N führt zu einem weiteren Methodenaufruf. SELF steht hier für das Objekt, mit dem die Methode aufgerufen wurde, d.h. ein Objekt der Klasse B.Damit ergibt sich wieder die im vorigen Beispiel dargestellte Situation:
   Die für ein Objekt der Klasse B zutreffende Methode ist diejenige, die in Klasse B verfügbar ist, d.h. aus der Menge $\{M_{(B)}, N_{(A)}\}$ (also nicht notwendigerweise eine Methode, die in der Klasse verfügbar ist, in der auch der invoke mit SELF steht: das wäre aus der Menge $\{M_{(A)}, N_{(A)}\}$).

Nützliche Hinweise

- Hier stellt sich wieder eine Form des „late binding“ dar.
- Erbt 'niemand' von Klasse A, dann ist die mit SELF gerufene Methode genau die, die in Klasse A auch verfügbar ist.
- Methoden können durch Verwendung von SELF andere Methoden für ihr „aktuelles Objekt“ aufrufen, die zur Übersetzungszeit eventuell noch nicht existieren. Diese Möglichkeit ist hilfreich, da zum Übersetzungszeitpunkt nicht immer absehbar ist „wer“ die entsprechende Klasse in Zukunft erben und ggf. Methoden davon durch eigene ersetzen wird.

Beispiel 12-21

für SUPER

```
Klasse A                 {M(A)} Menge der verfügbaren Methoden
  Methode M
     Display 'AAA'

Klasse B inherits A      {M(B),N(B)} Menge der verfügbaren Methoden
  Methode M override
     Display 'BBB'
  Methode N
     invoke SUPER 'M'              c)

Klasse C inherits B      {M(C),N(B)} Menge der verfügbaren Methoden
  Methode M override
     Display 'CCC'
```

Nutzerprogramm

```
01 aref usage object reference A

  invoke C 'NEW' returning aref      a)
  invoke aref AS C 'N'               b)

       Ausgabe -> AAA
```

Anmerkungen

1. Das aktuelle Objekt ist ein C Objekt.
2. Die Methode ist in der Menge der für C-Objekte verfügbaren Methoden zu suchen, d.h. aus $\{M_{(C)},N_{(B)}\}$. Es wird also die in Klasse B definierte Methode gerufen.
3. Dies führt zu einem internen Methodenaufruf für das gleiche Objekt, für das Methode $N_{(B)}$ gerufen wurde (also weiterhin das Klasse C Objekt). SUPER zeigt nun an, dass jetzt die Methode unter den in der Oberklasse (falls mehrere geerbte Klassen, dann die betreffende mit dem Klassennamen qualifiziert festlegen!) verfügbaren Methoden gesucht wird. Diese Oberklasse ist nicht relativ zum aktuellen Objekt zu sehen, sondern bezieht sich statisch auf die Klasse in der der Methodenaufruf mit SUPER steht, d.h. Klasse A. Methode M bei c) ist also in den für Klasse A verfügbaren Methoden zu suchen, d.h. aus der Menge $\{M_{(A)}\}$.

Nützliche Hinweise

- SUPER bietet die Möglichkeit, die Methodenauswahl zu steuern: statt der für das aktuelle Objekt verfügbaren Methoden, wird die Suche auf die statisch bekannte Vererbungshierarchie beschränkt.
- Durch das Überschreiben einer Methode und die Verwendung von SUPER schafft man die Möglichkeit, mit einer „neuen“ Methode bei gleichbleibender Schnittstelle zusätzliche Operationen ausführen zu können. Da in jeder Klasse die verfügbaren Methoden eindeutige Namen haben müssen, und die „neue“ Methode die ersetzte unsichtbar werden lässt, braucht man eine Möglichkeit, diese überschriebene Methode der Oberklasse nutzen zu können. Dazu verwendet man das Sprachmittel SUPER.

12.7.2 Parametrisierte Klassen und Interfaces

Eine parametrisierte Klasse ist eine Klasse mit einem oder mehreren formalen Parametern. Von einem formalen Parameter ist lediglich bekannt, ob es sich um eine Klasse oder ein Interface handelt. Darüber hinausgehende konkrete Eigenschaften der formalen Parameter bleiben offen.

Eine parametrisierte Klasse wird genutzt, indem im REPOSITORY Paragraf des Nutzerprogramms eine Klasse K als Expansion der parametrisierten Klasse P mit aktuellen Parametern spezifiziert wird: die formalen Parameter sind dann durch die Namen von konkreten Klassen bzw. Interfaces (die aktuellen Parameter) zu ersetzen, ebenso wie der Name der parametrisierten Klasse durch den der konkreten Expansion. Dadurch entsteht zu diesem Zeitpunkt aus P die neue konkrete Klasse K, die sich wie eine nicht-parametrisierte Klasse verhält und auch so behandelt wird.

Die eigentliche Übersetzung dieser neuen Klasse K mit den ersetzten Klassen- bzw. Interfacenamen erfolgt automatisch im Anschluss an die Übersetzung des Nutzers (siehe handbuch „COBOL2000 Benutzerhandbuch“ [1]).
In diesem Zusammenhang werden folgende Formulierungen gebraucht:

- Die konkrete Klasse K expandiert die parametrisierte Klasse P.
- Die parametrisierte Klasse P wird expandiert als konkrete Klasse K.

Der Vorgang der Parameterersetzung, der Übersetzung und die dabei entstehende Klasse wird als „Expansion der parametrisierten Klasse P“ bezeichnet.

Jede solche Expansion einer parametrisieren Klasse erzeugt eine eigenständige konkrete Klasse. Sie hat ihr eigenes Fabrikobjekt und ist unabhängig von anderen Expansionen der gleichen parametrisierten Klasse. Innerhalb einer Ablaufeinheit müssen jedoch die Namen aller angesprochenen Klassen eindeutig sein: wird eine parametrisierte Klasse P in mehreren Programmen der Ablaufeinheit mit dem gleichen Namen K expandiert, sollten auch bei diesen Expansionen die aktuellen Parameter die gleichen sein.

Der Vorteil von parametrisierten Klassen gegenüber solchen, die die gleiche Leistung an Stelle von Parametrisierung mittels universeller Objektreferenzen erbringen, liegt darin, dass der Compiler bereits statisch ein einziges mal zur Übersetzungszeit Prüfungen vornehmen kann, die im anderen Fall erst dynamisch wiederholt zur Ablaufzeit erfolgen.

Die obigen Beschreibungen für parametrisierte Klassen gelten für parametrisierte Interfaces analog.

Beispiel 12-22

Folgendes Beispiel zeigt eine „first-in-first-out“-Liste zur Aufnahme von Objektreferenzen. Deren Klasse soll noch nicht weiter festgelegt sein und ist daher als Parameter definiert. Eine solche Liste nimmt beliebig viele Elemente auf, wobei das Element, welches zuerst in die Liste eingetragen wurde, beim Lesen auch als erstes wieder zurückgegeben und gleichzeitig aus der Liste entfernt wird.

Realisiert wird diese Liste mit Hilfe der beiden parametrisierten Klassen FIFO-LISTE und LIST-ELEMENT1. Die Klasse FIFO-LISTE beschreibt die Verwaltungsstruktur für die gesamte Liste. Als Parameter hat sie die Klasse der aufzunehmenden Objekte, sowie zusätzlich die Klasse der entsprechenden konkreten Expansion von LIST-ELEMENT1.

Die parametrisierte Klasse LIST-ELEMENT1 beschreibt ein einzelnes Element der Liste. Als Parameter hat sie die Klasse der Objekte, deren Referenz in so einem Listenelement gespeichert wird. LIST-ELEMENT1 fungiert als „Hilfsklasse“, die nach außen hin gar nicht gesondert auftritt. Sie wird nur innerhalb der FIFO-LISTE genutzt und sollte dort automatisch mit passenden Parametern erzeugt werden.

Das Verschachteln von parametrisierten Klassen ist jedoch vom COBOL-Standard 2002 nicht erlaubt. Der Anwender muss daher die nötigen Expansionen selber programmieren.

```
        >>IMP Compiler-Action UPDATE-REPOSITORY
```

```
ON                          a)
       IDENTIFICATION DIVISION.
>>>    CLASS-ID.                LIST-ELEMENT1 INHERITS BASE USING
PAR.     b)
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL NAMES.
           TERMINAL IS TTT.
       REPOSITORY.
           CLASS
PAR,                                        b)
           CLASS                BASE.

FFF    FACTORY.
       PROCEDURE DIVISION.
+++    METHOD-ID.               MAKE-
NEW.                                  c)
       DATA DIVISION.
       LINKAGE SECTION.
       01 P USAGE OBJECT REFERENCE PAR.
       01 R USAGE OBJECT REFERENCE ACTIVE-CLASS.
       PROCEDURE DIVISION USING BY VALUE P RETURNING R.
       1.
           INVOKE SELF "NEW" RETURNING R.
           INVOKE R "SET-CONTENT" USING P.
           EXIT METHOD.
       END METHOD             MAKE-NEW.
       END FACTORY.

OOO    OBJECT.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 NXT  USAGE OBJECT REFERENCE LIST-
ELEMENT1.                       d)
       01 CONT USAGE OBJECT REFERENCE
PAR.                                  e)
       PROCEDURE DIVISION.

+++    METHOD-ID.               SET-
NEXT.                                  f)
       DATA DIVISION.
       LINKAGE SECTION.
       01 P USAGE OBJECT REFERENCE LIST-ELEMENT1.
       PROCEDURE DIVISION USING P.
       1.
           SET NXT TO P.
           EXIT METHOD.
       END METHOD               SET-NEXT.

+++    METHOD-ID.               GET-NEXT.
       DATA DIVISION.
       LINKAGE SECTION.
       01 R USAGE OBJECT REFERENCE LIST-ELEMENT1.
       PROCEDURE DIVISION RETURNING R.
       1.
           SET R TO NXT.
           EXIT METHOD.
```

```
        END METHOD               GET-NEXT.

+++     METHOD-ID.               GET-CONTENT.
        DATA DIVISION.
        LINKAGE SECTION.
        01 P USAGE OBJECT REFERENCE PAR.
        PROCEDURE DIVISION RETURNING P.
        1.
            SET P TO CONT.
            EXIT METHOD.
        END METHOD               GET-CONTENT.

+++     METHOD-ID.               SET-CONTENT.
        DATA DIVISION.
        LINKAGE SECTION.
        01 P USAGE OBJECT REFERENCE PAR.
        PROCEDURE DIVISION USING BY VALUE P.
        1.
            SET CONT TO P.
            EXIT METHOD.
        END METHOD               SET-CONTENT.
        END OBJECT.
        END CLASS                LIST-ELEMENT1.
```

Anmerkungen

a) Die Expansionen dieser Klasse gehen in die Expansionen der parametrisierten Klasse FIFO-LIST als Parameter ein. Daher müssen die Repository-Daten der jeweiligen Expansion von LIST-ELEMENT1 zur Verfügung stehen. Dies wird durch die Angabe der Direktive sichergestellt (für die korrekte Zuweisung eines Repository hat jedoch der Anwender zu sorgen, siehe Handbuch „COBOL2000 Benutzerhandbuch “ [1]).

b) Parameter PAR ist der Name der Klasse, deren Objektreferenzen in der Liste gespeichert werden sollen.

c) Die Methode MAKE-NEW erzeugt ein neues Listenelement und versorgt auch die darin zu speichernde Objektreferenz. In die Liste eingekettet wird das Listenelement-Objekt jedoch noch nicht - das ist Aufgabe der Listenklasse FIFO-LIST.

d) Die Listenelemente sind einfach „vorwärts“ über eine Objektreferenz auf das nächst folgende Element in der Liste verkettet.

e) Der Nutzinhalt eines Listenelements besteht aus einer Objektreferenz für Objekte der Klasse, die später als aktueller Parameter angegeben wird oder davon erbender Klassen.

f) Die anderen 4 Methoden stellen die elementaren Zugriffe zum Schreiben bzw. Lesen der beiden Daten eines Listenelements dar.

Die eigentlichen Listenzugriffe, d.h. die Methoden APPEND-ENTRY und REMOVE-ENTRY werden von FIFO-LIST bereitgestellt. Als zusätzlicher Informationsdienst fungiert die Methode LIST-LENGTH.

```
        IDENTIFICATION DIVISION.
>>>>    CLASS-ID.                FIFO-LIST INHERITS BASE
                                           USING PAR-OBJ, PAR-
ELEM.        a)
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SPECIAL NAMES.
            TERMINAL IS TTT.
        REPOSITORY.
            CLASS                PAR-OBJ,
```

```
            CLASS                PAR-ELEM,
            CLASS                BASE.

OOO     OBJECT.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 ELEM-COUNT PIC S9(8) COMP-5 VALUE
0.                             b)
        01 FST        USAGE OBJECT REFERENCE PAR-
ELEM.                       c)
        01 LST        USAGE OBJECT REFERENCE PAR-
ELEM.                       c)
        PROCEDURE DIVISION.

+++     METHOD-ID.                APPEND-
ENTRY.                             d)
        DATA DIVISION.
        LOCAL-STORAGE SECTION.
        01 W USAGE OBJECT REFERENCE PAR-ELEM.
        LINKAGE SECTION.
        01 P USAGE OBJECT REFERENCE PAR-OBJ.
        PROCEDURE DIVISION USING P.
        1.
            INVOKE PAR-ELEM "MAKE-NEW" USING P RETURNING W.
            IF LST = NULL
            THEN                        *> MAKE VERY FIRST ENTRY IN LIST
              SET FST, LST TO W
              MOVE 1 TO ELEM-COUNT
            ELSE                        *> APPEND ENTRY TO THE END OF
LIST
              INVOKE LST "SET-NEXT" USING W
              SET LST TO W
              ADD 1 TO ELEM-COUNT
            END IF.
            EXIT METHOD.
        END METHOD                APPEND-ENTRY.

+++     METHOD-ID.                REMOVE-
ENTRY.                            e)
        DATA DIVISION.
        LOCAL-STORAGE SECTION.
        01 W USAGE OBJECT REFERENCE PAR-ELEM.
        LINKAGE SECTION.
        01 R USAGE OBJECT REFERENCE PAR-OBJ.
        PROCEDURE DIVISION RETURNING R.
        1.
            IF FST = NULL
            THEN                        *> LIST IS EMPTY
              SET R TO NULL
            ELSE                        *> DELETE 1st LIST-ELEM,MAKE
SURE
                                        *> THAT IT WILL BE GARBAGE
COLLECTED
              INVOKE FST "GET-CONTENT" RETURNING R
              SUBTRACT 1 FROM ELEM-COUNT
              SET W TO FST
              INVOKE W "GET-NEXT" RETURNING FST
```

```
                INVOKE W "SET-NEXT" USING NULL
            END IF.
            EXIT METHOD.
        END METHOD                REMOVE-ENTRY.

+++     METHOD-ID.                LIST-LENGTH.
        DATA DIVISION.
        LINKAGE SECTION.
        01 R PIC S9(8) COMP-5.
        PROCEDURE DIVISION RETURNING R.
        1.
            MOVE ELEM-COUNT TO R.
            EXIT METHOD.
        END METHOD                LIST-LENGTH.
        END OBJECT.
        END CLASS                 FIFO-LIST.
```

Anmerkungen

a) Parameter PAR-OBJ ist der Name der Klasse, deren Objektreferenzen (auch die von Unterklassen) in der Liste gespeichert werden sollen.
Parameter PAR-ELEM ist der Name der dazu passenden expandierten Hilfsklasse LIST-ELEMENT1.

b) Dieser Zähler dient lediglich dem zusätzlichen Informationsdienst LIST-LENGTH. Für die Listenzugriffe und die Leistungen ist er nicht notwendig.

c) Zur Verwaltung der FIFO-Liste ist es ausreichend, das erste (FST) und das letzte (LST) Element der Liste auffinden zu können.

d) Die Methode APPEND-ENTRY erzeugt selbst ein neues Listenelement mit der als Parameter übergebenen Objektreferenz als Inhalt. Dieses neue Element wird als letztes in die Liste eingekettet und die Verwaltungsdaten werden entsprechend angepasst.

e) Die Methode REMOVE-ENTRY entfernt das erste Element aus der Liste und gibt die dort gespeicherte Objektreferenz zurück. Hierfür wäre es ausreichend, nur die Verwaltungsdaten anzupassen - um jedoch die garbage collection zu erleichtern, wird die Verkettung zur Liste im entfernten Listenelement auf NULL gesetzt.

Der folgende Programmausschnitt zeigt exemplarisch eine mögliche Anwendung dieser beiden parametrisierten Klassen:

Klasse A und B sind beliebige konkrete Klassen; FIFOB ist eine konkrete Klasse für eine „first-in-first-out“-Liste von Objektreferenzen für Objekte der Klasse B (und deren Unterklassen), FIFOA1 und FIFOA2 sind konkrete Klassen für „first-in-first-out“-Listen von Objektreferenzen der Klasse A (und deren Unterklassen).

```
        IDENTIFICATION DIVISION.
>>>>    PROGRAM-ID.                N.
        ...
        REPOSITORY.
             CLASS              A,
             CLASS              B,
             CLASS              FIFOA1 EXPANDS FIFO-LIST USING A
ELEMA        a)
             CLASS              FIFOA2 EXPANDS FIFO-LIST USING A
ELEMA        a)
             CLASS              FIFOB EXPANDS FIFO-LIST USING B
ELEMB         b)
             CLASS              ELEMA EXPANDS LIST-ELEMENT1 USING
A           c)
             CLASS              ELEMB EXPANDS LIST-ELEMENT1 USING
B        b) c)
             CLASS              LIST-ELEMENT1
             CLASS              FIFO-LIST
        ...
        WORKING-STORAGE SECTION.
        01 FLA1       USAGE OBJECT REFERENCE FIFOA1.
```

```
         01 FLA2       USAGE OBJECT REFERENCE FIFOA2.
         01 FLA3       USAGE OBJECT REFERENCE FIFOA2.
         01 FLB        USAGE OBJECT REFERENCE FIFOB.
         01 OB         USAGE OBJECT REFERENCE B.
         01 W          PIC X(10).
         ...
             INVOKE FIFOA1 "NEW" RETURNING FLA1.
             INVOKE FIFOA2 "NEW" RETURNING
FLA2.                                 d)
             INVOKE FIFOA2 "NEW" RETURNING
FLA3.                                 d)
             INVOKE FIFOB  "NEW" RETURNING FLAB.
         ...
             INVOKE FLB "APPEND-ENTRY" USING OB. ...
             INVOKE FLB "REMOVE-ENTRY" RETURNING OB.
             IF OB =
NULL                                                    e)
               DISPLAY " ---> LISTE LEER"
             ELSE
               ...
```

Anmerkungen

a) Obwohl bei FIFOA1 und FIFOA2 die parametrisierte Klasse und die aktuellen Parameter identisch sind, entstehen aufgrund der unterschiedlichen Namen bei den beiden Expansionen zwei verschiedene konkrete Klassen.

b) Es ist erlaubt, die Expansion einer parametrisierten Klasse (hier ELEMB) als aktuellen Parameter für eine andere Expansion (hier FIFOB) zu verwenden, solange dadurch keine zyklischen Abhängigkeiten entstehen. Es ist jedoch verboten, eine parametrisierte Klasse selbst, z.B. LIST-ELEMENT1, als aktuellen Parameter zu verwenden.
Der Compiler führt nachfolgende Expansionen nicht zwingend in der Reihenfolge aus, wie sie im Programm geschrieben wurden, sondern so, dass notwendige Daten zu aktuellen Parametern rechtzeitig vor der Expansion verfügbar sind.

c) Die Expansion der beiden Hilfsklassen ELEMA und ELEMB ist wegen den oben erwähnten Restriktionen des COBOL-Standards vom Anwender anzugeben. Diese Klassen zu verwenden, Objekte davon zu erzeugen usw. sollte allein den Expansionen von FIFO-LIST vorbehalten bleiben!

d) Von der Expansion einer parametrisierten Klasse können mehrere konkrete Objekte erzeugt werden. Mittels der Objektreferenzen FLA2 und FLA3 stehen dem Programm hier 2 FIFO-Listen für Objekte der Klasse A zur Verfügung. Dieses Verfahren leistet das gleiche wie die in Anmerkung a) beschriebene Lösung - mittels 2 verschiedenen, aber leistungsmäßig identischen Klassen. Es ist aber einfacher und daher auf jeden Fall vorzuziehen.

e) Ist die Liste leer, gibt die Methode REMOVE-ENTRY als Objektreferenz NULL zurück.

12.7.3 Dateien in Objekten

Eine Datei kann in einem Objekt definiert werden, indem man sie im FACTORY- bzw. OBJECT-Teil in einem FILE-CONTROL-Paragrafen und einer FILE SECTION beschreibt und in einer oder mehreren Methoden entsprechende Anweisungen zu ihrer Verarbeitung wie OPEN, CLOSE, READ, WRITE angibt.

Hat eine Anwendung mehrere Objektinstanzen von einer Klasse erzeugt, in der eine Datei definiert ist, dann ist diese Anwendung selbst dafür verantwortlich, dass es beim Zugriff auf die Datei zu keinen Konflikten kommt. Das kann folgendermaßen erreicht werden:

1. Die Datei hat die Eigenschaft EXTERNAL. Somit gibt es nur ein einziges Exemplar der Datei und alle Objekte arbeiten mit derselben Datei. Wegen der EXTERNAL-Eigenschaft können dann noch weitere Programme gleichzeitig auf die Datei zugreifen.
2. Die ASSIGN-Klausel der Datei verwendet die Variante „ASSIGN to datenname“. Gegebenenfalls kann dann zur Laufzeit jedes Objekt der Klasse beim OPEN ein eigenes Exemplar einer solchen Datei auswählen, indem es datenname mit einem objektspezifischen Wert versorgt. Auf diese Weise lassen sich Konflikte des Objektes mit den Dateien in den weiteren Objektinstanzen seiner Klasse vermeiden.
3. Wird Variante 2 nicht genutzt, dann arbeiten verschiedene Objektinstanzen auf derselben Datei. Die Anwendung koordiniert die Zugriffe auf die Datei so, dass beispielsweise zu jedem Zeitpunkt nur ein einziges Objekt (von mehreren derselben Klasse) die Datei geöffnet hat und bearbeitet. Eine andere Möglichkeit besteht darin, die Datei simultan zu bearbeiten (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]). In der Regel ist es daher in diesem Fall am sinnvollsten, nur ein einziges Objekt der Klasse zu erzeugen.

Vererbung gilt nicht nur für Daten der WORKING-STORAGE SECTION, sondern auch für die in OBJECT bzw. FACTORY definierten Dateien. Zu beachten ist daher, dass auch das Erben von einer Klasse, die eine Dateidefinition enthält, zu den oben erwähnten Konflikten führen kann, selbst wenn von jeder Klasse nur ein einziges Objekt instanziiert worden ist.

Im Unterschied zu den oben beschriebenen Dateien in Objekten verhalten sich *in Methoden definierte* Dateien analog zu Dateidefinitionen in herkömmlichen Programmen: Durch Instanziierung neuer Objekte oder Ableitung (Vererbung) von Klassen entstehen keine weiteren Exemplare der Dateidefinition.

Beispiel 12-23

Die folgenden Klasse gewährleistet den Zugriff auf sequenzielle Dateien mit Sätzen, die jeweils aus 80 Zeichen bestehen.

```
>>>>   CLASS-ID.                cseqf80 INHERITS BASE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       REPOSITORY.
           CLASS                BASE.
       OBJECT.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT seq ASSIGN TO link-name
                        FILE STATUS file-status.
       DATA DIVISION.
       FILE SECTION.
       FD seq                   RECORD CONTAINS 80.
       01 recs                  PIC X(80).
       WORKING-STORAGE SECTION.
       01 file-status PIC XX.
       01 link-namePIC X(8).
```

```
        PROCEDURE DIVISION.

+++     METHOD-ID.               F-OPEN.
        DATA DIVISION.
        LINKAGE SECTION.
        01 open-mode             PIC X.
          88 open-input          VALUE "I".
          88 open-output         VALUE "O".
        01 file-link           PIC X(8).
        PROCEDURE DIVISION USING open-mode, file-link.
        DECLARATIVES.
        erst-mal SECTION.
            USE AFTER ERROR PROCEDURE ON seq.
        open-error.
            IF file-status = ....
        END DECLARATIVES.
        1 SECTION.
        A.
            MOVE file-link TO link-name
            IF open-output
              OPEN OUTPUT seq
            ELSE
              OPEN INPUT seq
            END-IF
            EXIT METHOD.
        END METHOD              F-OPEN.

+++     METHOD-ID.              F-CLOSE.
           ....                 *> ggf DECLARATIVES
        1 SECTION.
        A.
            CLOSE seq.
            EXIT METHOD.
        END METHOD              F-CLOSE.
+++     METHOD-ID.              F-READ.
        DATA DIVISION.
        LINKAGE SECTION.
        01 rec                  PIC X(80).
        01 eof-ind              PIC X.
          88 eof                VALUE "Y" WHEN FALSE "N".
        PROCEDURE DIVISION USING rec RETURNING eof-ind.
             ....               *> ggf DECLARATIVES
        1 SECTION.
        A.
          SET eof TO FALSE
            READ seq INTO rec
              AT END SET eof TO TRUE
            END-READ
            EXIT METHOD.
        END METHOD              F-READ.

+++     METHOD-ID.              F-WRITE.
        DATA DIVISION.
        LINKAGE SECTION.
        01 rec                  PIC X(80).
        PROCEDURE DIVISION USING rec.
             ....               *> ggf DECLARATIVES
```

```
       1 SECTION.
       A.
           WRITE recs FROM rec.
           EXIT METHOD.
       END METHOD              F-WRITE.
       END OBJECT.
       END CLASS               cseqf80.
```

Diese Klasse kann z.B. verwendet werden, um eine solche Datei zu verarbeiten:

```
    ....
01 obj1 USAGE OBJECT REFERENCE cseqf80.
01 obj2 USAGE OBJECT REFERENCE cseqf80.
01 xrec PIC X(80).
01 xind PIX X VALUE "N".
  88 xeof VALUE "Y".
    ....
INVOKE cseqf80 "NEW" RETURNING obj1
INVOKE cseqf80 "NEW" RETURNING obj2
INVOKE obj1 "F-OPEN" USING "I", "EIN-DAT"
INVOKE obj2 "F-OPEN" USING "O", "AUS-DAT"
INVOKE obj1 "F-READ" USING xrec RETURNING xind
PERFORM UNTIL xeof
    ....               *> Satz verarbeiten
INVOKE obj2 "F-WRITE" USING xrec
INVOKE obj1 "F-READ" USING xrec RETURNING xind
END-PERFORM
INVOKE obj1 "F-CLOSE"
INVOKE obj2 "F-CLOSE"
    ....
```

12.7.4 Konformität

Konformität ist an bestimmte Regeln gebunden, die nachfolgend näher erläutert werden.

Der Begriff Konformität wird für zwei Sachverhalte verwendet:

1. Konformität zwischen Schnittstellen, die entweder explizit mit dem Sprachmittel Interface definiert sind oder implizit durch die Schnittstelle des Fabrik- oder Objektteils einer Klassendefinition gegeben sind.
   In Erweiterung dieser Aussage spricht man auch von Konformität von Klassen, wenn deren Fabrik- und Objekt-Schnittstellen jeweils konform zueinander sind.
2. Konformität zwischen den aktuellen Parametern und den formalen Parametern eines Unterprogramm- bzw. Methodenaufrufs.

12.7.4.1 Konformität zwischen Schnittstellen

Eine Schnittstelle interface-1 ist nur dann zu einer Schnittstelle interface-2 konform, wenn Folgendes gilt:

1. Es gibt für jede Methode bei interface-2 eine Methode in interface-1 mit demselben Namen, wobei dieselbe Anzahl von Parametern erforderlich ist mit übereinstimmenden Spezifikationen für BY REFERENCE, BY VALUE und OPTIONAL.
2. Ist der formale Parameter einer Methode in interface-2 eine Objektreferenz, dann muss der entsprechende Parameter in interface-1 auch eine Objektreferenz sein, die folgenden Regeln entspricht:
   a. Ist der Parameter in interface-2 eine universelle Objektreferenz, muss der entsprechende Parameter in interface-1 ebenfalls eine universelle Objektreferenz sein.
   b. Ist der Parameter in interface-2 mit einem Interfacenamen beschrieben, muss der Parameter in interface-1 denselben Interfacenamen haben.
   c. Ist der Parameter in interface-2 mit einem Klassennamen beschrieben, dann muss der entsprechende Parameter in interface-1 denselben Klassennamen haben. Die Angaben FACTORY und ONLY müssen in beiden Schnittstellen analog vorhanden oder nicht vorhanden sein.
   d. Ist der Parameter in interface-2 mit ACTIVE-CLASS beschrieben, dann muss der entsprechende Parameter in interface-1 ebenfalls mit ACTIVE-CLASS beschrieben sein. Die Angabe FACTORY muss in beiden Schnittstellen analog vorhanden oder nicht vorhanden sein.
3. Ist der formale Parameter einer Methode bei interface-2 keine Objektreferenz, dann müssen beim entsprechenden formalen Parameter bei interface-1 gleiche ANY LENGTH, PICTURE, USAGE, SIGN, JUSTIFIED, BLANK WHEN ZERO und SYNCHRONIZEDKlauseln angegeben sein. Zusätzliche Bedingung: Kommt in der Picture Klausel ein Dezimalpunkt vor, dann muss für Interface-1 und interface-2 die gleiche DECIMAL-POINT IS COMMA Klausel gelten.
4. Das Vorhandensein oder Fehlen der RETURNING-Angabe in der Procedure Division stimmt bei einander entsprechenden Methoden überein.
5. Ist das Rückgabe-Element bei einer Methode von interface-2 eine Objektreferenz, dann muss das entsprechende Rückgabe-Element von interface-1 auch eine Objektreferenz sein, die den folgenden Regeln entspricht:
   a. Ist das Rückgabe-Element bei interface-2 eine universelle Objektreferenz, dann muss das entsprechende Rückgabe-Element bei interface-1 ebenfalls eine Objektreferenz sein.
   b. Ist das Rückgabe-Element bei interface-2 mit einer Schnittstelle mit Namen int-r beschrieben, dann muss das entsprechende Rückgabe-Element bei interface-1 eines der folgenden sein:
      - eine Objektreferenz für eine Schnittstelle, die mit int-r konform ist
      - eine Objektreferenz für eine Klasse, gemäß folgender Regeln:
        i. ist eine FACTORY-Angabe vorhanden, muss die Fabrik(Factory)-Schnittstelle der angegebenen Klasse mit int-r konform sein
        ii. fehlt die FACTORY-Angabe, muss die Objekt-Schnittstelle der angegebenen Klasse mit int-r konform sein.
   c. Ist das Rückgabe-Element bei interface-2 mit einem Klassennamen beschrieben, muss das entsprechende Rückgabe-Element bei interface-1 eine Objektreferenz sein, wobei folgende Regeln gelten:
      - ist beim Rückgabe-Element bei interface-2 die ONLY-Angabe vorhanden, dann muss das auch beim Rückgabe-Element von interface-1 der Fall sein und es muss mit demselben Klassennamen beschrieben sein.
      - ist beim Rückgabe-Element bei interface-2 keine ONLY-Angabe vorhanden, dann muss das Rückgabe-Element bei interface-1 mit dem Namen derselben Klasse oder eine ihrer untergeordneten Klassen beschrieben sein.
      - das Vorhandensein oder Fehlen der FACTORY-Angabe muss übereinstimmen.

   d. Ist beim Rückgabe-Element bei interface-1 die ACTIVE-CLASS Angabe vorhanden, dann muss dies auch beim entsprechenden Rückgabe-Element von interface-1 der Fall sein. Außerdem muss das Vorhandensein oder Fehlen der FACTORY-Angabe auch hier analog sein.
   Falls die Beschreibung des Rückgabe-Elementes einer Methode in interface-1 direkt oder indirekt auf interface-2 verweist, dann darf die Beschreibung des Rückgabe-Elementes der entsprechenden Methode in interface-2 sich weder direkt noch indirekt auf interface-1 beziehen.

6. Ist das Rückgabe-Element einer Methode von interface-2 keine Objektreferenz, dann müssen beim korrespondierenden Rückgabe-Element gleiche ANY LENGTH, PICTURE, USAGE, SIGN, JUSTIFIED, BLANK WHEN ZERO und SYNCHRONIZED Klauseln angegeben sein. Zusätzliche Bedingung:
   Kommt in der Picture Klausel ein Dezimalpunkt vor, dann muss für Interface-1 und interface-2 die gleiche DECIMAL-POINT IS COMMA Klausel gelten.

Beispiel 12-24

```
Interface i1
     Methode X      keine Parameter
     Methode Y      using a mit 01 a Pic 999 usage display.
Interface i2
     Methode Y      using b mit 01 b Pic 9(3).
Interface i3
     Methode Y      using c mit 01 c Pic 9(4)
     Methode Z      returning d mit 01 d usage object reference i1
Interface i4
     Methode Z      returning e mit 01 e usage object reference i2
```

Anmerkungen

1. Interface i1 ist konform zu Interface i2, weil Methode Y aus i2 auch in i1 existiert und deren Parameter gleich beschrieben sind (9(3) ist gleichwertig mit 999 und usage display ist Standard).
2. Interface i2 ist nicht konform zu Interface i1, weil Methode X nicht in i2 existiert.
3. Interface i3 ist nicht konform zu Interface i2, weil zwar Methoden aus i2 auch in i3 vorhanden sind, aber die Parameter unterschiedlich beschrieben sind (9(4) statt 9(3)).
4. Interface i3 ist konform zu i4, weil Methode Z in beiden vorkommt und die RückgabeElemente zwar nicht identisch sind, aber doch „passen“: das Interface i1 (aus der Methodendefinition von Z in i3) ist zu i2 (aus der Methodendefinition von Z in i4) konform.

12.7.4.2 Konformität von Parametern und Rückgabe-Elementen

12.7.4.3 Parameter

Die Anzahl der Argumente in der rufenden Quelleinheit muss gleich sein der Anzahl der formalen Parameter in der gerufenen Einheit. Eine Ausnahme bilden gegebenenfalls die formalen „trailing“ Parameter, die in der Procedure Division Überschrift der gerufenen Einheit mit einer OPTIONAL-Angabe spezifiziert sind und in der Liste der Argumente der aufrufenden Einheit nicht aufgeführt sind.
Ist entweder der formale Parameter oder das zugehörige Argument eine stark typisierte Datengruppe, müssen der formale Parameter und das Argument vom gleichen Typ sein (siehe Abschnitt „Typen").
Eine nationale Gruppe wird immer wie ein Datenelement behandelt.

Gruppenelemente

Ist entweder der formale Parameter oder das Argument ein Gruppenelement, so muss jeweils das entsprechende Argument oder der formale Parameter ein Gruppenelement oder ein alphanumerisches Datenelement sein. In diesem Fall darf der formale Parameter kürzer sein als der aktuelle Parameter.

Elementare Daten

Die Konformitäts-Regeln für Datenelemente hängen davon ab, ob ein Argument BY REFERENCE, BY CONTENT oder BY VALUE (Angaben in der CALL bzw. INVOKE-Anweisung) übergeben wird.

BY REFERENCE übergebene Datenelemente

Ist entweder der formale Parameter oder das entsprechende Argument eine Objektreferenz, so muss jeweils das zugehörige Argument oder der formale Parameter eine Objektreferenz sein, gemäß folgender Regeln:

1. Ist entweder das Argument oder der formale Parameter eine universelle Objektreferenz, so muss die jeweilige Entsprechung auch eine universelle Objektreferenz sein.
2. Ist entweder das Argument oder der formale Parameter mit einem Interfacenamen angegeben, so muss die jeweilige Entsprechung den gleichen Interfacenamen haben.
3. Ist der formale Parameter mit einem Klassennamen spezifiziert, so muss das entsprechende Argument mit dem selben Klassennamen spezifiziert sein. Außerdem müssen die FACTORY- und ONLY-Angaben die gleichen sein.
4. Ist der formale Parameter mit ACTIVE-CLASS spezifiziert, so muss eine der folgenden Bedingungen gelten:
   a. Das Argument muss mit ACTIVE-CLASS spezifiziert sein und die Methode muss mit einer der vordefinierten Objektreferenzen SELF oder SUPER oder mit einer Objektreferenz, die mit ACTIVE-CLASS spezifiziert ist, aufgerufen werden. Außerdem muss die FACTORY-Angabe für den formalen Parameter und das Argument gleich sein.
   b. Das Argument muss mit einem Klassennamen und der ONLY-Angabe spezifiziert sein und die Methode muss entweder mit diesem Klassennamen oder mit einer Objektreferenz, die mit diesem Klassennamen und der ONLY-Angabe spezifiziert ist, aufgerufen werden. Außerdem muss die FACTORY-Angabe für den formalen Parameter und das Argument gleich sein.

Ist weder der formale Parameter noch das Argument von der Klasse objekt, gelten folgende Regeln:

1. Ist die gerufene Quelleinheit ein Programm, für das es im Repository-Paragrafen der rufenden Einheit keinen Eintrag eines Programmprototyp-Namen und in der CALL-Anweisung keine NESTED-Angabe gibt, muss der formale Parameter die gleiche Anzahl Zeichenpositionen haben, wie das entsprechende Argument.
2. Ist die gerufene Quelleinheit eine der folgenden,
   - ein Programm, für das im Repository-Paragrafen der rufenden Einheit ein Programmprototyp-Name angegeben ist
   - ein Programm und in der CALL-Anweisung der rufenden Einheit ist NESTED angegeben
   - eine Methode

dann muss die Definition der formalen Parameter und der Argumente die gleichen PICTURE, USAGE, SIGN, JUSTIFIED und BLANK WHEN ZERO Klauseln haben, mit folgenden zusätzlichen Bedingungen:

- kommt in der Picture Klausel ein Dezimalpunkt vor, dann muss für die rufende und die gerufene Quelleinheit die gleiche DECIMAL-POINT IS COMMA Klausel gelten
- hat der formale Parameter die SYNCHRONIZED Angabe, dann muss sie der aktuelle Parameter (Argument) auch haben oder auf Stufe 01 definiert sein.
- Wenn der formale Parameter mit der ANY LENGTH-Klausel definiert ist, wird als seine Länge die Länge des entsprechenden Arguments angenommen.
- Wenn das Argument mit der ANY LENGTH-Klausel definiert ist, muss der zugehörige formale Parameter ebenfalls mit der ANYLENGTH-Klausel definiert sein.
- Wenn das Argument oder der formale Parameter ein typbezogener Zeiger ist, müssen beide typbezogene Zeiger und vom gleichen Typ sein.

BY CONTENT oder BY VALUE übergebene Datenelemente

Ist der formale Parameter eine Objektreferenz, die mit ACTIVE-CLASS spezifiziert ist, so muss eine der folgenden Bedingungen gelten:

1. Die Methode muss mit einer der vordefinierten Objektreferenzen SELF oder SUPER oder mit einer Objektreferenz, die mit ACTIVE-CLASS spezifiziert ist, aufgerufen werden und eine SET-Anweisung mit dem Argument als Sendefeld und einer mit ACTIVE-CLASS spezifizierten Objektreferenz mit gleicher FACTORY-Angabe wie der formale Parameter als Empfangsfeld muss in der rufenden Laufzeit-Einheit gültig sein.
2. Die Methode muss mit einem Klassennamen oder mit einer Objektreferenz, die mit einem Klassennamen und der ONLY-Angabe spezifiziert ist, aufgerufen werden und eine SET-Anweisung mit dem Argument als Sendefeld und einer mit diesem Klassennamen und der ONLY-Angabe spezifizierten Objektreferenz mit gleicher FACTORY-Angabe wie der formale Parameter als Empfangsfeld muss in der rufenden Laufzeit-Einheit gültig sein.

Ist der formale Parameter eine Objektreferenz, die nicht mit ACTIVE-CLASS spezifiziert ist, dann sind die Konformitäts-Regeln die gleichen, wie wenn in der rufenden Laufzeit-Einheit eine SET-Anweisung ausgeführt wird, mit dem Argument als Sendefeld und dem entsprechenden formalen Parameter als Empfangsfeld.

Ist der formale Parameter nicht von der Klasse „objekt“, gelten folgende Konformitäts-Regeln:

1. Ist die gerufene Quelleinheit ein Programm, für das es im Repository-Paragrafen der rufenden Einheit keinen Eintrag eines Programmprototyp-Namen und in der CALL-Anweisung keine NESTED-Angabe gibt, muss der formale Parameter die gleiche Anzahl Zeichenpositionen haben, wie das entsprechende Argument.
2. Ist die gerufene Quelleinheit eine der folgenden,
   - ein Programm, für das im Repository-Paragrafen der rufenden Einheit ein Programmprototyp-Name angegeben ist
   - ein Programm und in der CALL-Anweisung der rufenden Einheit ist NESTED angegeben
   - eine Methode

   dann gelten folgende Regeln, abhängig von der Art des formalen Parameters:
   a. Ist der formale Parameter numerisch, muss eine COMPUTE-Anweisung, mit dem Argument als Sendefeld und dem entsprechenden formalen Parameter als Empfangsfeld, zulässig sein.
   b. Ist der formale Parameter ein Index-Datenelement oder ein Datenelement von der Klasse zeiger, muss eine SET-Anweisung, mit dem Argument als Sendefeld und dem entsprechenden formalen Parameter als Empfangsfeld, zulässig sein.
   c. In den übrigen Fällen muss eine MOVE-Anweisung, mit dem Argument als Sendefeld und dem entsprechenden formalen Parameter als Empfangsfeld, zulässig sein.

d. Wenn der formale Parameter mit der ANYLENGTH-Klausel definiert ist, gilt:
   - Das zugehörige Argument muss von der Klasse alphabetisch, alphanumerisch oder national und mit der gleichen USAGE DISPLAY oder USAGE NATIONAL definiert sein.
   - Das zugehörige Argument darf **nicht** mit der ANY LENGTH-Klausel definiert sein.
   - Die Länge des formalen Parameters entspricht der Länge des Arguments.

12.7.4.4 Rückgabe-Elemente

Ein Rückgabe-Element muss in einer rufenden Anweisung dann spezifiziert sein, wenn ein Rückgabe-Element in der Procedure Division Überschrift der gerufenen Quelleinheit definiert ist.

Ist einer der Operanden eine stark typisierte Datengruppe, so müssen beide vom gleichen Typ sein.

Eine nationale Gruppe wird immer wie ein Datenelement behandelt.

Gruppenelemente

Ist einer der Operanden ein Gruppenelement, so muss das entsprechende Element ein Gruppenelement oder ein alphanumerisches Datenelement und gleich lang sein.

Datenelemente

Ist einer der Operanden eine Objektreferenz, muss das entsprechende Element auch eine Objektreferenz sein und es gelten dann folgende Regeln:

1. Ist im Rückgabe-Element der gerufenen Quelleinheit keine ACTIVE-CLASS Angabe spezifiziert, dann muss im gerufenen Laufzeit-Element eine SET-Anweisung zulässig sein, mit dem Rückgabe-Element in der gerufenen Quelleinheit als Sendefeld und dem entsprechenden Rückgabe-Element in der rufenden Quelleinheit als Empfangsfeld.
2. Ist im Rückgabe-Element der gerufenen Quelleinheit eine ACTIVE-CLASS Angabe vorhanden, dann muss im rufenden Laufzeit-Element eine SET-Anweisung zulässig sein, mit dem Rückgabe-Element in der rufenden Quelleinheit als Empfangsfeld und einem Sendefeld, das nach folgenden Regeln mit einer USAGE OBJECT REFERENCE Angabe spezifiziert ist:
   a. Wird eine Methode mit einem Klassennamen aufgerufen, muss das Sendefeld mit dem gleichen Klassennamen und einer ONLY-Angabe spezifiziert sein.
   b. Wird eine Methode mit der vordefinierten Objektreferenz SELF oder SUPER aufgerufen, muss das Sendefeld mit der ACTIVE-CLASS Angabe spezifiziert sein.
   c. Wird eine Methode mit einer Objektreferenz aufgerufen, die mit einem Interfacenamen beschrieben ist, muss das Sendefeld eine universelle Objektreferenz sein.
   d. Wird eine Methode mit einer anderen Objektreferenz, als die vorher genannten, aufgerufen, so wird diese als Sendefeld verwendet, einschließlich einer eventuell vorhandenen ONLY-Angabe.
   e. Genügt das Sendefeld einer der aufgeführten Regeln und ist es mit einem Klassennamen oder ACTIVE-CLASS beschrieben, so muss das Vorhandensein oder Fehlen der FACTORY-Angabe dem im Rückgabe-Element der gerufenen Quelleinheit entsprechen.
   f. Sind die Operanden keine Objektreferenzen, dann gilt das unter b), für ’BY REFERENCE übergebene Datenelemente’, gesagte.

   Ist weder das formale noch das aktuelle Rückgabeelement von der Klasse objekt, dann müssen beide Operanden die gleiche PICTURE, USAGE, SIGN, JUSTIFIED und BLANK WHEN ZERO Klauseln haben, mit folgenden zusätzlichen Bedingungen:
   - kommt in der Picture Klausel ein Dezimalpunkt vor, dann muss für die rufende und die gerufene Quelleinheit die gleiche DECIMAL-POINT IS COMMA Klausel gelten
   - hat das formale Rückgabeelement die SYNCHRONIZED Angabe, dann muss sie das aktuelle Rückgabeelement auch haben oder auf Stufe 01 definiert sein.
   - Wenn das formale Rückgabeelement mit der ANY LENGTH-Klausel definiert ist, wird als seine Länge die Länge des aktuellen Rückgabeelements angenommen.
   - Wenn das aktuelle Rückgabeelement mit der ANY LENGTH-Klausel definiert ist, muss das zugehörige formale Rückgabeelement ebenfalls mit der ANY LENGTH-Klausel definiert sein.
   - Wenn einer der beiden Operanden ein typbezogener Zeiger ist, müssen beide typbezogene Zeiger und vom gleichen Typ sein.

12.7.5 Die Systemklasse BASE

BASE dient als Basis, um Objekte zu erzeugen. Wird diese Klasse BASE nicht verwendet, so kann nur mit Fabrikobjekten gearbeitet werden.
Die Schnittstellendefinition BaseFactoryInterface beschreibt die Schnittstelle des Fabrikobjektes der Klasse BASE. Die Schnittstellendefinition BaseInterface beschreibt die Schnittstelle der Objekte der Klasse BASE. Die Klasse BASE ist so definiert, als wären die IMPLEMENTS-Angaben für die Schnittstellen BaseFactoryInterface bzw. BaseInterface im FACTORY- bzw. OBJECT-Paragraf gemacht worden.
Sowohl die Klasse BASE als auch die Schnittstellendefinitionen BaseFactoryInterface und BaseInterface werden vom Compiler bzw. Laufzeitsystem bereitgestellt.

Die BASE Klasse enthält die Methoden ’NEW’ und ’FACTORYOBJECT’.

12.7.5.1 Methode NEW

NEW ist eine Fabrikmethode, die einen Standardmechanismus bietet, um Objekte einer Klasse zu erzeugen. Mit COBOL-Sprachmitteln beschrieben, sieht die Schnittstelle der Methode NEW wie folgt aus:

```
Interface-id. BaseFactoryInterface.
Procedure division.
   Method-id. New.
   Data division.
   Linkage section.
   01 outObject usage object reference active-class.
   Procedure division returning outObject.
   End-method New.
End Interface BaseFactoryInterface.
```

Allgemeine Regeln

1. Die Methode NEW teilt Speicherplatz für ein Objekt zu und initialisiert seine Daten. Ein Objekt befindet sich unmittelbar nach seiner Erzeugung im Initialzustand.
2. Ist für das Erzeugen eines neuen Objektes nicht ausreichend Speicherplatz vorhanden, so entsteht die Ausnahmesituation EC-OO-RESOURCE. Der weitere Ablauf erfolgt entsprechend Regel 6 f) der INVOKE-Anweisung (siehe Abschnitt „INVOKE-Anweisung").

Beispiel 12-25

Methode NEW überschreiben

Die Methode NEW kann überschrieben werden, um zusätzliche objektspezifische Initialisierungen auszuführen, die über die Initialwerte aus den VALUE Klauseln hinausgehen. In diesem Fall ist vorher das neue Objekt mit Hilfe der „originalen“ NEW Methode zu erzeugen.

```
...
FACTORY.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ident pic 9(9)  COMP VALUE 0.
...
METHOD-ID.         NEW OVERRIDE.
DATA DIVISION.
LINKAGE SECTION.
01 neues-objekt    USAGE OBJECT REFERENCE ACTIVE-CLASS.
PROCEDURE DIVISION RETURNING neues-objekt.
1.
   INVOKE SUPER "NEW" RETURNING neues-objekt.
   COMPUTE ident = ident + ...
   INVOKE neues-objekt "SPEZIELLE-INITIALISIERUNGEN" USING ident...
   ...
2.
   EXIT METHOD.
END METHOD         NEW.
...
```

12.7.5.2 Methode FactoryObject

FactoryObject ist eine Objektmethode, die einen Standardmechanismus bietet, um das Fabrikobjekt einer Klasse verwenden zu können. Mit COBOL-Sprachmitteln beschrieben, sieht die Schnittstelle der Methode FactoryObject wie folgt aus:

```
Interface-id. BaseInterface.
Procedure division.
   Method-id. FactoryObject.
   Data division.
   Linkage section.
   01 outFactory usage object reference factory of active-class.
   Procedure division returning outFactory.
   End method FactoryObject.
End Interface BaseInterface.
```

Allgemeine Regel

Wird die Methode FactoryObject für ein Objekt aufgerufen, dann bestimmt sie die Klasse des Objekts und verweist auf das Fabrikobjekt dieser Klasse.

Beispiel 12-26

Die Methode FactoryObjekt ist hilfreich, wenn die Klasse eines Objektes nicht bekannt ist. Ist die Klasse bekannt, so kann man auf die Methoden eines Fabrikobjektes wie folgt zugreifen:

```
INVOKE KLASSENNAME "XYFACTORYMETHODENNAME"
```

Ist die Klasse eines Objektes nicht bekannt, so kann trotzdem eine ihrer Fabrikmethoden wie folgt aufgerufen werden:

```
INVOKE OBJECTA "FACTORYOBJECT" RETURNING FACTORYOBJECTN
INVOKE FACTORYOBJECTN "XYFACTORYMETHODENNAME"
```

12.7.6 Automatische Speicherfreigabe (Garbage Collection)

Mit COBOL2000 braucht sich der Programmierer nicht um die Freigabe von Speicherplatz zu kümmern, der von nicht mehr benötigten Objekten seines Programms belegt wird. Diese Aufgabe übernimmt für COBOL2000-Programme ein Garbage Collector. Allerdings fordert der Garbage Collector den von den Objekten belegten Speicherplatz nicht bereits zum Zeitpunkt zurück, ab dem das Programm diese Objekte nicht mehr benötigt. Vielmehr fordert der Garbage Collector den von den Objekten nicht mehr benötigten Speicherplatz erst zurück, wenn der insgesamt zur Verfügung stehende Speicherplatz knapp wird. Hierbei kann es zu einem vorübergehenden, scheinbaren Programmstillstand kommen.

12.8 Datentypen

Ein Typ ist eine Vorlage, die alle Eigenschaften der Datenbeschreibung und ihrer untergeordneten Datenfelder enthält. Ein Typ wird durch Angabe der TYPEDEF-Klausel definiert. Die wesentlichen Eigenschaften eines Typs, der durch seinen Typnamen identifiziert wird, sind die relativen Positionen und Längen ihrer Datenelemente, die in der Typdeklaration definiert sind, sowie die Klauseln BLANK WHEN ZERO, JUSTIFIED, PICTURE, SIGN, SYNCHRONIZED und USAGE, die für diese Datenelemente angegeben sind (Einzelheiten zu den Klauseln finden Sie im Abschnitt „Datenerklärung").

Wir unterscheiden folgende Arten von Datentypen:

- schwach typisierte Datenelemente
- schwach typisierte Datengruppen
- stark typisierte Datengruppen

Anmerkung: Datenelemente können nicht stark typisiert sein.
Ein Typ - ob schwach oder stark - definiert eine bestimmte Datenstruktur, die einen genau bezeichneten Namen hat. Verwendet wird sie genau mit dieser Struktur. Dadurch wird verhindert, dass die Struktur, z.B. durch unterschiedliche Ausrichtung, oder Wechsel der Datendarstellung mittels USAGE Angabe auf übergeordneten Gruppenstufen, beeinflusst wird.

12.8.1 Schwach typisierte Datenbeschreibungen

Schwach typisierte Datenbeschreibungen beziehen sich auf eine Typdeklaration, die nicht die STRONG-Angabe enthält und sind auch solchen Typdeklarationen nicht untergeordnet. Schwach typisierte Datenbeschreibungen können entweder Datengruppen oder Datenelemente sein.
Ihre Datenstruktur wird nicht in jedem Fall, wie bei den stark typisierten Datenbeschreibungen beschrieben, geschützt, da schwach typisierte Datenbeschreibungen praktisch wie jede untypisierte Datenbeschreibung benutzt werden können. Man könnte die Typdeklaration als Abkürzung für eine Reihe von Datenbeschreibungen ansehen.

Das folgende Beispiel veranschaulicht die Verwendung der TYPEDEF-Klausel, um einen Typ zu definieren, und der TYPE-Klausel, um einen Typ zu benutzen:

Beispiel 12-27

```
01 MitarbeiterIn TYPEDEF.                    *> definiert Typnamen
MitarbeiterIn
   02 name PIC X(30).                        *> mit dieser Beschreibung
   02 personalnummer PIC 9(8).
01 Abteilung.
   02 Abt-bezeichnung PIC X(16).
   02 Abt-leiterIn TYPE MitarbeiterIn.     *> verwendet Typnamen
MitarbeiterIn
   02 Abt-stellvertreterIn TYPE MitarbeiterIn.
   02 Abt-sekretariat TYPE MitarbeiterIn.
```

Dies führt zu einem Datensatz mit der Beschreibung:

```
01 Abteilung.
   02 Abt-bezeichnung PIC X(16).
   02 Abt-leiterIn.
      03 name PIC X(30).
      03 personalnummer PIC 9(8).
   02 Abt-stellvertreterIn.
      03 name PIC X(30).
      03 personalnummer PIC 9(8).
   02 Abt-sekretariat.
      03 name PIC X(30).
      03 personalnummer PIC 9(8).
```

Die Expansion eines Typs kann Hierarchien mit einer Tiefe von mehr als 49 Stufen erzeugen, die sonst in einer Datenbeschreibung maximal direkt hingeschrieben werden dürfen.

12.8.2 Stark typisierte Datenbeschreibungen

Stark typisierte Datenbeschreibungen werden mit einer Typdeklaration definiert, die die STRONG-Angabe enthält oder sind einer Typdeklaration mit der STRONG-Angabe untergeordnet. Nur Datengruppen können stark typisiert sein, da das Konzept starker Typisierung für Datenelemente eine Vielzahl an gravierenden Einschränkungen bedeuten würde.

Neben dem Schutz der Datenstruktur, welche durch den Typ festgelegt ist, soll die starke Typisierung vor allem die Integrität der Dateninhalte schützen. So sollte die richtige Anwendung von stark typisierten Datenbeschreibungen nie zu Daten führen, die inkompatibel zu ihrer Datenbeschreibung sind.
Eine wichtige Konsequenz ist daher, jegliche impliziten und expliziten Redefinitionen zu verbieten.

Eine Datengruppe kann als eine Art Neudefinition der ihr untergeordneten Datenelemente betrachtet werden. Daraus folgt, dass Zugriffe auf stark typisierte Datengruppen auf solche eingeschränkt werden, die die Integrität der Daten, die dieser Datengruppe untergeordnet sind, nicht tangieren. Wenn eine stark typisierte Datengruppe ein Empfangsfeld ist, muss das Sendefeld vom „gleichen Typ“ sein. Der „gleiche Typ“ ist definiert als Typbeschreibung mit dem gleichen Namen und den gleichen relevanten Eigenschaften.

Es besteht keine Notwendigkeit, dieselben Einschränkungen für Datenelemente zu fordern, da Zugriffe auf Datenelemente im Allgemeinen den Inhalt des Empfangsfeldes nicht korrumpieren.

Weitere Restriktionen existieren für stark typisierte Datengruppen und Datenelemente, die stark typisierten Datengruppen untergeordnet sind. Um die Datenintegrität zu gewährleisten, können sie nicht umbenannt (RENAMES-Klausel) werden. Teilfeldselektion ist nur für alphanumerische und nationale Datenelemente erlaubt.

Restriktionen für Adressen von stark typisierten Datengruppen und Zeiger, die solche Adressen enthalten, werden im Abschnitt „Typbezogene Zeiger" beschrieben.

Es ist nur dann sinnvoll, die Integrität der Daten zu schützen, wenn die Daten „korrekt“ sind.

12.9 Adressen und Zeiger

Adressen und Zeiger beziehen sich auf den Speicher im Rechner und auf Adressen, die es erlauben, Speicherstellen aufzufinden. Im Normalfall wird von dieser Möglichkeit in Anwendungen kaum Gebrauch gemacht.

Im Sinne der universellen Verwendbarkeit von Programmiersprachen und der Notwendigkeit des Zusammenspiels mit Systemen wie POSIX oder Programmiersprachen wie C und C++, wurde dieses Adress- und Zeiger-Handling auch in COBOL zur Verfügung gestellt. Adress- und Zeiger-Handling sollte aber sorgfältig eingesetzt werden, da Lesbarkeit und Wartbarkeit von Programmen mit dieser Programmiertechnik erschwert werden können.

12.9.1 Datenadressen und Datenzeiger

Der Begriff **Datenadresse** bezeichnet den Speicherplatz eines Datenelements. Er wird mittels eines Datenadressbezeichners angesprochen. Ein Datenadressbezeichner darf nicht als Empfangsfeld verwendet werden. Dabei ist zu beachten, dass die ADDRESS OF-Angabe für das Empfangsfeld in der SET-Anweisung keinen Datenadressbezeichner darstellt, sondern Bestandteil der SET-Syntax ist.

Ein **Datenzeiger** ist ein Datenelement, in dem eine Datenadresse gespeichert werden kann.

Die Datenadresse eines mit einer BASED-Klausel definierten Datenelements kann entweder mittels eines Datenadressbezeichners oder eines Datenzeigers gesetzt werden. Datenadressen und Datenzeiger können an andere Quelleinheiten übergeben, Datenzeiger auch von anderen Quelleinheiten zurückgegeben werden.

12.9.2 Verwendung von Datenzeigern

Zeiger können wie folgt verwendet werden:

1. als formale Parameter in der PROCEDURE DIVISION USING-Angabe
2. als aktuelle Parameter in der CALL- oder INVOKE-Anweisung
3. in der SET-Anweisung: Zuweisen und Aktualisieren einer Adresse (UP/DOWN)
4. in Vergleichsbedingungen: vergleichen mit der vordefinierten Adresse NULL, mit einem Datenadressbezeichner und anderen Zeigern auf Gleichheit und Ungleichheit
5. in der LENGTH-Funktion: diese Funktion liefert den Wert 4 für FUNCTION LENGTH (LENGTH OF ...) und FUNCTION LENGTH (zeiger). FUNCTION LENGTH (ADDRESS OF ...) wird derzeit nicht unterstützt.

Beispiel 12-28

Ein Unterprogramm GNR liefere entsprechend folgendem Programmprototyp einen Zeiger auf einen Datensatz zurück:

```
Program-id. Get-Next-Record is prototype. *> returns the address of a
                                             record
Data Division.
Linkage Section.
01 ptrl usage pointer.
Procedure Division returning ptrl.
End-program Get-Next-Record.
```

Ein Anwenderprogramm habe folgende Deklarationen:

```
program get-next-record  *> in the Repository Paragraph

01 p usage pointer.  *> in the Working-Storage Section

01 my-wreck based.   *> in the Linkage Section
   02 name pic x(30).
   02 addr pic x(30).
```

Folgende Anweisung der Procedure Division ruft das durch den Prototyp beschriebene Programm:

```
Call "GNR" as Get-Next-Record returning p
```

Auf das Datum kann mit my-wreck zugegriffen werden, da der Zeiger p die Adresse eines Datensatzes enthält:

```
Set address of my-wreck to p
Move "SAM JONES" to name in my-wreck
```

Beispiel 12-29

```
01 p2 usage pointer.
01 data-record.     *> the full record layout is described
   02 ...
```

Soll ein Zeiger dem Programm Process-Record "PR“ übergeben werden, dann kann wie folgt codiert werden:

```
Set p2 to adress of data-record.
Call "PR" using p2.
```

Alternativ kann die Adresse von `data-record` auch folgendermaßen übergeben werden * :

```
Call "PR" using address of data-record
```

* in diesem Fall bleibt jedoch eine mögliche Änderung des übergebenen Zeigers durch das gerufene Programm für den Rufer ohne Wirkung.

12.9.3 Programmadressen und Programmzeiger

Der Begriff Programmadresse bezeichnet die Adresse eines Einsprungpunktes des Programmes. Er wird mittels eines Programmadressbezeichners angesprochen. Ein Programmadressbezeichner darf nicht als Empfangsfeld verwendet werden.

Ein Programmzeiger ist ein Datenelement, in dem eine Programmadresse gespeichert werden kann.

Ein Programmzeiger kann dazu verwendet werden, ein Programm aufzurufen. Programmadressbezeichner und Programmzeiger können an andere Quelleinheiten übergeben, Programmzeiger auch von anderen Quelleinheiten zurückgegeben werden.

12.9.4 Verwendung von Programmzeigern

Programmzeiger können wie folgt verwendet werden:

1. als das zu rufende Programm in der CALL-Anweisung
2. als formale Parameter in der PROCEDURE DIVISION USING-Angabe
3. als aktuelle Parameter in der CALL- oder INVOKE-Anweisung
4. in der SET-Anweisung: Zuweisen von Programmzeigern oder Programmadressbezeichnern
5. in der INITIALIZE-Anweisung, um Programmzeiger-Datenfelder vorzubesetzen
6. in Vergleichsbedingungen: vergleichen mit der vordefinierten Adresse NULL, mit einem Programmadressbezeichner und anderen Programmzeigern auf Gleichheit und Ungleichheit
7. in der LENGTH-Funktion: diese Funktion liefert den Wert 4 für FUNCTION LENGTH (programmzeiger). FUNCTION LENGTH (ADDRESS OF PROGRAM ...) wird derzeit nicht unterstützt.

Beispiel 12-30

Aufruf eines Programmes mittels Programmzeiger:

```
01 pgmptr USAGE PROGRAM-POINTER.
SET pgmptr TO ADDRESS OF PROGRAM "UPROGP".
CALL pgmptr.
```

Beispiel 12-31

Aufruf eines Unterprogrammes mit einem Programmzeiger als Parameter:

```
01 pgmptr USAGE PROGRAM-POINTER.
SET pgmptr TO ADDRESS OF PROGRAM "UPROGP".
CALL "UNTER" USING pgmptr.
```

12.9.5 Typbezogene Zeiger

Ein typbezogener Datenzeiger darf nur die Adresse einer Datenbeschreibung vom gleichen Typ enthalten. Er darf nur innerhalb einer Typbeschreibung definiert werden. Die Nutzung von typbezogenen Datenzeigern ist wichtig, um die Sicherheit von Typen zu unterstützen. Damit wird vermieden, Daten eines Typs wie Daten eines anderen Typs zu behandeln.

Diese Integrität ist besonders bei stark typisierten Datenfeldern wichtig. Die Adresse eines stark typisierten Datenfelds wird als typbezogener Zeiger betrachtet. Deshalb kann sie nur einem Zeiger zugewiesen werden, der auf den gleichen Typ bezogen ist.

Zudem kann ein typbezogener Zeiger nur benutzt werden, um ein Datenelement, das mit der BASED-Klausel spezifiziert ist, mit dem gleichen Typ zu adressieren. Umgekehrt kann die Adresse eines stark typisierten Datenelements, das mit der BASED-Klausel spezifiziert ist, nur auf die Adresse eines Datenelements vom gleichen Typ gesetzt werden. Dadurch können bestehende Einschränkungen, die die Integrität von stark typisierten Datenelementen erzwingen, nicht durch Nutzung von Adressen, Zeigern und Datenfeldern mit der BASED-Klausel umgangen werden.

12.10 Sprachmittel zur Verarbeitung von XML

COBOL2000 bietet Sprachmittel für das Lesen von XML-Dokumenten. Solche Dokumente können in Dateien oder im Arbeitsspeicher bereitgestellt sein. Ein separates Open Source-Programmpaket, der **Parser**, analysiert und zerlegt das XML-Dokument. Dieses Programmpaket wird nicht mit dem COBOL-Compiler bzw. CRTE ausgeliefert, sondern steht im Internet zum Herunterladen zur Verfügung. Sie müssen den Parser vor dem Ablauf von COBOL-Programmen, die XML-Sprachmittel verwenden, bereitstellen, und dafür sorgen, dass er sich zu ihnen gesellt.

Die Spracherweiterungen von COBOL2000 erlauben es, ein XML-Dokument auf zwei verschiedene Arten zu verarbeiten:

- strukturorientiert, basierend auf einer Baumdarstellung des Dokuments
- ereignisorientiert, basierend auf einer rein sequenziellen Sichtweise des Dokuments

	Strukturorientiert	**Ereignisorientiert**
Sprachmittel	wie im Technical Report "Native COBOL Syntax for XML Support"	angelehnt an Spracherweiterung anderer Compiler-Anbieter
Dokument	im Speicher oder in Datei	nur im Speicher
Zugriff auf Teile des XML-Dokuments	beliebig Gleiche Teile sind auch wie derholt möglich.	sequenziell Jedes Teil ist nur ein einzi ges Mal möglich.
gelieferte Daten	eigentliche Daten des Dokuments, normalisiert	fast alles aus dem Dokument
zusätzlicher Verarbeitungsaufwand des Parsers	Das Dokument muss in die Baumdarstellung überführt werden.	Das Dokument kann direkt verarbeitet werden.
zusätzlicher Speicherbedarf des Parsers	hoch (für den gesamten Baum)	gering (für Verwaltungsdaten)
Parser-Schnittstelle	DOM (Document Object Model)	SAX (Simple API for XML)

Tabelle 46: Vergleich von struktur- und ereignisorientierter Verarbeitungsweise

Um die neuen Sprachmittel nutzen zu können, muss beim Übersetzen des Programms die entsprechende Option eingeschaltet sein.

12.10.1 Strukturorientierte Verarbeitung

Der Parser transformiert das gesamte XML-Dokument zu Beginn der Verarbeitung in eine Baumdarstellung im Arbeitsspeicher. Den hierarchischen Strukturen des XML-Dokuments entsprechen im COBOL-Programm die bekannten Datenstrukturen. Mit Hilfe von zwei zusätzlichen neuen Klauseln lässt sich damit die Struktur des XML-Dokuments beschreiben. Dabei müssen Sie jedoch nicht das komplette Dokument beschreiben, sondern es genügt, diejenigen Teile zu beschreiben, die das Programm verarbeiten will. Die erweiterten Anweisungen für Dateiverarbeitung erlauben dann die Zuordnung solcher COBOL-Datenstrukturen zu konkreten Positionen im Baum, die Übertragung von derart ausgewählten Daten aus dem Baum und den Zugriff darauf im COBOL-Programm.

12.10.1.1 XML-Dokument als Baum

Für das Verständnis der strukturorientierten Verarbeitung ist es hilfreich, die hierarchische Ordnung von Elementen bzw. Attributen eines XML-Dokuments als Baumstruktur aufzufassen. Eine solcher Baum sei, stark vereinfacht, wie folgt definiert:

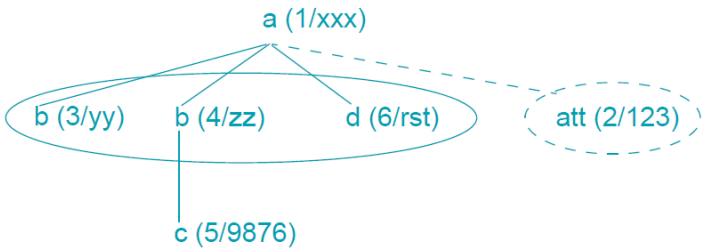
- Jedes Element bzw. Attribut aus dem XML-Dokument entspricht einem **Knoten** im Baum. Ihm ist der Name und der Wert des Elements bzw. Attributs zugeordnet. Dabei müssen Elementnamen nicht eindeutig sein; Attributnamen müssen nur relativ zu ihren Elementnamen eindeutig sein.
- Jeder **Element-Knoten** hat zwei **geordnete** Folgen von ihm **direkt untergeordneten** Knoten, seine **Kinder**: eine Folge von Element-Knoten und eine Folge von AttributKnoten. Jede dieser beiden Folgen kann leer sein, einen oder mehrere Knoten enthalten.
- Der erste Knoten in einer der geordneten Folgen ist das **älteste**, der letzte das **jüngste Kind** und folglich heißen alle nach einem Knoten in einer solchen Folge stehenden Knoten **jüngere Geschwister**, alle davor stehenden Knoten **ältere Geschwister**.
- Das äußerste, alles umfassende Element des XML-Dokuments stellt die **Wurzel** des ganzen Baumes dar. Analog dazu stellt jeder andere Element-Knoten gleichzeitig auch die Wurzel eines **Teilbaums** dar. Ein Knoten legt einen Teilbaum fest; der Begriff Teilbaum schließt auch einen einzelnen Knoten, der keine Kinder hat, mit ein.
- Mit Ausnahme der Wurzel des gesamten Baums hat jeder Knoten im Baum genau einen direkt übergeordneten Knoten, seinen **Vater**.
- Knoten, die keine Kinder haben, werden auch **Blätter** genannt.
- Jedes Element bzw. Attribut im Baum hat eine eindeutige **Position**. Mit ihrer Hilfe wird die Verbindung zwischen der Beschreibung im COBOL-Programm und den aktuell zugeordneten Elementen bzw. Attributen hergestellt.

Beispiel 12-32 XML-Dokument als Baum

XML-Dokument:

```
<a
   att="123"
>xxxx
   <b>yy</b>
   <b>zz
      <c>9876<
/c>
  </b>
  <d>rst</d>
</a>
```

in Baumdarstellung:



Bei jedem Knoten ist in Klammern die Position und der Wert vermerkt. Element-Knoten sind mit durchgezogenen Linien dargestellt, Attribut-Knoten gestrichelt.

Anmerkungen:

- 'a' bildet die Wurzel des gesamten Baums.
- Ihre Kinder sind die Knoten an den Positionen 2, 3, 4 und 6.
- Die Folge der Element-Knoten besteht aus den beiden 'b' und dem 'd'.

- Die Folge der Attribut-Knoten besteht aus einem einzigen Knoten, dem 'att'.
- Das 'b' an Position 4 hat als jüngere Geschwister das 'd' und als ältere Geschwister das 'b' an Position 3.

- ’a’ ist der Vater der beiden 'b', des 'd' und des 'att'-Knoten
- Der Knoten 'a' hat sowohl Element-, als auch Attribut-Knoten als Kinder. Der Knoten an Position 4 hat nur einen einzigen Element-Knoten als Kind, aber keine Attribut-Knoten. Die Knoten an den Positionen 2, 3, 5 und 6 sind Blätter, sie haben keine Kinder, also weder untergeordnete Element- noch Attribut-Knoten.

12.10.1.2 COBOL-Sprachmittel zur Beschreibung eines XML-Dokuments

Das in COBOL bereits vorhandene Stufenkonzept für die Strukturierung von Datensätzen wird verwendet, um die hierarchische Struktur eines XML-Dokuments darzustellen.

Der Übersichtlichkeit halber werden die neuen COBOL-Sprachmittel im ersten Schritt ohne Berücksichtigung von Namensräumen vorgestellt. Die Besonderheiten bei der Beschreibung und Verarbeitung von Namensräumen entnehmen Sie Abschnitt "Namensraum (namespace)".

Einem Knoten aus dem Baum entspricht ein Datenfeld in der COBOL-Datenstruktur. in der COBOL-Struktur darf es zusätzlich weitere Datenfelder geben, denen im Baum kein Knoten entspricht.

- Das einer Wurzel entsprechende Datenfeld hat Stufennummer 01.
- Die Datenfelder für **Kinder eines Knoten** haben alle die **gleiche Stufennummer**, die größer ist als die ihres Vaters.
- Die neue IDENTIFIED-Klausel kennzeichnet in der COBOL-Struktur diejenigen Datenfelder, denen im Baum ein Knoten entspricht, und gibt die **Art** des Knotens, Element bzw. Attribut, an.
- Die IDENTIFIED-Klausel dient gleichzeitig dazu, den **Namen** eines Elements bzw. Attributs anzugeben, wie er im XML-Dokument in den Tags steht. Klein- bzw. Großschreibung ist dabei relevant.
- Das Datenfeld, das den **Wert** eines Elements bzw. Attributs aufnehmen soll, wird in der Struktur dem Datenfeld mit der IDENTIFIED-Klausel für das Element bzw. Attribut **direkt untergeordnet**. Wenn dieses Datenfeld das einzige einer IDENTIFIED-Klausel untergeordnete Datenfeld ist, darf es entfallen, und die PICTURE-Klausel für den Wert kann direkt zusammen mit der IDENTIFIED-Klausel angegeben werden.
- Wenn für ein Datenfeld eine IDENTIFIED-Klausel angegeben ist, müssen auch alle in der Struktur übergeordneten Datengruppen eine IDENTIFIED-Klausel haben. Das bedeutet, dass die hierarchische Struktur des Baums lückenlos mit der COBOL-Struktur wiedergegeben werden muss. Zusätzliche Zwischenebenen auf der COBOL-Seite – selbst wenn sie nur einer besseren Strukturierung dienen sollen – sind nicht erlaubt.

Beispiel 12-33 COBOL-Beschreibung des gesamten XML-Dokuments

XML-Dokument

```
<a
   att="123">

   xxxx
   <B>zz

      <c>9876</c>

   </B>
</a>
```

COBOL-Datenstruktur

```
01 wurzel             IDENTIFIED BY "a" ELEMENT.
   02 wurzel-att      IDENTIFIED BY "att" ATTRIBUTE.
      03 att-wert     PIC 999.
   02 wurzel-wert     IX X(10).
   02 kind            IDENTIFIED BY "B".
```

```
    03 kind-wert       PIC X.
       03 enkel        IDENTIFIED BY "c" ELEMENT
                      PIC 9(8) BINARY.
```

Anmerkungen:

- Das Datenfeld enkel ist nicht weiter unterstrukturiert und kann daher gleichzeitig neben der Angabe des Elementnamens (’c’) auch zur Aufnahme des Wertes verwendet werden.-
- Die Werte aus dem XML-Dokument werden entsprechend der COBOL-Beschreibung zur Verfügung gestellt:
  - numerische Werte am Dezimalpunkt ausgerichtet, ggf. konvertiert (z.B. Datenfeld enkel)
  - alphanumerische Werte ggf. abgeschnitten (z.B. Datenfeld kind-wert) bzw. mit Leerzeichen aufgefüllt (z.B. Datenfeld wurzel-wert)
- Es ist nicht notwendig, dass für den Wert eines Knoten auch ein Datenfeld in der COBOL-Datenstruktur definiert wird, z.B. wenn der Knoten im Baum nie einen Wert hat, oder das Programm den Wert nicht verarbeiten will.
- Die Ende-Tags aus dem XML-Dokument tauchen in der COBOL-Darstellung nicht auf. Sie sind implizit in der hierarchischen Struktur enthalten.
- ELEMENT als Art eines Knotens ist der Defaultwert und darf daher in der IDENTIFIED-Klausel auch weggelassen werden (z.B. bei der Datengruppe kind).
- Die COBOL-Beschreibung enthält auch Datenfelder, die keinem Knoten im Baum entsprechen (z.B. att-wert, wurzel-wert und kind-wert).
- Verwechseln Sie nicht das in der IDENTIFIED-Klausel angegebene Literal mit dem Initialwert des Datenfeldes. Dieser wird nur durch eine VALUE-Klausel angegeben.

Angabe eines Element- bzw. Attribut-Namens in der IDENTIFIED-Klausel

Für die Angabe des Namens eines Elements bzw. Attributs in der IDENTIFIED-Klausel gibt es mehrere Möglichkeiten, siehe nachfolgende Liste.

Die Besonderheiten von namespaces, die auch Bestandteil des Namens sind, werden der Übersichtlichkeit halber im Abschnitt "Namensraum (namespace)" zusammenfassend dargestellt.

- Sie haben den Namen **vorgegeben** mittels IDENTIFIED **BY**
  - konstant, als Literal
  - variabel, als Namen eines Datenfeldes, das den aktuellen Element- bzw. Attributnamen enthält
- Der Name ist **nicht vorgegeben**; statt dessen wird der im Dokument gefundene Name zurückgegeben mittels IDENTIFIED **USING**.
  Dies kann man auch als eine besondere Form von 'vorgegeben' ansehen, bei der nicht exakt ein einziger Name, sondern alle möglichen Namen vorgegeben sind, also eine Art Wild Card-Notation.
- Die in einer IDENTIFIED-Klausel angegebenen Datenfelder für die Element- bzw. Attributnamen dürfen 'fast überall' im Programm definiert sein. Wenn ihre Datenerklärung jedoch in einer Datenstruktur für ein XML-Dokument steht, muss diese Datenerklärung der IDENTIFIED-Klausel, die sie referenziert, direkt untergeordnet sein.
- Die Verwendung von BY bzw. USING in Datenstrukturen unterliegt folgenden Einschränkungen:
  - Wenn einem Datenfeld **mehrere** Datenfelder mit ELEMENT-Angabe in der IDENTIFIED-Klausel direkt untergeordnet sind, müssen diese alle die BY-Angabe machen.
  - Wenn einem Datenfeld **mehrere** Datenfelder mit ATTRIBUTE-Angabe in der IDENTIFIED-Klausel direkt untergeordnet sind, müssen diese alle die BY-Angabe machen.

- Wenn einem Datenfeld nur **ein** Datenfeld mit ELEMENT-Angabe in der IDENTIFIED-Klausel direkt untergeordnet ist, darf es entweder die BY- oder die USING-Angabe machen.
- Wenn einem Datenfeld nur **ein** Datenfeld mit ATTRIBUTE-Angabe in der IDENTIFIED-Klausel direkt untergeordnet ist, darf es entweder die BY- oder die USING-Angabe machen.

Beispiel 12-34 Gleiche Beschreibung für unterschiedliche Dokumente

XML-Dokument 1

```
<a
   att="123">
   xxxx
   <b>zz
      <c>9876</c>
   </b>
</a>
```

XML-Dokument 2

```
<a_2
  att="345">
   abc
   <b>@#!??*
      <d>100</d>
   </b>
</a_2>
```

COBOL-Datenstruktur

```
01 wurzel                IDENTIFIED BY wurzel-name.
   02 wurzel-name        PIC XXXX VALUE "a".
   02 wurzel-att         IDENTIFIED BY "att" ATTRIBUTE.
      03 att-wert        PIC 999.
   02 wurzel-wert        PIC X(10).
   02 kind               IDENTIFIED BY "b".
      03 kind-wert       PIC X.
      03 enkel           IDENTIFIED USING enkel-name.
         04 enkel-wert   PIC 9(8) BINARY.
         04 enkel-name   PIC X.
```

Anmerkungen:

- Die Indirektion in der IDENTIFIED-Klausel bei der Datengruppe wurzel, statt des Literals "a" ein Datenfeld anzugeben, das den Wert "a" hat, erlaubt es, mit der gleichen COBOL-Datenstruktur gleich aufgebaute XML-Dokumente zu verarbeiten, bei denen sich der Name des Wurzelelements unterscheidet. Für XML-Dokument 2, beispielsweise, wäre das Datenelement wurzel-name mit "a_2" zu versorgen.
- Die Angabe IDENTIFIED USING beim Datenfeld enkel bedeutet keine Vorgabe für den Elementnamen, d.h. beliebige Namen passen dazu (z.B. das ’c’ aus XML-Dokument 1 bzw. das ’d’ aus XML-Dokument 2) und werden im Datenelement enkel-name zurückgeliefert.
- Die Reihenfolge der Datenfelder für Name und Wert in der Struktur ist beliebig (z.B. wäre auch wurzel-wert vor wurzel-name bzw. enkel-name vor enkel-wert möglich).

Umfang der COBOL-Beschreibung

> ¡ Das XML-Dokument kann und muss nicht komplett eins zu eins in COBOL wiedergegeben werden.

Es reicht, wenn das COBOL-Programm nur diejenigen Teile des XML-Dokuments beschreibt, die es auch verarbeiten will. Dabei ist jedoch eine Anforderung zu erfüllen: zu jedem Knoten aus dem XML-Dokument, der im COBOL Programm beschrieben ist, muss auch dessen Vater im COBOL-Programm beschrieben sein. Die **Wurzel** des XML-Dokuments muss also **immer** im COBOL-Programm beschrieben sein.

Die COBOL-Datenstruktur darf sowohl mehr Knoten beschreiben, als im XML-Dokument aktuell vorhanden sind, als auch weniger Knoten.

Knoten aus dem XML-Dokument können in der COBOL-Beschreibung aus folgenden zwei Gründen entfallen:

- **Untergeordnete Knoten** müssen in der COBOL-Struktur nicht beschrieben werden, wenn die Anwendung einen Knoten – und damit auch alle Knoten in dem Teilbaum, dessen Wurzel der Knoten darstellt – nicht verarbeiten will.
- **Wiederholungen** von Knoten mit gleichem Namen in einer Folge von Element-Knoten dürfen in der COBOL-Struktur nicht beschrieben werden. Dies kann bei Attributen nicht vorkommen, da sie relativ zu ihrem Element eindeutige Namen haben müssen.
  Analogie zu nicht XML-organisierten Dateien: In deren Dateierklärung (FD) hat nicht jeder einzelne Satz aus der Datei seine eigene Beschreibung, sondern nur die unterschiedlichen Satzstrukturen sind in der FD beschrieben. Analog spielen Wiederholungen von Elementen mit gleichem Namen im XML-Dokument für die Beschreibung in COBOL keine Rolle – eine einzige Beschreibung des Elements reicht aus. Analog zur READ-Anweisung, die weitere Sätze liefert, wobei der Zugriff immer über die gleiche Satzbeschreibung erfolgt, liefern die XML-spezifischen Anweisungen auch die Wiederholungen von Elementen, und der Zugriff erfolgt ebenfalls immer nur über die eine Beschreibung des Elements in der COBOL-Struktur.

Beispiel 12-35 Teilweise Beschreibung eines XML-Dokuments

XML-Dokument

```
<a>xxx
   <b>yyy</b>
   <d>123</d>
   <b>zz
      <c>9876</c>
   </b>
 </a>
```

COBOL-Datenstruktur 1

```
01 a             IDENTIFIED BY "a".
   02 a-w        PIC X(10).
   02 b          IDENTIFIED BY "b".
      03         b-w PIC X(10).
      03 c       IDENTIFIED BY "c".
         04 c-w  PIC X(10).
```

COBOL-Datenstruktur 2

```
01 a           IDENTIFIED BY "a".
   02 a-w      PIC X(10).
   02 b        IDENTIFIED BY "b".
      03       b-w PIC X(10).
   02 d        IDENTIFIED BY "d".
      03 d-w   PIC X(10).
```

Anmerkungen:

- COBOL-Datenstruktur 1 erlaubt die Verarbeitung der Wurzel des XML-Dokuments und darunter nur die Teilbäume, deren Wurzel den Namen ’b’ hat – nicht aber den Teilbaum mit Wurzel ’d’.
- Bei der Verarbeitung des XML-Dokuments mit der COBOL-Datenstruktur 1 hat der erste b-Knoten im Dokument keine Kinder, in der COBOL-Struktur ist dafür aber ein Kind beschrieben. Was dies im Detail bedeutet, entnehmen Sie dem Beispiel "Prinzip der Zuordnung von Knoten".
- COBOL-Datenstruktur 2 erlaubt die Verarbeitung der Wurzel des XML-Dokuments und deren Kinder (’b’ und ’d’), aber nicht die Verarbeitung weiterer, ihnen untergeordneter Knoten (’c’).
- Bei der Verarbeitung des XML-Dokuments mit der COBOL-Datenstruktur2 hat der zweite b-Knoten im Dokument Kinder, in der COBOL-Struktur ist dafür aber kein Kind beschrieben. Was dies im Detail bedeutet, entnehmen Sie dem Beispiel "Sequenzielles Lesen" in Kapitel "READ".

COUNT-Klausel

Wenn in der COBOL-Struktur Knoten beschrieben sind, die aktuell im Baum nicht existieren, kann es für die weitere Verarbeitung wichtig sein, zu wissen um welche Datenfelder der Struktur es sich dabei handelt.

Diesem Zweck dient die COUNT-Klausel: Sie legt ein ganzzahliges numerisches Datenelement fest, dessen Inhalt nur die Werte 0 oder 1 annehmen kann.

- Der Wert 1 zeigt an, dass beim Lesen des XML-Dokuments ein Knoten aus dem Baum der Beschreibung in der COBOL-Struktur zugeordnet werden konnte, siehe auch "Zuordnungsvorgang".
- Der Wert 0 zeigt an, dass kein entsprechender Knoten im Baum existiert.

Beispiel 12-36 COUNT-Angabe

```
...
 08 x-knoten   IDENTIFIED BY "x"  COUNT IN x-anzahl.
   09 x-wert   PIC 9(8).
...
```

Anmerkungen:

- Sie dürfen den Namen 'x-anzahl' des Datenelements frei wählen. Der Name muss jedoch **ohne Qualifizierung** im gesamten Programm eindeutig sein. Das Datenfeld mit diesem Namen dürfen Sie jedoch **nicht selbst definieren**.
- Die Definition erfolgt implizit durch die Angabe des Datennamens in der COUNT-Klausel.
- Die COUNT-KLausel darf nur für Datenfelder angegeben werden, die auch eine IDENTIFIED-Klausel haben.
- Das COUNT-Datenelement gibt **nicht** die Anzahl der Wiederholungen von Knoten mit dem Namen 'x' im Baum an.
- Das COUNT-Datenelement hat nichts mit dem Wert eines Knoten zu tun.

12.10.1.3 Definition eines XML-Dokuments in einem COBOL-Programm

Das XML-Dokument, das von einem COBOL-Programm verarbeitet werden soll, kann sowohl in einer Datei, als auch im Speicher (in alphanumerischer oder nationaler Darstellung) bereitgestellt werden – z.B. in einem Datenfeld der LINKAGE-SECTION. In **beiden Fällen** wird es wie eine Datei mit der neuen Dateiorganisation XML behandelt. Der Begriff XML-Datei umfasst daher im folgenden immer diese beiden Möglichkeiten. Folglich sind für ein XML-Dokument analog zu den bisher bereits existierenden Dateien ebenfalls Einträge in der ENVIRONMENTDIVISION und in der DATA DIVISION nötig.

ENVIRONMENT DIVISION

Im FILE-CONTROL-Paragraf benötigt das XML-Dokument einen Dateisteuereintrag:

- Neben der SELECT-Klausel **muss** die ORGANIZATION-Klausel mit der (neuen) Organisation XML angegeben werden.
- In der ASSIGN-Klausel wird die Verbindung zum Medium hergestellt, das das XML-Dokument enthält.
  - Wenn das XML-Dokument in einer Datei steht, wird die gleiche ASSIGN-Klausel, wie bisher für Dateien üblich, verwendet.
  - Wenn das XML-Dokument in einem Speicherbereich steht, können Sie das entsprechende Datenfeld direkt angeben mittels **DATA datenname-1**.

    Alternativ können Sie mittels **datenname-2 LENGTH datenname-3** ein Zeigerdatenfeld angeben, das auf den Speicherbereich zeigt, in dem das XML-Dokument in der durch datenname-3 angegebenen Länge (in Zeichen) steht.
- Die ACCESS MODE-Klausel darf entfallen. Wenn sie jedoch angegeben ist, ist nur die Zugriffsart **XML** zulässig.
- Alle anderen Klauseln sind für ein XML-Dokument verboten mit Ausnahme der FILE STATUS-Klausel. Deren erweiterte Form ist zulässig. Dadurch steht ergänzend ein Returncode des Parsers zur Verfügung.

> ¡ Der Dateisteuereintrag beschreibt für ein XML-Dokument keine Eigenschaften einer konkreten Datei.

Sofern das XML-Dokument also in einer Datei vorliegt, legt der Eintrag kein Dateiformat fest. Vielmehr sind sequenzielle und index-sequenzielle Dateien, POSIX-Dateien und auch Bibliothekselemente möglich. Bei index-sequenziellen Dateien wird der ISAM-Key am Anfang (8 Bytes) entfernt.

Beispiel 12-37 Dateisteuereintrag für XML-Dokument

```
...
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT xml-doc1 ASSIGN TO "LINK-NAM"
                    ORGANIZATION IS XML
                    ACCESS XML.
    SELECT xml-doc2 ASSIGN TO DATA w-s-doc
                    ORGANIZATION IS XML.
    SELECT xml-doc3 ASSIGN TO doc-ptr LENGTH doc-lg FOR NATIONAL
                    ORGANIZATION IS XML.
...
WORKING-STORAGE SECTION.
01 w-s-doc    VALUE "<a att="123">xxxx<b>zz<c>9876</c></b></a>".
01 doc-ptr    USAGE POINTER.
01 doc-lg     PIC 9(8) BINARY.
```

Anmerkungen:

- Das mit xml-doc1 verarbeitete XML-Dokument steht in einer Datei mit dem Linknamen LINK-NAM.
- Das mit xml-doc2 verarbeitete XML-Dokument steht im Speicher im Datenfeld w-s-doc. Seine Länge ist die des Datenfeldes w-s-doc.
- Das mit xml-doc3 verarbeitete XML-Dokument steht in einem Speicherbereich in UTF-16-Darstellung (national). Das Datenfeld doc-ptr zeigt auf diesen Speicherbereich, und die Länge des Dokuments in Zeichen steht im Datenfeld doc-lg. Sie müssen beide Datenfelder vor Beginn der Verarbeitung mit passenden Werten versorgen.

DATA DIVISION

In der FILE SECTION braucht das XML-Dokument eine Dateierklärung:

- Mit Ausnahme der EXTERNAL- und der GLOBAL-Klausel sind alle anderen Klauseln für ein XML-Dokument verboten.
- Die 01-Satzbeschreibungen in der FD beschreiben Teile des XML-Dokuments oder das ganze Dokument. Mindestens eine der Satzbeschreibungen muss die Wurzel des Dokuments beschreiben. Mehrere Satzbeschreibungen sind sinnvoll, wenn mit einer Dateierklärung unterschiedlich strukturierte XML-Dokumente verarbeitet werden sollen, oder aber einzelne Satzbeschreibungen jeweils nur Teile des Dokuments beschreiben, siehe auch "OPEN DOCUMENT". In der Beschreibung für jeden Knoten dürfen Sie einzelne oder auch alle Kinder weglassen – egal ob es sich dabei um Elemente oder Attribute handelt. Wenn es für einen Knoten des Baums kein Datenfeld in der Satzbeschreibung gibt, kann auch kein Knoten aus dem entsprechenden Teilbaum in der Satzbeschreibung vorkommen.
- Knoten aus dem Dokument müssen mit der neuen IDENTIFIED-Klausel beschrieben werden (siehe auch "Angabe eines Element- bzw. Attribut-Namens in der IDENTIFIED-Klausel" in Kapitel "COBOL-Sprachmittel zur Beschreibung eines XML-Dokuments"). Diese Klausel ist nur in Satzbeschreibungen von Dateien mit Organisation XML erlaubt.

> **i** Die Dateierklärung beschreibt keine Eigenschaften einer konkreten Datei sondern nur die logische Struktur des XML-Dokuments.

Die 01-Satzbeschreibungen haben nichts mit der Darstellung der Datei in einem Dateiverwaltungssystem oder Ähnlichem zu tun. Aus diesem Grund redefinieren sich die Satzbeschreibungen in der FD einer XML-Datei **nicht**, so wie es implizit bei sequenzieller, relativer und indizierter Dateiorganisation der Fall ist.

Wenn das XML-Dokument in einer Datei vorliegt, sind beliebige Eigenschaften der Sätze (Satzformat, Satzlänge, Blockung, Zeichencode) möglich.

Beispiel 12-38 Dateierklärung für XML-Dokument

```
...
FILE SECTION.
FD xml-doc2.
01 a               IDENTIFIED BY "a".
   02 a-value      PIC X(10).
   02 att          IDENTIFIED BY "att"
                   ATTRIBUTE PIC 999.
   02 b1           IDENTIFIED BY "b"
                   PIC X(10).
01 b2               IDENTIFIED BY "b".
   02 b-value      PIC X(10).
   02 c            IDENTIFIED BY "c"
                   PIC X(10).
...
```

Anmerkungen

- Die Dateierklärung bezieht sich auf die entsprechende SELECT-Klausel aus Beispiel 12-37.
- Die Satzbeschreibung a beschreibt den Ausschnitt aus dem XML-Dokument, bestehend aus dessen Wurzel ’a’, ihrem Attribut ’att’ und ihren Kindern ’b’. Die Satzbeschreibung b2 beschreibt den Ausschnitt aus dem XML-Dokument bestehend aus Knoten ’b’ und dessen Kindern ’c’.
- Durch die Überlappung der beiden 01-Strukturen beim Knoten ’b’ ist auch der Zugriff auf Enkel des Wurzelknoten ’a’ möglich, obwohl jede einzelne der 01-Strukturen nur Väter und Kinder beschreibt.

12.10.1.4 Anweisungen für die XML-Verarbeitung

Entsprechend der an die Dateiverarbeitung angelehnten Beschreibung von XML-Dokumenten erfolgt auch die Verarbeitung von XML-Dokumenten mit Hilfe der leicht erweiterten Anweisungen für Dateiverarbeitung:

- OPEN DOCUMENT:
  Beginn der Verarbeitung eines XML-(Teil-)Dokuments, sowie Zuordnen einer Satzbeschreibung aus der FD des XML-Dokuments zu einem Dokument in Baumdarstellung
- START ATTRIBUTE bzw. START ELEMENT:
  Positionieren im Baum, ausgehend von einer bereits zugeordneten Satzbeschreibung
- READ ATTRIBUTE bzw. READ ELEMENT:
  Übertragen von Daten aus dem Baum in entsprechende Datenfelder der zugeordneten Satzbeschreibung
- CLOSE DOCUMENT:
  Ende der Verarbeitung eines XML-(Teil-)Dokuments

Die Fehlerbehandlung ist wie bei der Dateiverarbeitung über anweisungsspezifische Angaben (AT END, INVALID KEY) bzw. über USE-Anweisungen möglich. Das Datenfeld der FILE STATUS-Klausel beschreibt das Ergebnis der Anweisung, wobei neben den bisher schon definierten Werten neue XML-spezifische Werte möglich sind.

Bei der Verarbeitung eines XML-Dokuments ist für das Verständnis der Anweisungen wesentlich:

- Knoten aus dem Baum werden den Datenfeldern einer Satzbeschreibung zugeordnet, die eine IDENTIFIED-Klausel haben.
- Die aktuelle Zuordnung wird festgehalten.
- Die aktuelle Zuordnung wirkt auf nachfolgende Anweisungen.
- Die aktuelle Zuordnung kann durch Anweisungen geändert werden.

Element-Positions-Vektor (EPV)

Die Zuordnung von Knoten im Baum zu ihrer Beschreibung im COBOL-Programm wird mit Hilfe des Element-Positions-Vektor festgehalten. Der Name entspricht dem im Technical Report "Native COBOL Syntax for XML Support" verwendeten 'Element Position Vector' – er dient jedoch der Zuordnung **aller** Knoten, Element- und Attribut-Knoten. Es handelt sich dabei um eine interne Verwaltungsstruktur des COBOL-Systems für die Verarbeitung eines XML-Dokuments: Jeder im Programm beschriebene Knoten (d.h. nur Datenfelder mit einer IDENTIFIED-Klausel) hat einen Eintrag im EPV. In diesem Eintrag ist die Position eines aktuell zugeordneten Knotens aus dem Baum gespeichert – dieser Zustand wird im Folgenden auch mit '**das Datenfeld hat eine Position**' bezeichnet.

Wenn einem Datenfeld noch kein Knoten zugeordnet wurde, oder wenn die Zuordnung durch eine Anweisung ungültig geworden ist, wird auch dieser spezielle Zustand im EPV vermerkt. Die Interpretation der gespeicherten Positionen durch nachfolgende Anweisungen hängt außerdem davon ab, ob sie als Resultat einer READ- oder einer OPEN- bzw. START-Anweisung entstanden sind.

Zudem kann ein Datenfeld nur dann eine gültige Position haben, wenn alle ihm in der Struktur übergeordneten Datenfelder auch eine gültige Position haben.

Zuordnen von Knoten

> ¡ Eine Zuordnung modifiziert nur den EPV, überträgt aber keine Daten.

Voraussetzung für das Zuordnen eines Knotens aus dem Baum zu einem Datenfeld

- Die gleiche Art des Knotens im Baum und in der IDENTIFIED-Klausel des Datenfeldes, d.h. beide sind Element-Knoten bzw. beide Attribut-Knoten.

- Die 'Übereinstimmung' des in der IDENTIFIED-Klausel angegebenen Namens mit dem Namen des Knoten im Baum:
  - Bei einer BY-Angabe müssen die Namen identisch sein, abgesehen von endständigen Leerzeichen.
  - Bei einer USING-Angabe wird jeder beliebige Name aus dem Baum als übereinstimmend angesehen.

Zuordnungsvorgang

1. **Höchstens ein** Knoten aus dem Baum wird genau einem Datenfeld mit IDENTIFIED-Klausel zugeordnet. Dabei stellen die Regeln der Sprachmittel sicher, dass es höchsten ein solches Datenfeld geben kann.Die Menge von Knoten und Datenfeldern, die betrachtet werden müssen, hängt von der Anweisung ab.
2. Für jedes Datenfeld, dem ein Knoten aus dem Baum zugeordnet werden konnte wird versucht, **jedem** der Datenfelder mit IDENTIFIED-Klausel, die diesem Datenfeld direkt untergeordnet sind, genau eines der Kinder des zugeordneten Knoten zuzuordnen, beginnend mit dem ältesten, noch nicht zugeordneten Kind.
3. Jedes bei den vorhergehenden Schritten betrachtete Datenfeld, dem kein Knoten zugeordnet werden konnte, erhält die Position 'invalid', ebenso auch alle ihm untergeordneten Datenfelder.

Beispiel 12-39 Prinzip der Zuordnung von Knoten

XML-Dokument	Knoten-Position	COBOL-Datenstruktur	EPV
`<a>`	1	`FD xml-fil`	
`<d>ddd</d>`	2	`01 x IDENTIFIED BY "b".`	inv
`<b>1</b>`	3	`02 y IDENTIFIED BY "c".`	inv
`<b>22`	4	`03 ...`	
`<c>level3</c>`	5	`01 z IDENTIFIED BY "a".`	1
`</b>`		`02 u IDENTIFIED BY "b".`	3
`<c>level2</c>`	6	`03 v IDENTIFIED USING w-name.`	inv
`</a>`		`04 ...`	
		`02 w IDENTIFIED BY "d".`	2
		`03 ...`	

Anmerkungen:

- Datenfelder für Werte spielen bei der Zuordnung keine Rolle. Sie sind daher in der COBOL-Beschreibung weggelassen.
- Annahme für den ersten Schritt: Auf COBOL-Seite werden die Datenfelder auf Stufe 01 (x und z) betrachtet, auf XML-Seite nur der Knoten1 – das entspricht dem Verhalten einer OPEN DOCUMENT-Anweisung, siehe auch "OPEN DOCUMENT". Für die Zuordnung des XML-Dokuments zur COBOL-Beschreibung muss ein Datenfeld gefunden werden, das den Namen des Wurzel-Knotens 'a' in seiner IDENTIFIED-Klausel angibt: das ist das Datenfeld z.
- In den folgenden Schritten wird die Zuordnung anschließend fortgesetzt auf Seite des Baums mit den Kindern des soeben zugeordneten Knoten 1 - beginnend bei dem ältesten, d.h. mit den Knoten 2, 3, 4 und 6. Auf der Seite der COBOL-Beschreibung kommen dafür entsprechend nur die Datenfelder mit einer IDENTIFIED-Klausel in Frage, die dem Datenfeld, dem gerade der Vaterknoten zugeordnet wurde, direkt untergeordnet sind, d.h. die dem Datenfeldz untergeordneten Datenfelder u und w.

- Dem Datenfeldu kann Knoten 3 mit Namen 'b' zugeordnet werden, dem Datenfeld w kann Knoten 2 mit Namen 'd' zugeordnet werden.
- Knoten 4, ebenfalls mit Namen 'b', kann nicht zugeordnet werden: Im Prinzip käme das Datenfeldu dafür in Frage, aber u steht nicht mehr zur Verfügung, da ihm bereits der Knoten 3 zugeordnet wurde. Da Knoten 4 nicht zugeordnet werden kann, kommen auch alle seine Kinder, deren Kinder usw. – hier nur der Knoten 5 – für weitere Zuordnungen nicht in Frage.
- Knoten 6 kann ebenfalls nicht zugeordnet werden, da keines der dem Datenfeld z direkt untergeordneten Datenfelder (u, w) in seiner IDENTIFIED-Klausel den Namen 'c' angibt. Datenfeld y gibt zwar den passende Namen an, gehört aber nicht zu den betrachteten Datenfeldern, weil es kein Kind von Datenfeld z ist.
- Nachdem Datenfeld u der Knoten 3 zugeordnet wurde, kommen auch die dem Datenfeldu direkt untergeordneten Datenfelder (v) für Zuordnungen in Frage. Da aber der zugeordnete Knoten 3 keine Kinder hat, sind auch keine weiteren Zuordnungen möglich.
- Im letzten Schritt bekommen alle während des Verfahrens betrachteten Datenfelder, denen keine Knoten zugeordnet werden konnten, ungültige Positionen: das sind aus dem ersten Schritt das Datenfeld x mit seinem untergeordneten Datenfeldy und aus den folgenden Schritten das Datenfeld v.

Das Verfahren für die Zuordnung von Knoten bedeutet:

- Die Reihenfolge von zugeordneten Geschwister-Knoten im Baum muss nicht mit der Reihenfolge übereinstimmen, in der sie in der COBOL-Datenstruktur beschrieben sind.
- Die Hierarchie der Datenfelder in der COBOL-Beschreibung entspricht direkt der Hierarchie der zugeordneten Knoten im Baum. Es gibt keine Lücken. Für die Knoten bedeutet dies: Wenn ein Knoten aus dem Baum einem Datenfeld zugeordnet ist, ist auch sein Vater einem Datenfeld der COBOL-Struktur zugeordnet.
- Bei mehreren 01-Strukturen in einer FD kann es immer nur in einer einzigen 01-Struktur gültige Positionen geben.
- Ziel ist es, nicht so viele Zuordnungen wie möglich zu Stande zu bringen, sondern die Zuordnungen immer in einer gleichen festen Reihenfolge vorzunehmen.

Anweisungssequenzen

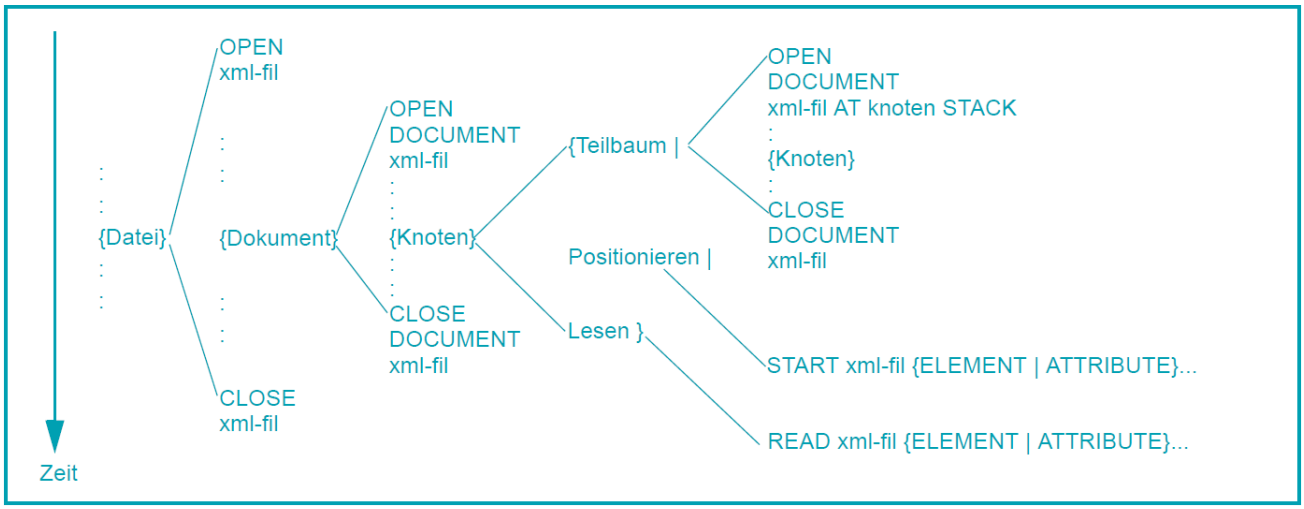
In den folgenden Abschnitten werden die wesentlichen Wirkungen der Anweisungen kurz dargestellt, jedoch ohne Details der Syntax und aller Regeln.

Für jede Themengruppe wird zunächst ein Mindestmaß an Information angeboten, das aufzeigt, wie die Anweisungen prinzipiell funktionieren. Dies wird dann anhand von Beispielen mit Anmerkungen veranschaulicht. Abschließend werden pro Themengruppe wichtige Informationen zusammengefasst, jedoch ohne Anspruch auf Vollständigkeit. Um alle Informationen zu einem Thema gebündelt nachzulesen, sichten Sie die entsprechenden Abschnitte im Kapitel "XML" .

Die Anweisungen zur Verarbeitung des XML-Dokuments bzw. einzelner Teile davon müssen die in der nachfolgenden Abbildung dargestellten Reihenfolgen einhalten. Weitere COBOL-Anweisungen zur eigentlichen Verarbeitung der XML-Daten sind der Übersichtlichkeit halber weggelassen.

{} kennzeichnen wiederholbare Einzelschritte

| trennt mögliche Alternativen



Der Übersichtlichkeit halber verzichtet die Darstellung der neuen Anweisungen weitgehend auf mögliche Fehlersituationen und deren Behandlung. Der Überblick zu Fehlerfällen folgt in Abschnitt "Fehlerbehandlung".

Wichtige Hinweise zur Darstellung der nachfolgenden Beispiele 12-40 bis 12-56:

Von links nach rechts wird in den Spalten der Beispieltabellen jeweils Folgendes dargestellt:

- XML-Dokument:

  Die XML-Dokumente sind auf das für den Beispielzweck Wesentliche reduziert. Sie sollten nicht als wohlgeformtes XML missverstanden werden.

  Zusätzlich sind die XML-Dokumente der besseren Lesbarkeit halber mit Einrückungen/Leerzeichen und Zeilenwechseln versehen. Diese Editierungen sind nicht als Teil der Werte eines Knotens gemeint.
- Positionen, die den XML-Knoten entsprechen
- COBOL-Beschreibung
- Anschließend folgen spaltenweise die nacheinander ausgeführten Anweisungen. Dabei ist jeweils der Zustand nach Ausführung der Anweisung festgehalten: Ein im EPV zugeordneter Knoten (d.h. seine Position) ist dazu bei der IDENTIFIED-Klausel vermerkt. In Klammern wird angefügt, durch welche Anweisung die Position entstanden ist:

  o OPEN DOCUMENT, START

  r READ

  Alle als Folge der Anweisung modifizierten Positionen und Inhalte sind in der zugehörigen Spalte fett dargestellt, alle unveränderten Werte kursiv.

  Für den Einzelfall unwichtige Anweisungen und Definitionen sind weggelassen.

12.10.1.5 OPEN, CLOSE

Die Verarbeitung eines XML-Dokuments muss **immer** mit den bei der Dateibearbeitung üblichen Anweisungen OPEN und CLOSE beginnen und enden, unabhängig davon, ob das Dokument in einer Datei oder im Speicher vorliegt.

OPEN erlaubt für XML-Dateien nur den Öffnungsmodus INPUT. CLOSE erlaubt keine der zusätzlichen Angaben.

Für Dokumente in einer Datei wirken diese Anweisungen analog denen für Nicht-XML-Dateien. Für Dokumente, die im Speicher vorliegen, wird bei der OPEN-Anweisung die Verbindung zum Speicher hergestellt, bzw. ein Zeigerdatenfeld und die Länge aus der ASSIGN-Klausel werden ausgewertet.

Die bei der bisherigen Dateiverarbeitung möglichen Werte für den Ein-/Ausgabe-Zustand sind bei beiden Arten von Dokumenten zu erwarten.

12.10.1.6 OPEN DOCUMENT

Das zweite Format der OPEN-Anweisung (ausschließlich für XML-Dateien) mit der Angabe DOCUMENT leitet die eigentliche Verarbeitung eines XML-Dokuments ein. Es hat folgende drei Aufgaben:

- Beschaffen von Arbeitsspeicher, der die Baumdarstellung des Dokuments aufnimmt.
- Erzeugen der dem XML-Dokument entsprechenden Baumdarstellung in diesem Speicher.
- Ausführen einer ersten Zuordnung des Baums zu einer der 01-Satzbeschreibungen aus der FD.

Der Zuordnungsvorgang beginnt im ersten Schritt auf der COBOL-Seite mit allen Datenfeldern auf Stufe 01 in der FD. Auf XML-Seite beginnt er mit der Wurzel des XML-Dokument-Baums. Anschließend werden weitere Zuordnungen durchgeführt, wie unter "Zuordnungsvorgang" beschrieben.

Danach sind Positionierungen (START) bzw. das Lesen (READ) über erfolgreich positionierte Datenfelder möglich.

OPEN DOCUMENT erlaubt zusätzlich auch die Angabe von
AT datenname-1 [STACK] (dabei muss datenname-1 mit einer IDENTIFIED-Klausel für Element-Knoten definiert sein). Im Unterschied zur Anweisung ohne die AT-Angabe, die ein ganzes XML-Dokument zur Verarbeitung bereitstellt, wird durch die AT-Angabe die (weitere) Verarbeitung auf einen Teilbaum dieses Dokuments begrenzt. Die Wurzel des Teilbaums muss jedoch (durch vorhergehende Anweisungen) erreichbar geworden sein, d.h. der angegebene datenname-1 muss eine gültige Position haben.

Der Zuordnungsvorgang beginnt im ersten Schritt auf der COBOL-Seite mit allen Datenfeldern auf Stufe 01 in der FD, auf XML-Seite mit der Wurzel eines Teilbaums. Als Wurzel dient der dem datenname-1 zugeordnete Knoten. Anschließend werden weitere Zuordnungen durchgeführt, wie unter "Zuordnungsvorgang" beschrieben.

Die Angabe STACK bewirkt, dass der Zustand des EPV vor Ausführung der OPEN-Anweisung gesichert wird und sich später mittels einer CLOSE DOCUMENT-Anweisung wieder herstellen lässt. Ohne die STACK-Angabe geht der 'alte' EPV-Zustand verloren.

Beispiel 12-40 OPEN DOCUMENT

XML-Dokument	Knoten-Position	COBOL-Datenstruktur	`OPEN DOCUMENT xml-fil`	`OPEN DOCUMENT xml-fil AT y`
`<a>`	1	`FD xml-fil`		
`<d>ddd</d>`	2	`01 x IDENTIFIED BY "a".`	1(o)	inv
`<b>22`	3	`02 y IDENTIFIED BY "b".`	3(o)	inv
`<c>level3</c>`	4	`03 ...`		
`</b>`		`01 z IDENTIFIED BY "b".`	inv	3(o)
`<c>level2</c>`	6	`02 u IDENTIFIED BY "c".`	inv	4(o)
`</a>`		`03 ...`		

Anmerkungen

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Nach der ersten OPEN DOCUMENT-Anweisung hat das Datenfeld y eine gültige Position und kann daher in der AT-Angabe bei der zweiten OPEN DOCUMENT-Anweisung verwendet werden.

- Der Zuordnungsvorgang bei der zweiten OPEN DOCUMENT-Anweisung bezieht alle 01-Strukturen mit ein, betrachtet aber nur den Teilbaum mit Knoten 3 als Wurzel.Dies bewirkt ungültige Positionen für die Datenfelder x und y.
- Da keine STACK-Angabe gemacht wurde, sind für weitere Zugriffe nur die gültigen Positionen im EPV verfügbar. D.h. die Wurzel des XML-Dokuments (Knoten 1) ist durch nachfolgende Anweisung nicht mehr erreichbar, es sei denn, das Dokument und die Datei werden geschlossen und erneut geöffnet.

Zusammenfassung

Zusammenfassend ist bei der OPEN DOCUMENT-Anweisung Folgendes zu beachten:

- Der Zugriff auf ein XML-Dokument beginnt immer bei dessen Wurzel. Daher muss als erstes immer eine OPEN DOCUMENT-Anweisung **ohne** AT-Angabe erfolgen. Selbst die Angabe der Wurzel als Bezeichner in der AT-Angabe funktioniert nicht, denn diese setzt eine gültige Position des angegebenen Datenfeldes voraus.
- OPEN DOCUMENT mit AT, aber ohne STACK-Angabe entspricht einer Bewegung im Baum von der Wurzel in Richtung Blätter ohne Umkehrmöglichkeit.
- Die COBOL-Satzstrukturen sind eigentlich nur Schablonen, die Ausschnitte aus dem Baum beschreiben. Ausgehend von der Wurzel des Baums können sie mit beliebigen Teilen des Baums zur Deckung gebracht werden, solange sie sich zur Laufzeit in mindestens einem Knoten überlappen, dem Datenfeld aus der AT-Angabe.
- Um dasselbe Dokument ein zweites Mal zu öffnen, reicht eine CLOSE DOCUMENT-Anweisung allein nicht aus. Zu diesem Zweck muss auch die Datei geschlossen und wieder geöffnet werden.
- **OPEN DOCUMENT modifiziert nur den EPV, lässt Daten aber unverändert**.

12.10.1.7 CLOSE DOCUMENT

Die Funktionsweise hängt von der letzten OPEN DOCUMENT-Anweisung der Datei ab, für die es noch keine CLOSE DOCUMENT-Anweisung gegeben hat:

- Wenn die letzte OPEN DOCUMENT-Anweisung keine STACK-Angabe hatte, werden alle Einträge im EPV auf ungültig gesetzt, die interne Baumdarstellung des Dokuments wird entsorgt, und der dafür verwendete Arbeitsspeicher wird freigegeben
- Wenn die letzte OPEN DOCUMENT-Anweisung eine STACK-Angabe hatte, wird der EPV in den Zustand zurückversetzt, den er vor der Ausführung dieser OPEN-Anweisung hatte.
- **CLOSE DOCUMENT modifiziert nur den EPV, lässt Daten aber unverändert.**

Eine CLOSE DOCUMENT-Anweisung muss nicht explizit gegeben werden: Wenn sie fehlt, wird sie implizit durch eine CLOSE-Anweisung oder eine OPEN DOCUMENT-Anweisung ohne STACK-Angabe angestoßen (jeweils für die gleiche Datei).

Beispiel 12-41 CLOSE DOCUMENT und OPEN DOCUMENT mit STACK

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	OPEN DOCUMENT xml-fil AT y STACK	CLOSE DOCUMENT	CLOSE DOCUMENT
<a>	1	FD xml-fil				
<d>ddd</d>	2	01 x IDENTIFIED BY "a".	1(o)	inv	1(o)	inv
22	3		3(o)	inv	3(o)	inv
<c>level3</c>	4	02 y IDENTIFIED BY "b".				
	6	03 ...	inv	3(o)	inv	inv
		01 z IDENTIFIED BY "b".	inv	4(o)	inv	inv
<c>level2</c>		02 u IDENTIFIED BY "c".				
		03 ...				

Anmerkungen

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Die STACK-Angabe bei der OPEN DOCUMENT-Anweisung hat unmittelbar keine Auswirkung auf den aktuellen Inhalt des EPV nach dieser Anweisung (vgl. Beispiel "OPEN DOCUMENT").
- Die erste CLOSE DOCUMENT-Anweisung stellt den EPV mit dem Zustand vor der zu-gehörigen (zweiten) OPEN DOCUMENT-Anweisung wieder her, weil dort die STACK-Angabe gemacht wurde.
- Die zur zweiten CLOSE DOCUMENT-Anweisung gehörige (erste) OPEN DOCUMENT-Anweisung hatte keine STACK-Angabe. Daher werden alle EPV-Einträge auf 'ungültig' gesetzt.

12.10.1.8 READ

Das neue, dritte Format der READ-Anweisung nur für XML-Dateien dient dem Lesen eines oder mehrerer Knoten aus dem XML-Dokument-Baum. Das Lesen eines XML-Dokuments unterscheidet sich vom bisherigen Lesen einer Datei in mehrfacher Hinsicht:

1. Die feste, bekannte Datenmenge 'Satz', die eine bisherige READ-Anweisung überträgt, findet keine Entsprechung bei XML-Dokumenten. Vielmehr überträgt die neue READ-Anweisung Daten zu einer ggf. von Anweisung zu Anweisung variierenden Menge von Knoten des Baums (Element- und/oder Attribut-Knoten).
2. Der feste, implizite Wirkungsbereich 'nächster Satz' bei einer bisherigen, sequenziellen bzw. 'ganze Datei' bei einer bisherigen, wahlfreien READ-Anweisung findet keine Entsprechung bei XML-Dokumenten. Vielmehr muss der **Wirkungsbereich** der neuen READ-Anweisung – ein oder mehrere Teilbäume des Dokuments – explizit in der Anweisung angegeben werden.
3. Die bei einer bisherigen READ-Anweisung möglichen Zugriffsarten 'sequenziell' bzw. 'wahlfrei' finden keine Entsprechung bei XML-Dokumenten. Vielmehr ist bei der neuen READ-Anweisung die Zugriffsart für jeden Knoten statisch in den 01-Strukturen der FD festgelegt.
4. Der eine Schlüssel (primary oder alternate) für den wahlfreien Zugriff bei einer bisherigen READ-Anweisung findet keine Entsprechung bei XML-Dokumenten. Vielmehr definieren die 01-Strukturen die Schlüssel, und die einzelne neue READ-Anweisung gibt an, welche Teilmenge dieser Schlüssel jeweils zu verwenden ist.
5. Die neue READ-Anweisung behandelt Element-Knoten bzw. Attribut-Knoten beim Lesen unterschiedlich. Entsprechendes findet man bei der bisherigen READ-Anweisung nicht.

Diese aufgeführten Unterschiede spiegeln sich in den erweiterten 01-Datenstrukturen in der FD und der von der bisherigen READ-Anweisung abweichenden Syntax und Semantik der neuen READ-Anweisung wieder:

- Es wird in der Regel nur ein Teil der in einer 01-Datenstruktur beschriebenen Daten versorgt – es gibt keine INTO-Angabe (siehe auch 1.). In der Satzbeschreibung kann es Datenfelder geben, die durch READ-Anweisungen nie geändert werden, z.B. das in IDENTIFIED BY angegebene Datenfeld.
- Der Wirkungsbereich (siehe auch 2.) wird in Form eines Datenfeldes bei der neuen READ-Anweisung angegeben. Diesem Datenfeld muss ein Knoten des Baums zugeordnet sein. Dieser Knoten bestimmt den Wirkungsbereich:
  - Wenn die Zuordnung durch eine OPEN- oder START-Anweisung erfolgte, umfasst er den durch diesen Knoten festgelegten Teilbaum und die durch seine jüngeren Geschwister festgelegten Teilbäume.
  - Wenn die Zuordnung durch eine READ-Anweisung erfolgte, umfasst er nur die durch die jüngeren Geschwister festgelegten Teilbäume
- Die IDENTIFIED-Klauseln in den Satzbeschreibungen der FD legen die Zugriffsart (siehe auch 3.) fest:
  - BY: wahlfreier Zugriff
  - USING: sequenzieller Zugriff, ohne die Möglichkeit, rückwärts zu lesen; es gibt keine PREVIOUS-Angabe.
- Der Wechsel von wahlfreiem zu sequenziellem Lesen (siehe auch 3.) ist nicht möglich – es gibt keine NEXT-Angabe. Das Lesen von weiteren Element-Knoten mit gleichem Namen (d.h. die oben erwähnten Wiederholungen) ist auch wahlfrei möglich, da ein ge-rade gelesener Knoten nicht mehr zum Wirkungsbereich einer nachfolgenden READ-Anweisung mit unverändertem Schlüssel gehört.
- Das bei der neuen READ-Anweisung anzugebende Datenfeld legt gleichzeitig auch die Teilmenge der für den wahlfreien Zugriff zu verwendenden Schlüssel fest (siehe auch 4.). Es gibt keine KEY-Angabe. Für das angegebene Datenfeld und alle ihm untergeordneten Datenfelder gilt:
  - Die mit IDENTIFIED BY gemachten Angaben beschreiben die Schlüssel für wahlfreien Zugriff.
  - Die mit IDENTIFIED USING gemachten Angaben beschreiben die Datenfelder, in denen der Name des sequenziell gelesenen Element- bzw. Attribut-Knoten zur Verfügung gestellt wird.

- Die (kontextsensitiven) Schlüsselwörter ATTRIBUTE bzw. ELEMENT geben die Art des von der READ-Anweisung zu bearbeitenden Knotens an (siehe auch 5.).

Eine neue READ-Anweisung sieht kurz folgendermaßen aus:

```
READ xml-file ELEMENT datenname-1
```

oder

```
READ xml-file ATTRIBUTE datenname-1
```

Dabei ist datenname-1 der Name eines Datenfeldes mit IDENTIFIED-Klausel, also mit der Beschreibung eines Knotens in der COBOL-Struktur (nicht zu verwechseln mit dem in der IDENTIFIED-Klausel ggf. angegebenen Datenfeld, das den Namen des Knotens enthält). Die dort gemachte Angabe ELEMENT bzw. ATTRIBUT muss mit derjenigen in der READ-Anweisung übereinstimmen.

Das angegebene Datenfeld datenname-1 ist entscheidend für die neue READ-Anweisung. Daher ist in diesem Zusammenhang nicht mehr vom 'Lesen der Datei' die Rede, sondern vom ’Lesen über (Datenfeld) datenname-1’.

Voraussetzung für die erfolgreiche Ausführung der READ-Anweisung ist, dass bereits die Zuordnung eines Knotens aus dem Baum zu dem angegebenen datenname-1 existiert. Bei der Ausführung der READ-Anweisung werden dann

- (einigen) Datenfeldern in einer 01-Struktur der FD Knoten zugeordnet,
- zu einzelnen Knoten Daten aus dem Baum in die COBOL-Datenstruktur übertragen.

Zuordnung

Der Zuordnungsvorgang beginnt im ersten Schritt auf der COBOL-Seite mit dem Datenfeld, das in der READ-Anweisung angegeben wurde.

- Wenn das in der READ-Anweisung angegebene Datenfeld eine IDENTIFIED BY-Klausel mit ATTRIBUTE-Angabe hat, werden der zugeordnete Attribut-Knoten und **alle** seine Geschwister auf eine mögliche Zuordnung hin überprüft.
- Wenn das in der READ-Anweisung angegebene Datenfeld eine IDENTIFIED USING-Klausel mit ATTRIBUTE-Angabe hat, oder eine IDENTIFIED USING/BY-Klausel mit ELEMENT-Angabe, werden auf XML-Seite die Wurzel-Knoten der Teilbäume aus dem Wirkungsbereich (wie oben definiert) auf eine mögliche Zuordnung hin überprüft, in der Reihenfolge von den älteren zu den jüngeren.

Anschließend werden weitere Zuordnungen durchgeführt, wie unter "Zuordnungsvorgang" beschrieben.

i
- Datenfelder in der Struktur über dem angegebenen Datenfeld, auf gleicher Stufe daneben oder in andern 01-Strukturen der FD werden nicht betrachtet, und ihre Zuordnung im EPV bleibt unverändert.
- Bereits existierende Positionen für Datenfelder, die dem in der READ-Anweisung angegebenen Datenfeld untergeordnet sind, spielen für die Zuordnung keine Rolle.
- Es scheint eigentlich überflüssig, das Datenfeld datenname-1, für das als Voraussetzung schon ein zugeordneter Knoten verlangt wurde, erneut in das Zuordnungsverfahren mit einzubeziehen. Diese Zuordnung muss jedoch aktuell gar nicht mehr zutreffen, wenn sich etwa bei wahlfreiem Zugriff inzwischen der Wert eines Schlüssels geändert hat.
- Da Attributnamen eindeutig sein müssen, spielt die Reihenfolge der Attribut-Knoten keine Rolle. Es kann maximal einen passenden Namen geben. Daher werden in diesem Fall auch immer alle Attribut-Knoten auf mögliche Zuordnung überprüft.

Datenübertragung

Der in der READ-Anweisung angegebene datenname-1 teilt die in der FD definierten Datenfelder in drei Mengen: den datenname-1 selbst, die ihm untergeordneten Datenfelder und alle anderen Datenfelder aus der FD.

Abhängig von der erfolgreichen Zuordnung eines Knotens zu datenname-1, der Zugehörigkeit zu einer der drei Mengen und der Zuordnung von Knoten zu den untergeordneten Datenfeldern, bleiben Datenfelder unverändert, werden initialisiert oder mit Daten aus dem Baum versorgt. Dies verdeutlicht die nachfolgende Tabelle:

	datenname-1 wurde ein Knoten zugeordnet	**datenname-1 wurde kein Knoten zugeordnet**
datenname-1	Datenübertragung	unverändert
untergeordnetes Datenfeld: Knoten zugeordnet	Datenübertragung	
untergeordnetes Datenfeld: kein Knoten zugeordnet	Initialisierung	unverändert
alle anderen Datenfelder	unverändert	unverändert

Für die Datenübertragung sind die Sendefelder im XML-Dokument-Baum mit USAGE NATIONAL (d.h. UTF-16) anzunehmen.

Die Datenübertragung bzw. Initialisierung erfolgt entsprechend den COBOL-Regeln (dazu findet bei Übertragung von Werten aus dem Baum zu alfanumerischen Empfangsfeldern eine Konvertierung entsprechend FUNCTION DISPLAY-OF statt, zu numerischen Empfangsfeldern entsprechend FUNCTION NUMVAL-C, ohne einen zweiten Parameter). Die Datenübertragung bzw. Initialisierung betrifft dabei für einen in der COBOL-Struktur beschriebenen Knoten (d.h. ein Datenfeld mit IDENTIFIED-Klausel) die damit ggf. verbundenen Datenfelder zur Aufnahme des Namens, des Werts und der Anzeige:

	Datenübertragung	**Initialisierung**
Datenfeld aus USING-Angabe in IDENTIFIED-Klausel	Name des Knotens aus dem Baum	Leerzeichen
Datenfeld für den Wert	Wert des Knotens aus dem Baum	initialisiert mittels INITIALIZE...TO DEFAULT
Datenfeld aus COUNT-Klausel	1	0

Beispiel 12-42 Sequenzielles Lesen

XML-Dokument	Pos	COBOL-Datenstruktur	**OPEN DOCUMENT xml-fil**	**READ xml-fil ELEMENT y**	**READ xml-fil ELEMENT y**	**READ xml-fil ELEMENT y**
<dok>000	1	**FD xml-fil**				
<a>111	2	**01 x IDENTIFIED BY "dok".**	**1(o)**	*1(o)*	*1(o)*	*1(o)*
222	3	**02 y IDENTIFIED USING**	**2(o)**	**2(r)**	**3(r)**	**at-end**
<c>333</c>	4	**y-name.**				
				a	**b**	**b**
				111	**222**	**222**

</dok>		03 y-name PIC X. 03 y-wert PIC 999.				
		FILE STATUS	00	00	08	10

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Für das Lesen über y ist die erfolgreiche Zuordnung eines Knotens (Knoten 2) durch die OPEN DOCUMENT-Anweisung ausreichend. Es ist nicht notwendig, dass in der 01-Struktur übergeordnete Datenfelder (x) bereits gelesen wurden.
- Das Lesen über y erfolgt sequenziell, da IDENTIFIED USING angegeben ist.
- Die dem y zugeordnete Position vor der ersten READ-Anweisung wurde durch eine OPEN DOCUMENT-Anweisung gesetzt. Als Wirkungsbereich kommen dementsprechend die Teilbäume mit Wurzeln Knoten2 und 3 (in dieser Reihenfolge) in Frage. Knoten 2 wird zugeordnet sowie Name und Wert des gelesenen Knotens übertragen.
- Einem Datenfeld mit IDENTIFIED-Klausel darf nur ein **einziges** Datenfeld direkt untergeordnet sein, das **nicht** in der IDENTIFIED-Klausel angesprochen wird: In dieses Feld (y-wert) wird der Wert des zugeordneten Knoten übertragen.
- Wenn der Wert eines Knoten für das Programm ohne Bedeutung ist, kann das Datenfeld dafür auch ganz entfallen, wie bei Datenfeld x.-
- Nach der ersten READ-Anweisung ist y weiterhin der Knoten 2 zugeordnet. Die Position vermerkt jedoch, dass sie durch READ entstanden ist. Für die Zuordnung bei der folgenden READ-Anweisung kommen nur jüngere Geschwister, d.h. Knoten 3, in Frage.
- Bei der zweiten READ-Anweisung enthält der zugeordnete Teilbaum einen Knoten (c), der in der entsprechenden COBOL-Struktur y nicht beschrieben ist. Das zeigt der Ein-/Ausgabe-Zustand 08 an.
- Bei der dritten READ-Anweisung hat der zugeordnete Knoten 3 keine jüngeren Geschwister, also entsteht die Ende-Bedingung. Die Datenfelder für Name und Wert bleiben unverändert.

Beispiel 12-43 Wahlfreies Lesen und Wiederholungen

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	READ xml-fil ELEMENT y	READ xml-fil ELEMENT y	READ xml-fil ELEMENT y
<dok>000 <c>111</c> <a>222 <a>333 <c>444</c> </dok>	1 2 3 4 5	FD xml-fil 01 x IDENTIFIED BY "dok". 02 y IDENTIFIED BY y-name. 03 y-name PIC X VALUE "a". 03 y-wert PIC 999.	1(o) 3(o) *a*	*1(o)* 3(r) *a* 222	*1(o)* 4(r) *a* 333	*1(o)* at-end *a* *333*
		FILE STATUS	00	00	00	10

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Das Lesen über y erfolgt wahlfrei, da IDENTIFIED BY angegeben ist. Aktueller Wert des Schlüssels ist 'a'.
- Die dem y zugeordnete Position bei der ersten READ-Anweisung wurde durch eine OPEN DOCUMENT-Anweisung gesetzt. Also kommen als Wirkungsbereich die Teilbäume mit Wurzeln Knoten 3, 4 und 5 in Frage. Als erster davon lässt sich Knoten 3 zuordnen.
- Nach der ersten READ-Anweisung ist y weiterhin der Knoten 3 zugeordnet. Die Position vermerkt jedoch jetzt, dass sie durch READ entstanden ist.
- Der Wirkungsbereich der zweiten READ-Anweisung umfasst die Teilbäume mit Wurzeln Knoten4 und 5. Davon lässt sich Knoten 4 zuordnen. Da Knoten 3 nicht mehr Teil des Wirkungsbereichs ist, liest diese READ-Anweisung die Wiederholung, d.h. das zweite Exemplar des Knotens mit Namen 'a'.
- Bei der dritten READ-Anweisung hat der zugeordnete Knoten4 zwar noch ein jüngeres Geschwister (Knoten 5). Es ist jedoch wegen des nicht übereinstimmenden Namens keine Zuordnung möglich.
- Bei wahlfreiem Lesen entsteht ggf. auch eine **Ende**-Bedingung.

Beispiel 12-44 Wahlfreies Lesen mit wechselndem Schlüsselwert

XML-Dokument	Pos	COBOL-Datenstruktur	**OPEN DOCUMENT xml-fil**	**READ xml-fil ELEMENT y**		**READ xml-fil ELEMENT y**
<dok>000 <c>111</c> <a>222 <a>333 <c>444</c> </dok>	1 2 3 4 5	**FD xml-fil** **01 x IDENTIFIED BY "dok".** **02 y IDENTIFIED BY y-name.** **03 y-name PIC X VALUE "a".** **03 y-wert PIC 999.**	**1(o)** **3(o)** *a*	*1(o)* **3(r)** *a* **222**	**MOVE "c" TO y-name**	*1(o)* **5(r)** *c* **444**

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Bis einschließlich der ersten READ-Anweisung besteht kein Unterschied zum vorhergehenden Beispiel.
- Der Wirkungsbereich der zweiten READ-Anweisung umfasst wie im vorhergehenden Beispiel die Teilbäume mit Wurzeln Knoten 4 und 5. Die Änderung des Werts von Schlüssel y-name zu 'c' bewirkt jetzt die Zuordnung von Knoten 5.
- Da die OPEN DOCUMENT-Anweisung mit dem Schlüsselwert 'a' ausgeführt wurde, kann Knoten 2 trotz des danach modifizierten und eigentlich passenden Schlüsselwerts 'c' nie mehr durch Lesen über y erreicht werde. Das Öffnen hat y den Knoten 3 zugeordnet. Daher kommen für alle weiteren READ-Anweisungen nur mehr dieser Knoten und seine jüngeren Geschwister als Wirkungsbereich in Frage, jedoch nicht das ältere Geschwister Knoten 2.

Beispiel 12-45 Wirkung von wechselnden Schlüsselwerten

XML-Dokument	Pos	COBOL-Datenstruktur	**OPEN DOCUMENT xml-fil**			

					READ xml-fil ELEMENT z	READ xml-fil ELEMENT y
<dok>111 <a>222 <c>333</c> <d>444</d> 555 <c>666</c> <d>777</d> </dok>	1 2 3 4 5 6 7	FD xml-fil 01 x IDENTIFIED BY "dok". 02 x-wert PIC 999. 02 y IDENTIFIED BY y-name. 03 y-name PIC X VALUE "a". 03 y-wert PIC 999. 03 z IDENTIFIED BY z-name. 04 z-name PIC X VALUE "c". 04 z-wert PIC 999.	1(o) 2(o) *a* 3(o) *c*	MOVE "b" TO y-name MOVE "d" TO z-name	*1(o)* *2(o)* *b* 4(r) *d* 444	*1(o)* 5(r) *b* 555 7(r) *d* 777

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Änderungen von Schlüsselwerten (y-name und z-name) nach einer erfolgreichen Zuordnung nach der OPEN DOCUMENT-Anweisung wirken sich nicht immer auf nachfolgende Zuordnungen aus.
- Die beiden MOVE-Anweisungen nach OPEN DOCUMENT legen nahe, dass die erste READ-Anweisung wahlfrei auf den Teilbaum mit Wurzel Knoten 5 (Name = 'b') und darin auf den Knoten 7 (Name = 'd') zugreifen soll: Das Lesen über Datenfeld z ist für diesen Zweck jedoch nicht geeignet.
- Werte von Schlüsseln, die in der COBOL-Struktur dem beim Lesen verwendeten Schlüssel (z mit z-name) **übergeordnet** sind (y mit y-name), werden bei einer Zuordnung **nicht berücksichtigt**. Änderungen solcher Schlüsselwerte bleiben wirkungslos. Im Beispiel erfolgt der Zugriff weiterhin im zugeordneten Teilbaum mit Wurzel 2 und darin auf Knoten 4.
- Neben der Änderung der Schlüsselwerte ist auch über ein Datenfeld zu lesen, das kei-nem der geänderten Schlüssel untergeordnet ist, wie Datenfeld y in der zweiten READ-Anweisung.

Beispiel 12-46 Übertragene Daten

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	READ xml-fil ELEMENT x
<dok>000 <a>22 pqrst <c>99</c> </dok>	1 2 3 4	FD xml-fil 01 x IDENTIFIED BY "dok". 02 x-wert PIX X(5). 02 y IDENTIFIED BY "b".	1(o) 3(o)	1(r) 000'BLANK' 'BLANK' 3(r) pq

		03 y-wert PIC XX. 02 z IDENTIFIED BY "a" COUNT z-count. 03 z-wert PIC 999V99. 02 u IDENTIFIED BY "C" COUNT u-count. 03 u-wert PIC XXX.	2(o) inv	2(r) 1 02200 inv 0 'BLANK' 'BLANK' 'BLANK'

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Beim wahlfreien Lesen spielt die Reihenfolge der Datenfelder in der COBOL-Struktur – zuerst y für Knoten mit Namen 'b', dann z für Knoten mit Namen 'a' – und die der zugeordneten Knoten im Baum – erst der mit Name 'a', dann der mit Name 'b' – keine Rolle.
- Nicht-numerische Werte aus dem Baum, die zu kurz sind, werden mit Leerzeichen aufgefüllt (x-wert), Werte die zu lang sind, werden abgeschnitten (y-wert).
- Numerische Werte werden dezimalpunktgerecht übertragen (z-wert), ggf. auch in eine andere USAGE konvertiert.
- Die COUNT-Datenelemente zeigen an, ob einem Datenfeld Knoten **zugeordnet** werden konnten (z-count) oder nicht (u-count). Nur die READ-Anweisung setzt diese Anzeige.
- Klein-/Großschreibung der Schlüsselwerte spielt eine Rolle. Daher wird Knoten 4 mit Namen 'c' nicht dem Datenfeld u mit Schlüsselwert 'C' zugeordnet.
- Datenfelder, die demjenigen über das gelesen wurde (x) untergeordnet sind und denen kein Knoten zugeordnet wurde (u), bekommen ungültige Positionen im EPV und werden initialisiert.

Beispiel 12-47 Elemente und Attribute

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	READ xml-fil ELEMENT x	READ xml-fil ATTRIBUTE y-att	READ xml-f ELEMENT y
<doc	1	FD xml-fil				
d1="g"	2	01 x IDENTIFIED BY	1(o)	1(r)	*1(r)*	*1(r)*
d2="f"	3	"dok".	3(o)	3(r)	*3(r)*	*3(r)*
>	4	02 x-att1				
<a	5	IDENTIFIED BY "d2"		f	*f*	*f*
a1="	6	ATTRIBUTE.	inv	inv	*inv*	*inv*
1"	7	03 x-att1-wert				
a2="		PIC X.		0	*0*	*0*
2"		02 x-att2		'BLANK'	*'BLANK'*	'BLANK
a3="	8	IDENTIFIED BY "d3"	4(o)	4(r)	*4(r)*	8(r)
3">	9	ATTRIBUTE		a'BLANK''BLANK'	*a'BLANK''BLANK'*	a'BLANK''Bl
ppp	10	COUNT x-	5(o)	5(r)		
		att2-count.			6(r)	9(r)
<a		03 x-att2-wert		a1		
a1="		PIC X.		001	a2	a1
9"		02 y IDENTIFIED		ppp	002	009
a4="		USING y-name.			*ppp*	qqq
8">		03 y-name PIC XXX.				
qqq		03 y-att				

 </doc>		**IDENTIFIED USING y-att-name ATTRIBUTE. 04 y-att-name PIC XX. 04 y-att-wert PIC 999. 03 y-wert PIC XXX.**				
		FILE STATUS	**00**	**08**	**00**	**00**

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Die OPEN DOCUMENT-Anweisung ordnet gleichzeitig sowohl Element-Knoten, als auch Attribut-Knoten zu.
- Bei vorgegebenen Schlüsseln wird dem Attribut x-att1 (Schlüssel='d2') der Knoten 3 zugeordnet. Dem Attribut x-att2 (Schlüssel='d3') kann kein Knoten zugeordnet werden. Ohne vorgegebene Schlüssel wird dem Attribut y-att der erste Attribut-Knoten zugeordnet (Knoten 5).
- Die COUNT-Klausel ist auch für Attribute möglich (x-att2).
- Das nicht in der COBOL-Struktur beschriebene Attribut d2 bewirkt beim Lesen über x den Ein-/Ausgabe-Zustand 08.
- Auch für das sequenzielle Lesen von Attributen gilt die Regelung mit dem Wirkungsbereich, selbst wenn dabei die Teilbäume nur aus einem Knoten bestehen. Der Wirkungsbereich der zweiten READ-Anweisung umfasst die Knoten 6 und 7. Da bei sequenziellem Zugriff jeder Name passt, wird also Knoten 6 zugeordnet.
- Es ist nicht notwendig, über ein Datenfeld sequenziell bis zum Ende zu lesen, z.B. auch noch 'a3' über y-att. Es darf bereits vorher zu einem anderen Datenelement gewechselt werden (y).
- Die dritte READ-Anweisung (über y) ordnet y den Knoten 8 zu und dem untergeordneten Attribut (y-att) das erste Attribut 'a1' dieses Knotens (Knoten9).
- Ein erreichtes Ende beim sequenziellen Lesen wird im EPV entsprechend vermerkt (für Datenfeld y). In diesem Fall werden die EPV-Einträge von allen untergeordneten Datenfeldern auf ungültig gesetzt (y-att). Die Werte der Datenfelder bleiben jedoch unverändert (y-name, y-wert, y-att-name, y-att-wert).
- Die Angabe von BY (x-att1) und USING (y) für Datenfelder mit der selben unmittelbar übergeordneten Datengruppe (x) ist zulässig, weil sie unterschiedliche Arten von Knoten beschreiben (Attribut bzw. Element).-

Beispiel 12-48 Wahlfreies Lesen von Attributen

XML-Dokument	Pos	COBOL-Datenstruktur	**OPEN DOCUMENT xml-fil**	READ xml-fil ATTRIBUTE y-att READ	READ xml-fil ATTRIBUTE y-att		READ xml-fil ATTRIBUTE y-att
<dok <a a1="1" a2="2"	1 2 3 4 5	**FD xml-fil 01 x IDENTIFIED BY "dok". 02 y IDENTIFIED BY "a".**	**1(o) 2(o) 4(o)**	*1(o)* *2(o)* **4(r)**	*1(o)* *2(o)* **4(r)**		*1(o)* *2(o)* **3(r)**

a3=" 3"> ppp </dok>		03 y-att IDENTIFIED BY y-att-name ATTRIBUTE. 04 att-name PIC XX VALUE "a2". 04 att-wert PIC 999.	*a2*	*a2* 002	*a2* 002	MOVE "a1" TO att- name	*a1* 001

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Die erste READ-Anweisung ordnet y-att das Attribut Knoten 4 mit dem vorgegebenen Schlüssel 'a2' zu.
- Die zweite READ-Anweisung zum wahlfreien Lesen eines Attributs untersucht – wegen des unveränderten Schlüssels – daher die Knoten 3, 4 und 5 auf mögliche Zuordnung, d.h. den schon zugeordneten Knoten und alle seine Geschwister: Sie ordnet erneut Knoten 4 zu.
- Da beim wahlfreien Lesen von Attributen (im Gegensatz zur ELEMENT-Angabe, vgl. Beispiel „Wahlfreies Lesen und Wiederholungen") alle Knoten betrachtet werden, kann bei unverändertem (und im Baum vorhandenem) Schlüssel nie die EndeBedingung auftreten.
- Da beim wahlfreien Lesen von Attribut-Knoten **alle** Geschwister-Knoten betrachtet werden, lassen sich durch Änderung des Schlüsselwerts (zu 'a1') mit der dritten READ-Anweisung auch ältere Geschwister (Knoten 3) des dem Datenfeld y-att zugeordneten Knotens (Knoten 4) lesen.

Beispiel 12-49 Mehrfaches Lesen des gleichen Knotens

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	READ xml-fil ELEMENT z	READ xml-fil ELEMENT z	READ xml-fil ELEMENT y
<dok>111 <a>222 <c>333 </c> <d>444 </d> 555 </dok>	1 2 3 4 5	FD xml-fil 01 x IDENTIFIED BY "dok". 02 x-wert PIC 999. 02 y IDENTIFIED USING y-name. 03 y-name PIC X. 03 y-wert PIC 999. 03 z IDENTIFIED USING z-name. 04 z-name PIC X. 04 z-wert PIC 999.	1(o) 2(o) 3(o)	*1(o)* *2(o)* 3(r) c 333	*1(o)* *2(o)* 4(r) d 444	*1(o)* 2(r) a 222 3(r) c 333

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".

- Daten zu Knoten 3 werden durch das Lesen über z geliefert. Die zweite READ-Anweisung setzt die Position von z auf Knoten 4.
- Für das anschließende Lesen über das übergeordnete Datenfeld y spielt nur die dem y selber zugeordnete Position (Knoten 2) eine Rolle. Die Positionen von untergeordneten Datenfeldern (z) sind ohne Bedeutung.
- Entsprechend dem Zuordnungsverfahren ordnet die dritte READ-Anweisung dem z dann erneut den Knoten3 zu und liefert auch dessen Daten (nicht die von Knoten 5).

Eingeschränkte Datenübertragung mit ONLY

READ erlaubt optional die Angabe von **ONLY** vor dem Schlüsselwort ELEMENT: In diesem Fall werden nur Daten in Verbindung mit dem in der READ-Anweisung angegebenen Datenfeld übertragen, einschließlich ggf. vorhandener Attribute. Für alle anderen untergeordneten Datenfelder unterbleibt eine Datenübertragung. Die Zuordnung von Knoten erfolgt jedoch auch für die untergeordneten Datenfelder. Da für die untergeordneten Datenfelder noch keine Daten übertragen wurden, ist ihr EPV-Eintrag bei späteren lesenden Zugriffen so zu interpretieren, als ob er durch eine OPEN DOCUMENT-Anweisung entstanden wäre.

Für das Lesen von Attributen ist ONLY verboten und macht auch keinen Sinn, da AttributKnoten keine Kinder haben.

Beispiel 12-50 Eingeschränkte Datenübertragung mit ONLY

XML-Dokument	Pos	COBOL-Datenstruktur	**OPEN DOCUMENT xml-fil**	**READ xml-fil ELEMENT y**	**READ xml-fil ELEMENT z**	**READ xml-fil ONLY ELEMENT y**	**READ xml-fil ELEMENT z**
<dok>1 2 4 <c>5</c> 6 8 <c>9</c> </dok>	1 2 3 4 5 6 7 8 9	**FD xml-fil** **01 x IDENTIFIED** **BY "dok".** **02 x-wert PIC 9.** **02 y IDENTIFIED** **USING y-name.** **03 y-name PIC X.** **03 y-wert PIC 9.** **03 z IDENTIFIED** **USING z-name.** **04 z-name PIC X.** **04 z-wert PIC 9.** **03 y-att IDENTIFIED** **BY "q" ATTRIBUTE.** **04 y-att-wert PIC X.**	**1(o)** **2(o)** **4(o)** **3(o)**	*1(o)* **2(r)** **a** **2** **4(r)** **b** **4** **3(r)** **A**	*1(o)* *2(r)* *a* *2* **5(r)** **c** **5** *3(r)* *A*	*1(o)* **6(r)** **a** **6** **8(o)** *c* *5* **7(r)** **B**	*1(o)* *6(r)* *a* *6* **8(r)** **b** **8** *7(r)* *B*

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Die erste READ-Anweisung liest den Knoten 2 mit seinem Attribut Knoten 3, sowie den untergeordneten Knoten 4. Die nachfolgende READ-Anweisung über das untergeordnete Datenfeld z liest sequenziell den nächsten Knoten 5.
- Zum Vergleich das Lesen im analog strukturierten Teilbaum mit Wurzel 6, jedoch mit ONLY-Angabe, d.h. ohne Übertragung der untergeordneten Elemente: Für das Datenfeld y aus der READ-Anweisung werden der Name und Wert des Elements übertragen ('a' und '6'). Für die ihm untergeordneten Datenfelder, die Attribut-Knoten beschreiben (y-att), erfolgt ebenfalls die Datenübertragung ('B').
- Für die dem y untergeordneten Datenfelder, die Element-Knoten (hier nur z) beschreiben, werden zwar Knoten zugeordnet (Knoten 8), aber keine Daten übertragen. In der COBOL-Struktur vorhandene Daten bleiben unverändert. Die Zuordnung im EPV vermerkt, dass sie **nicht** durch READ erfolgte.
- Die anschließende vierte READ-Anweisung über das untergeordnete Datenfeld (z) überträgt jetzt erst Name und Wert des zugeordneten Knoten 8, nicht wie die zweite READ-Anweisung Name und Wert des folgenden Knoten.

Zusammenfassung

Zusammenfassend ist bei der READ-Anweisung Folgendes zu beachten:

- Wahlfreier lesender Zugriff auf Elemente erlaubt immer nur einen **Zugriff auf Kinder und jüngere Geschwister**, dagegen **nie auf Väter oder ältere Geschwister**.
- Wahlfreies Lesen, ohne zwischenzeitliche OPEN- bzw. START-Anweisung oder Änderung des Schlüssels, liefert bei Element-Knoten die Wiederholungen und nach dem letzten Exemplar die Ende-Bedingung. Bei Attribut-Knoten wird immer wieder das gleiche (einzige) Exemplar geliefert.
- Durch eingeschobene, untergeordnete Elemente **aufgesplitterte Werte** werden zu einem einzigen Stück **zusammengefasst** geliefert.
- Eine READ-Anweisung ohne ONLY-Angabe liefert für den Ein-/Ausgabe-Zustand den Wert 08, wenn es in dem für die Zuordnung verwendeten Teilbaum mindestens einen Knoten unterhalb der Wurzel gibt, der nie einem Datenfeld in der COBOL-Struktur zugeordnet werden kann.
- **Es geht hier um die prinzipielle Möglichkeit, den Knoten einem Datenfeld zuordnen zu können. Das hat nichts damit zu tun, ob bei der aktuellen READ-Anweisung diese Zuordnung erfolgt ist und Daten übertragen wurden.** Insbesondere bedeutet das: Wiederholungen von Elementnamen im Baum, von denen ein Exemplar zugeordnet werden konnte, bewirken keinen Wert 08 für den Ein-/Ausgabe-Zustand.
- Bei einer READ-Anweisung mit ONLY-Angabe kann es den Ein-/Ausgabe-Zustand 08 nur für nicht zuordenbare Attribute des gelesenen Elements geben, nicht jedoch für andere Knoten des betrachteten Teilbaums.
- Aktuelle Positionen bzw. zugeordnete Knoten für Datenfelder, die dem in der READ-Anweisung angegebenen Datenfeld in der COBOL-Struktur untergeordnet sind, spielen keine Rolle.
- Das in der READ-Anweisung angegebene Datenfeld bestimmt gleichzeitig, **wo** gesucht/gelesen werden soll und **was** gesucht/gelesen werden soll.
- Eine READ-Anweisung ist meist eine Mischung aus sequenziellem und wahlfreiem Zugriff, wobei auf alle Element-Kinder bzw. Attribut-Kinder eines Knotens auf die gleiche Art zugegriffen wird, unabhängig davon, auf welche Art auf den Knoten selbst zugegriffen wird.
- Eine READ-Anweisung ändert im EPV nur die Position des Datenfeldes, über das gelesen wird, und der ihm untergeordneten Datenfelder. Die Positionen aller anderen Datenfelder bleiben unverändert.

12.10.1.9 START

Das neue zweite Format der START-Anweisung nur für XML-Dateien dient dem Positionieren auf einen oder mehrere Knoten (Elemente und/oder Attribute) aus dem XML-Dokument-Baum. Der Vergleich zur bisherigen START-Anweisung für Dateien zeigt für die einfachste Form nur folgende, kleinere Unterschiede:

- Eine START-Anweisung kann gleichzeitig auf mehrere Schlüssel positionieren: auf das in der Anweisung angegebene Datenfeld und alle ihm untergeordneten.
- Die KEY-Angabe entfällt. Da es sich bei einem XML-Dokument-Baum i.A. um ein zweidimensionales Gebilde handelt, ist die bei eindimensionalen Dateien mögliche Positionierung 'vor' bzw. 'nach' einem Schlüssel nicht sinnvoll auf XML-Dokumente (mit gleichzeitig mehreren Schlüsseln) übertragbar.
- Analog zum XML-Format der READ-Anweisung ist die Art des Datenfeldes anzugeben, über das positioniert wird.

Eine neue START-Anweisung sieht kurz folgendermaßen aus:

```
START xml-file ELEMENT datenname-1
```

oder

```
START xml-file ATTRIBUTE datenname-1
```

Der Unterschied zwischen START-Anweisung und READ-Anweisung, die auch auf Knoten im Baum positioniert, besteht in der Festlegung des Wirkungsbereichs, ausgehend von dem in der Anweisung angegebenen Datenfeld.

Das in der START-Anweisung angegebene Datenfeld spielt dafür selbst keine Rolle. Im Unterschied zur READ-Anweisung muss diesem Datenfeld kein Knoten zugeordnet zu sein. Entscheidend ist, dass dem in der COBOL-Struktur **direkt übergeordneten Datenfeld** ein Knoten aus dem Baum zugeordnet sein muss. Der Wirkungsbereich der START-Anweisung umfasst dann den Teilbaum, der diesen Knoten als Wurzel hat.

Der Zuordnungsvorgang beginnt im ersten Schritt auf der COBOL-Seite mit dem in der START-Anweisung angegeben Datenfeld. Auf XML-Seite werden im Wirkungsbereich (wie oben definiert) **alle Kinder** des Wurzel-Knotens auf eine mögliche Zuordnung hin überprüft, in der Reihenfolge von den älteren zu den jüngeren (also werden im Unterschied zur READ-Anweisung, sofern datenname-1 eine gültige Position hat, auch dessen ältere Geschwister überprüft). Anschließend werden weitere Zuordnungen durchgeführt, wie unter "Zuordnungsvorgang" beschrieben.

START modifiziert nur den EPV, lässt Daten aber unverändert.

Beispiel 12-51 Positionieren durch START

XML-Dokument	Pos	COBOL-Datenstruktur		**OPEN DOCUMENT xml-fil**	**READ xml-fil ELEMENT x**		**START xml-fil ELEMENT y**	**READ xml-fil ELEMENT y**
<dok>1	1	**FD xml-fil**						
<a>2	2	**01 x IDENTIFIED**		**1(o)**	**1(r)**		*1(r)*	*1(r)*
<b>3<	3	**BY "dok".**						
/b>	4	**02 x- wert PIC 9.**			**1**		*1*	*1*
<c>4<		**02 y IDENTIFIED**	MOVE	**inv**	**inv**	MOVE	**2(o)**	**2(r)**
/c>	5	**BY y-name.**	"f" TO			"a" TO		
	6	**03 y-name PIC X.**	y-	*f*	*f*	y-	*a*	*a*
<a>5	7	**03 y- wert PIC 9.**	name			name		**2**

6 </dok>		03 z IDENTIFIED BY z-name. 04 z-name PIC X. 04 z- wert PIC 9.	MOVE "g" TO z-name	inv g	inv g	MOVE "b" TO z-name	3(o) b	3(r) b 3

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Das in der START-Anweisung angegebene Datenfeld (y) muss keine gültige Position haben. Notwendig ist die gültige Position des direkt übergeordneten Datenfeldes (x mit Knoten 1).
- Der Wirkungsbereich der START-Anweisung ist der Teilbaum mit Wurzel Knoten 1. Für die Zuordnung zu Datenfeld y werden die Kinder dieses Knotens (zuerst Knoten 2, dann Knoten 5) überprüft. Als erstes passt Knoten2 und die Zuordnung wird mit diesem Teilbaum fortgesetzt.
- Da die START-Anweisung nur positioniert, aber keine Daten überträgt, werden durch die nachfolgende READ-Anweisung die Daten zu dem positionierten Knoten 2 und dem durch Knoten 2 bestimmten Teilbaum geliefert.

Beispiel 12-52 Wiederholtes Positionieren auf den gleichen Knoten (Fortsetzung von Beispiel 12-51)

XML-Dokument	Pos	COBOL-Datenstruktur	...	READ xml-fil ELEMENT y	START xml-fil ELEMENT y
<doc>1	1	FD xml-fil			
<a>2	2	01 x IDENTIFIED	1(r)	1(r)	1(r)
3	3	BY "dok".			
<c>4</c>	4	02 x- wert PIC 9.	1	1	1
		02 y IDENTIFIED	2(r)	5(r)	2(o)
<a>5	5	BY y-name.			
6	6	03 y-name PIC X.	a	a	a
<c>7</c>	7	03 y- wert PIC 9.	2	5	5
		03 z IDENTIFIED	3(r)	6(r)	3(o)
</doc>		BY z-name.			
		04 z-name PIC X.	b	b	b
		04 z- wert PIC 9.	3	6	6

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Das Lesen über Datenfeld y liefert das nächste Exemplar des Knotens mit Namen 'a'.
- Für die folgende START-Anweisung über y spielen die Positionen der Datenfelder y und z keine Rolle, einzig die Position des dem y übergeordneten Datenfeldes x ist von Bedeutung.
- Da sich die Position von Datenfeld x gegenüber der ersten START-Anweisung (siehe Beispiel 12-51) nicht geändert hat, ist der Wirkungsbereich noch der gleiche. Da auch die Werte der Schlüssel y-name und z-name unverändert sind, ergibt sich die gleiche Zuordnung, wie bei der ersten START-Anweisung.
- Die START-Anweisung erlaubt es, erneut auf eines der bereits gelesenen älteren Geschwister zu positionieren (Knoten 2). Dies ist mit der READ-Anweisung nie möglich, dort wären nur jüngere Geschwister im Wirkungsbereich.

- Die START-Anweisung überträgt keine Daten. Daher sind die Inhalte der zu y und z gehörenden Werte (y-wert, z-wert) unverändert, entsprechen aber nicht denen der durch die neue Positionierung zugeordneten Knoten (Knoten 2 und 3).

INDEX-Angabe bei START

START erlaubt zusätzlich nach dem datenname-1 noch die Angabe von

```
INDEX {bezeichner-1 | integer-1}
```

Der angegebene ganzzahlige positive Wert bestimmt, auf den wievielten Knoten von mehreren mit gleichem Elementnamen positioniert werden soll.

Die INDEX-Angabe ist auch für Attribut-Knoten erlaubt, obwohl Attributnamen eindeutig sein müssen. Wenn das i-te Attribut unabhängig vom Namen (d.h. IDENTIFIED USING) gelesen werden soll, macht die Index-Angabe für Attributnamen dennoch Sinn.

Beispiel 12-53 INDEX-Angabe bei START

XML-Dokument	Pos	COBOL-Datenstruktur		**OPEN DOCUMENT xml-fil**	**START xml-fil ELEMENT z INDEX 2**	**START xml-fil ELEMENT u INDEX 3**	**START xml-fil ELEMENT y**
<doc>1 <a>2 3 <c>4 <d>5</d> <d>6</d> </c> <c>7 <e>8</e> <e>9</e> </c> <f>10</f> </doc>	1 2 3 4 5 6 7 8 9 10	**FD xml-fil 01 x IDENTIFIED BY "dok". 02 x-wert PIC 99. 02 y IDENTIFIED BY y-name. 03 y-name PIC X. 03 y-wert PIC 99. 03 z IDENTIFIED BY z-name. 04 z-name PIC X. 04 z- wert PIC 99. 04 u IDENTIFIED BY u-name. 05 u-name PIC X. 05 u-wert PIC 99.**	MOVE "a" TO y-name MOVE "c" TO z-name MOVE "e" TO u-name	**1(o)** **2(o)** *a* **4(o)** *c* **inv** *e*	*1(o)* *2(o)* *a* **7(o)** *c* **8(o)** *e*	*1(o)* *2(o)* *a* *7(o)* *c* **inv** *e*	*1(o)* **2(o)** *a* **4(o)** *c* **inv** *e*
		FILE STATUS		**00**	**00**	**23**	**00**

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Der Wirkungsbereich der ersten START-Anweisung über z ist der Teilbaum mit Wurzel Knoten 2 (als der dem übergeordneten Datenfeld y zugeordnete Knoten).
- Im ersten Schritt werden die Kinder von Knoten 2, d.h. Knoten 3, 4 und 7 auf eine mögliche Zuordnung überprüft. Da als Index 2 angegeben ist, wird nicht der erste passende Knoten (Knoten 4), sondern der zweite passende Knoten (Knoten 7) zugeordnet.

- Der Index bezieht sich nur auf das in der START-Anweisung angegebene Datenfeld, nicht auf die Zuordnung von untergeordneten Datenfeldern (u). Von den 2 Exemplaren mit Namen 'e' wird das erste zugeordnet. Wenn für untergeordnete Datenfelder auch eine Index-Angabe wirken soll, ist für jedes eine entsprechende START-Anweisung erforderlich.
- Der Zuordnungsvorgang für die zweite START-Anweisung betrachtet Knoten8 und 9 (als Kinder des dem z zugeordneten Knoten 7), findet also nicht so viele Knoten, wie verlangt (Index 3, aber nur 2 passende Knoten mit Namen 'e'). Es kann daher kein Knoten zugeordnet werden. Die Position des Datenfelds (u) wird auf ungültig gesetzt und eine Schlüsselfehler-Bedingung existiert.
- Die dritte START-Anweisung hat als Wirkungsbereich das ganze Dokument. Positionierungen für untergeordnete Datenfelder (y, z, u), die bereits erfolgt sind, spielen keine Rolle.
- Bei der dritten START-Anweisung kann dem Datenfeld (u) kein Knoten zugeordnet werden. Da u dem in der Anweisung angegebenen Datenfeld y **untergeordnet** ist, führt diese Situation **nicht** zu einer Schlüsselfehler-Bedingung.

12.10.1.10 Fehlerbehandlung

Für die Behandlung von Fehlern stehen die bei der bisherigen Datei-Ein-/Ausgabe vorhandenen Sprachmittel auch für das Verarbeiten von XML-Dokumenten durch die neuen Anweisungs-Formate zur Verfügung:

- OPEN DOCUMENT und READ: **AT END**- und **NOT AT END**-Angabe
- START: **INVALID KEY**- und **NOT INVALID KEY**-Angabe
- Für alle neuen Formate der Anweisungen: **USE-Prozeduren** und **zusätzliche neue XML-spezifische Werte für den Ein-/Ausgabe-Zustand** (FILE STATUS).
- OPEN DOCUMENT: neue Ausnahmesituation EC-XML-CODESET-CONVERSION

Die [NOT] END- bzw. [NOT] KEY-Angaben entsprechen sowohl syntaktisch, als auch von ihrer Wirkungsweise her den Angaben bei den bisherigen Ein-/Ausgabe-Anweisungen.

OPEN DOCUMENT

Prinzipiell sind in einer Datei mehrere XML-Dokumente hintereinander erlaubt. Momentan werden aber nur Dateien mit einem einzigen Dokument unterstützt. Jede OPEN
DOCUMENT-Anweisung ohne AT-Angabe schließt ein ggf. noch offenes Dokument und öffnet das nächste Dokument in der Datei. Wenn es kein nächstes Dokument gibt, oder die Datei leer ist, tritt die Ende-Bedingung auf, und der Ein-/Ausgabe-Zustand wird auf 10 gesetzt.
Wenn das XML-Dokument Zeichen enthält, die sich nicht als nationale Zeichen (UTF-16) darstellen lassen, tritt die Ausnahmesituation EC-XML-CODESET-CONVERSION auf. Diese Ausnahmebedingung führt nicht zu einem Abbruch. Die betroffenen Zeichen werden im XML-Dokument-Baum durch das Ersatzzeichen '.' ersetzt. EC-XML-CODESET-CONVERSION tritt nur bei der Umwandlung von der externen Darstellung des XML-Dokuments zum Baum auf – höchstens einmal.

READ

Die AT END-Angabe bezieht sich nur auf eine Ende-Bedingung für das eine Datenfeld, das in der READ-Anweisung angegeben worden ist, nicht auf untergeordnete. Die **Ende-Bedingung** tritt sowohl beim sequenziellen, als **auch beim wahlfreien Lesen** auf, wenn es keinen Knoten im XML-Dokument-Baum gibt, der dem Datenfeld (aus der Anweisung) zugeordnet werden kann. Der entsprechende Wert für den Ein-/Ausgabe-Zustand ist 10.Als Zeichensatz zur Darstellung der Daten ist für Übertragungen aus dem XML-DokumentBaum national (UTF-16) anzunehmen. Daher kann bei jeder einzelnen Übertragung im Rahmen einer READ-Anweisung die Ausnahmesituation EC-DATA-CONVERSION auftreten.

Im Gegensatz zu EC-XML-CODESET-CONVERSION von der OPEN DOCUMENT-Anweisung kann EC-DATA-CONVERSION bei Übertragungen von Daten aus dem Baum zum Programm auftreten – ggf. sogar mehrmals bei einer einzelnen READ-Anweisung.

START

Die INVALID KEY-Angabe bezieht sich nur auf eine Schlüsselfehler-Bedingung für das eine Datenfeld, das in der START-Anweisung angegeben worden ist, nicht auf untergeordnete. Die Schlüsselfehler-Bedingung tritt auf, wenn es keinen Knoten im XML-Dokument-Baum gibt, der diesem Datenfeld zugeordnet werden kann. Der entsprechende Wert für den Ein-/Ausgabe-Zustand ist 23.

Alle Anweisungen

Wenn die Reihenfolge der Anweisungen nicht eingehalten wird (siehe auch "Anweisungssequenzen" in Kapitel "Anweisungen für die XML-Verarbeitung"), sind die Voraussetzungen für die Ausführung einzelner Anweisung nicht erfüllt. Das führt zu einer erfolglosen Ausführung wegen logischem Fehler und einem entsprechenden Ein-/Ausgabe-Zustand.

Ein-/Ausgabe-Zustand für XML-Dateien

Ein-/Ausgabe-Zustand	Bedeutung
00 05 08	**Erfolgreiche Ausführung** Die Anweisung wurde erfolgreich ausgeführt. Es sind keine weiteren Informatio nen verfügbar. OPEN-Anweisung auf eine nicht vorhandene, optionale Datei. Eine READ-Anweisung konnte einige Elemente und/oder Attribute aus dem XML-Dokument keiner COBOL-Beschreibung zuordnen.
10	**Erfolglose Ausführung: Endebedingung** 1. Es wurde versucht, mit einer READ-Anweisung, ein Attribut oder Element zu lesen, das nicht existiert. 2. Es wurde versucht, für eine leere oder eine nicht vorhandene, optionale Da tei eine OPEN DOCUMENT-Anweisung auszuführen. 3. Es wurde versucht, ein zweites Dokument in einer XML-Datei mittels OPEN DOCUMENT-Anweisung zu öffnen.
23 25	**Erfolglose Ausführung: Schlüsselfehlerbedingung** 1. Es wurde versucht, mit einer START-Anweisung auf ein Attribut oder Ele ment zu positionieren, das nicht existiert. 2. Die Index-Angabe bei einer START-Anweisung war nicht positiv. Es wurde versucht, eine START-Anweisung ohne gültige Position auszuführen.
30 35 39 3A 3D	**Erfolglose Ausführung: Permanenter Fehler** Keine weitere Information verfügbar. OPEN auf nicht vorhandene Datei. Es wurde versucht, eine Datei mit unpassenden Dateimerkmalen zu öffnen. Es wurde versucht, eine OPEN DOCUMENT-Anweisung für ein XML-Dokument auszuführen, das nicht wohlgeformt ist. Das schließt auch Dateiinhalte mit ein, die keine XML-Dokumente darstellen. Es wurde versucht, eine OPEN DOCUMENT-Anweisung auszuführen, aber der Zeichensatz, in dem das XML-Dokument dargestellt ist, konnte nicht ermittelt werden.
41 42	**Erfolglose Ausführung: Logischer Fehler** Es wurde versucht, eine OPEN-Anweisung für eine Datei auszuführen, die be reits geöffnet ist. Es wurde versucht, eine CLOSE-Anweisung für eine Datei auszuführen, die nicht geöffnet ist.
46 47 4B	1. Es wurde versucht, eine OPEN DOCUMENT-Anweisung (mit der AT-Angabe) oder eine READ-Anweisung ohne eine gültige Position auszuführen.

4C 4D 4E	2. Es wurde versucht, eine OPEN DOCUMENT-Anweisung (ohne AT-Angabe) auszuführen, nachdem die vorherige OPEN DOCUMENT-Anweisung eine Ende-Bedingung verursacht hat. Es wurde versucht, eine READ- oder START-Anweisung für eine nicht geöffnete Datei auszuführen. Es wurde versucht, eine OPEN DOCUMENT- oder CLOSE DOCUMENT-Anweisung für eine Datei auszuführen, die nicht geöffnet ist. Es wurde versucht, eine OPEN DOCUMENT-, START- oder READ-Anweisung auszuführen, bei der Namen in IDENTIFIED-Klauseln für Geschwisterknoten nicht eindeutig waren. Es wurde versucht, eine CLOSE DOCUMENT-, OPEN DOCUMENT- (mit der AT-Angabe), READ- oder START-Anweisung für ein Dokument auszuführen, das nicht geöffnet ist. Es wurde versucht eine OPEN DOCUMENT-, READ- oder START-Anweisung auszuführen, bei der sich im COBOL-Programm angegebene Element- und/oder Attributnamen nicht in UTF-16 darstellen lassen.
	Sonstige erfolglose Ausführungen
90	Systemfehler - keine weitere Information über die Ursache vorhanden.
91	OPEN Fehler. Eigentliche Ursache ggf. aus dem SIS-Code ersichtlich.
97	Der für eine OPEN DOCUMENT-Anweisung erforderliche Arbeitsspeicher steht nicht zur Verfügung.

Tabelle 47: Ein-/Ausgabe-Zustände für XML-Dateien

Diese Ein-/Ausgabe-Zustände beziehen sich nur auf die primäre zu verarbeitende XML- Datei. Wenn beim ggf. zusätzlich erforderlichen Zugriff auf externe Entitäten Fehler auftre ten, führen diese i.A. zum Ein-/Ausgabe-Zustand 3A.

Beispiel 12-54 Mehrdeutige Namen

XML-Dokument	Pos	COBOL-Datenstruktur		**OPEN DOCUMENT xml-fil**		**READ xml-fil ELEMENT x**
<dok>1 <a>2 3 <a>4 </dok>	1 2 3 4	**FD xml-fil** **01 x IDENTIFIED BY "dok".** **02 y IDENTIFIED BY y-name.** **03 y-name PIC X.** **02 z IDENTIFIED BY z-name.** **03 z-name PIC X.**	**MOVE "a" TO y-name** **MOVE "b" TO z-name**	**1(o)** **2(o)** *a* **3(o)** *b*	**MOVE "a" TO z-name**	**inv** **inv** *a* **inv** *a*
		FILE STATUS		**00**		**4C**

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Die aktuellen Schlüsselwerte der Datenfelder y und z sind bei der READ-Anweisung gleich: Dies ist verboten, da keine eindeutige Zuordnung für diesen Fall definiert ist (ist Knoten 2 dem z und Knoten 4 dem y zuzuordnen oder umgekehrt?). Daher liefert die READ-Anweisung den Ein-/Ausgabe-Zustand 4C.

- In diesem Fall erhalten das Datenfeld, **über das gelesen wurde –** obwohl das Problem gar nicht bei diesem Datenfeld, sondern bei untergeordneten aufgetreten ist –, sowie alle ihm untergeordneten Datenfelder eine ungültige Position.

12.10.1.11 Namensraum (namespace)

Qualifizierte Namen in XML-Dokumenten setzen sich zusammen aus einem lokalen Namen und einem Namensraum, innerhalb dessen der lokale Name eindeutig sein muss. Der Namensraum wird wie ein Attribut bei einem Element angegeben:

- Er kann einen Standard-Namensraum festlegen: `xmlns="..."`
  Dieser Standard-Namensraum gilt für dieses Element und alle untergeordneten, sofern dort kein anderer Namensraum in einem Elementnamen angegeben ist, nicht jedoch für Attribute.
- Er kann ganz ausgeschaltet werden: `xmlns=""`
  Dieser leere Namensraum gilt ebenfalls für dieses Element und alle untergeordneten, sofern dort nichts anderes angegeben ist.
- Er kann eine Abkürzung (Präfix) für einen (Nicht-Standard-)Namensraum definieren:
  `xmlns:präfix="..."`
  Dies ist auch mehrfach erlaubt, mit unterschiedlichen Präfixen, um über mehrere Namensräume verfügen zu können.
  Ein solches Präfix darf in diesem Element, allen ihm untergeordneten Elementen **und auch** den Attributen dieser Elemente den lokalen Namen vorangestellt werden, um einen eindeutigen, qualifizierten Namen zu bilden, in der Form `präfix:lokalername` .

Die in der **IDENTIFIED-Klausel** aufgeführten Element- bzw. Attributnamen beschreiben **immer den lokalen Namen**. Der Verarbeitung von Namensräumen aus dem XML-Dokument dient die optionale NAMESPACE-Angabe in der IDENTIFIED-Klausel. Sie erlaubt es, für das beschriebene Element bzw. Attribut:

- einen gewünschten Namensraum vorzugeben:
  NAMESPACE IS {datenname-3 | literal-2}
- beliebige Namensräume zu akzeptieren: **NAMESPACE USING datenname-4**
  Die Wirkung entspricht der USING-Angabe bei IDENTIFIED: In datenname-4 wird der gelesene Namensraum zurückgeliefert.
- den leeren Namensraum festzulegen: **NAMESPACE IS NULL**

Die implizite Wirkung von Namensräumen für untergeordnete Elemente eines XML-Dokuments findet sich analog in COBOL wieder:

- Die NAMESPACE-Angabe einer IDENTIFIED-Klausel gilt für das angegebene Datenfeld **und** vererbt sich auf alle untergeordneten Datenfelder mit IDENTIFIED-Klausel und ELEMENT-Angabe, jedoch **nicht** auf Datenfelder mit ATTRIBUTE-Angabe.
- Wenn ein untergeordnetes Datenfeld selbst eine NAMESPACE-Angabe macht, hat diese Vorrang vor einer evtl. geerbten.
- Wenn auf Stufe 01 in einer IDENTIFIED-Klausel die NAMESPACE-Angabe fehlt, ist das gleichbedeutend mit der Angabe NAMESPACE NULL.

Die Präfixe, welche als Abkürzung in der XML-Notation möglich sind, haben mit dem NAMESPACE in COBOL nichts zu tun. Sie sind in COBOL nicht sichtbar.

> **i** Als Wert eines Namensraums verwenden die COBOL-Anweisungen immer den 'kompletten' Wert aus der xmlns-Angabe im XML-Dokument, nie das Präfix.

Die Unterscheidung auf der XML-Seite zwischen Standard-Namensraum und mit Präfixen explizit angegebenen Namensräumen ist auf der COBOL-Seite ohne Bedeutung, da hier keine Präfixe existieren.

Die möglichen Kombinationen der Angaben von BY bzw. USING bei lokalen Namen und Namensräumen unterliegen bestimmten Einschränkungen. Dabei spielt es keine Rolle, ob die NAMESPACE-Angabe explizit

gemacht wurde oder implizit gilt. Die Einschränkungen betreffen einerseits die Angaben innerhalb einer einzigen IDENTIFIED-Klausel. Anderer-seits betreffen Sie die Kombination von zwei oder mehr Datenerklärungen mit IDENTIFIED-Klausel der gleichen Art (ELEMENT bzw. ATTRIBUTE), sofern sie in der COBOL-Struktur dem gleichen Datenfeld direkt untergeordnet sind.

Die nachfolgende Tabelle gibt einen Überblick über erlaubte Kombinationen der Angaben von BY bzw. USING innerhalb einer IDENTIFIED-Klausel mit beliebiger Angabe der Art (ELEMENT oder ATTRIBUTE):

Lokaler Name	NAMESPACE		
	NULL	IS	USING
BY	X	X	–
USING	X	–	X

Die nachfolgende Tabelle gibt einen Überblick über erlaubte Kombinationen der Angaben von BY bzw. USING bei zwei Datenerklärungen mit IDENTIFIED-Klauseln mit gleicher An-gabe der Art. D.h. die Verbote gelten jeweils nur innerhalb der Menge von IDENTIFIED-Klauseln, die alle die ELEMENT- bzw. alle die ATTRIBUTE-Angabe machen. Für Mischungen gibt es keine Einschränkungen. Es ist also zulässig, einem Datenfeld zwei Datenfelder mit IDENTIFIED **BY**...ATTRIBUTE und IDENTIFIED **USING**...ELEMENT direkt unterzuordnen.

(1) IDENTIFIED-Klausel für sich bereits ungültig

(2) ungültig wegen Kombination von BY mit USING

(3) mehr als eine IDENTIFIED-Klausel mit USING ungültig

NAMESPACE	NAMESPACE lokaler Name lokaler Name	NULL BY	NULL USING	IS BY	IS USING	USING BY	USING USING
NULL	BY	X	– (2)	X	– (1)	– (1)	– (2)
NULL	USING	– (2)	– (3)	– (2)	– (1)	– (1)	– (3)
IS	BY	X	– (2)	X	– (1)	– (1)	– (2)
IS	USING	– (1)	– (1)	– (1)	– (1)	– (1)	– (1)
USING	BY	– (1)	– (1)	– (1)	– (1)	– (1)	– (1)
USING	USING	– (2)	– (3)	– (2)	– (1)	– (1)	– (3)

Das oben beschriebene **Verfahren** der Zuordnung ist durch Namensräume nicht betroffen. Die Namensräume sind jedoch **als Voraussetzungen für die Zuordnung eines einzelnen Knotens zu berücksichtigen**.

Für die Zuordnung eines Knoten aus dem Baum zu einem Datenfeld gelten daher folgende Voraussetzungen:

- Die gleiche Art des Knotens im Baum und in der IDENTIFIED-Klausel des Datenfeldes (d.h. beide sind Element-Knoten bzw. Attribut-Knoten)
- Der in der IDENTIFIED-Klausel angegebene **lokale Namen** muss mit dem lokalen Namen des Knotens im Baum übereinstimmen:
  - Bei einer BY-Angabe müssen die Namen identisch sein, abgesehen von endständigen Leerzeichen.
  - Bei einer USING-Angabe wird jeder beliebige Name aus dem Baum als übereinstimmend angesehen.
- Der in der IDENTIFIED-Klausel angegebene **Namensraum** muss mit dem Namensraum des Knotens im Baum übereinstimmen (dabei spielt es keine Rolle, ob die NAMESPACE-Angabe für das Datenfeld explizit gemacht ist, oder geerbt wurde; gleiches gilt für die namespaces im XML-Baum):
  - Bei einer IS-Angabe müssen die Namensräume identisch sein, abgesehen von endständigen Leerzeichen.
  - Bei einer USING-Angabe wird jeder beliebige Namensraum aus dem Baum als übereinstimmend angesehen, auch der leere Namensraum.

- Bei der Angabe NULL muss der Knoten im Baum einen leeren Namensraum haben.

Beispiel 12-55 Vorgebener NAMESPACE

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	READ xml-fil ELEMENT x
<doc xmlns="http://www.abc" xmlns:r="http://www.efg"> <a>2 3 <r:c>4</r:c> </doc>	1 2 3 4	FD xml-fil 01 x IDENTIFIED BY "doc" NAMESPACE "http://www.abc". 02 y IDENTIFIED BY "a" NAMESPACE "http://www.abc". 03 y-wert PIC 9. 02 z IDENTIFIED BY "c". 03 z-wert PIC 9. 02 u IDENTIFIED BY "b" NAMESPACE NULL. 03 u-wert PIC 9.	1(o) 2(o) inv inv	1(r) 2(r) 2 inv inv
		FILE STATUS	00	08

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Knoten 1 und Datenfeld x geben explizit den gleichen Namensraum an. Da auch die Elementnamen gleich sind, kann Knoten1 zugeordnet werden.
- Die NAMESPACE-Angabe bei Datenfeld y ist eigentlich überflüssig, da sie identisch ist mit derjenigen des übergeordneten Datenfeldes x. y würde ohne eigene Angabe den Namensraum von x erben.
- Knoten 2 erbt den Standard-Namensraum von Knoten1 und hat somit den gleichen Namensraum wie Datenfeld y. Da auch die Elementnamen gleich sind, kann Knoten 2 zugeordnet werden.
- Knoten 3 und Datenfeld u haben zwar den gleichen lokalen Namen (b), Knoten 3 erbt jedoch den Namensraum http://www.abc, während Datenfeld u explizit den leeren Namensraum verlangt. Knoten 3 kann also nicht zugeordnet werden.
- Knoten 4 und Datenfeld z haben ebenfalls den gleichen lokalen Namen (c). Knoten 4 gibt jedoch explizit den Namensraum http://www.efg an, während Datenfeld z als Element ohne eigene NAMESPACE-Angabe den Namensraum http://abc des übergeordneten Datenfeldes x erbt. Da die Namensräume verschieden sind, kann Knoten4 nicht zugeordnet werden.
- Beim Lesen über x lassen sich die Knoten 3 und 4 nie zuordnen und bewirken den Ein-/Ausgabe-Zustand 08.

Beispiel 12-56 Beliebiger NAMESPACE

XML-Dokument	Pos	COBOL-Datenstruktur	OPEN DOCUMENT xml-fil	READ xml-fil ELEMENT x	READ xml-fil ELEMENT y
<doc xmlns=" http://www.abc"	1	FD xml-fil 01 x IDENTIFIED	1(o)	1(r)	*1(r)*

xmlns:r=" http://www.efg"> <a>2 3 <r:c>4</r:c> </doc>	2 3 4	USING x-name NAMESPACE USING x-nsp. 02 x-name PIC XXX. 02 x-nsp PIC X(14). 02 y IDENTIFIED USING y-name NAMESPACE USING y-nsp. 03 y-name PIC XXX. 03 y-nsp PIC X(14). 03 y-wert PIC 9.	2(o)	dok 'BLANK' (x14) [1] 2(r) a 'BLANK''BLANK' 'BLANK' (x14) [1] 2	*dok* 'BLANK' (x14) 1 3(r) c 'BLANK''BLANK' http://www.efg 4
		FILE STATUS	00	00	00

[1] Das Symbol 'BLANK' wird 14 mal wiederholt.

Anmerkungen:

- Hinweise zur Darstellung dieses Beispiels finden Sie auf "Anweisungen für die XML-Verarbeitung".
- Weder der Elementname, noch der Namensraum macht Vorgaben. Daher lassen sich alle Knoten prinzipiell zuordnen. Aus diesem Grund liefert die erste READ-Anweisung (anders als in Beispiel 12-55) den Ein-/Ausgabe-Zustand 00.
- Knoten 1 hat keinen (d.h. einen leeren) Namensraum: Für die Übertragung in das Datenfeld x-nsp wirkt er wie ein Sendefeld mit der Länge Null. Daher die Leerzeichen in x-nsp. Gleiches gilt für Knoten 2 und y-nsp.
- Die zweite READ-Anweisung liest Knoten 3: Als Elementname wird in y-name nur der lokale Name 'c' geliefert. Als Namensraum wird der komplette Namensraum aus der zugehörigen xmlns-Angabe (bei Knoten 1) geliefert. Das zur Abkürzung verwendete Präfix 'r' ist in COBOL nicht sichtbar.

12.10.2 Ereignisorientierte Verarbeitung

Die neue Anweisung **XML PARSE** zerlegt ein XML-Dokument. Dabei wird das Dokument sequenziell als Datenstrom verarbeitet. Wenn der Parser im Dokument syntaktische Einheiten, wie etwa Tags für Beginn und Ende eines Elements, Attribute, Werte usw. erkennt, löst er ein dafür spezifisches Ereignis aus. Das COBOL-Programm stellt für die Behandlung solcher Ereignisse eine Routine bereit. Die Daten, die das Ereignis charakterisieren, stehen dort in Sonderregistern zur Verfügung. Wesentlich ist dabei, dass während der Ausführung der XML-Anweisung das COBOL-Programm **mehrfach** die Kontrolle in der Behandlungsroutine erhält – immer wenn eines der Ereignisse aufgetreten ist –, aber diese Kontrolle auch wieder an den Parser zurückgeben muss. Erst wenn auf diese Art das ganze Dokument verarbeitet worden ist, oder der Anwender auf die weitere Verarbeitung verzichtet hat, ist die XML-Anweisung beendet.

12.10.2.1 XML-Anweisung

Die neue Anweisung **XML** dient dem Lesen von XML-Dokumenten. Dazu müssen folgende zwei Angaben gemacht werden, die dritte Angabe ist optional:

1. Was soll getan werden? (Pflicht)

   PARSE bezeichner-1

   Das XML-Dokument, das im Datenfeld bezeichner-1 steht, ist zu zerlegen.

2. Wer soll die Ergebnisse der Zerlegung verarbeiten? (Pflicht)

   PROCESSING PROCEDURE IS prozedurname-1 [{ THRU | THROUGH } prozedurname-2]

   Bei jedem vom Parser erkannten 'Ereignis' während der Ausführung der XML-Anweisung geht die Steuerung an die Verarbeitungsprozedur prozedurname-1. Wenn die letzte Anweisung von prozedurname-1 bzw. prozedurname-2 durchlaufen ist, geht die Steuerung automatisch zurück an die noch aktive XML-Anweisung.

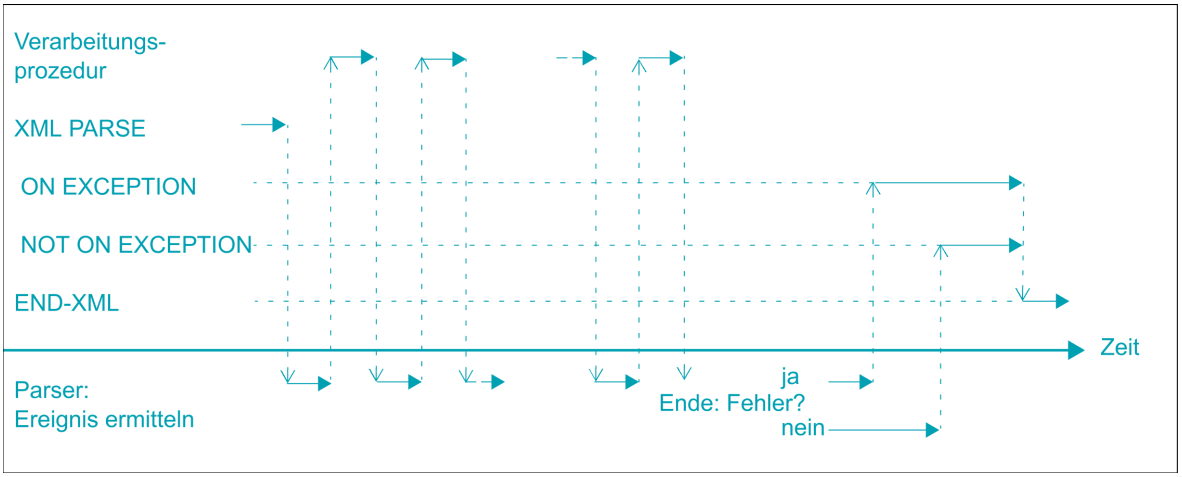
3. Wie soll die Anweisung beendet werden? (optional)

   [ON EXCEPTION unbedingte-anweisung-1]

   [NOT ON EXCEPTION unbedingte-anweisung-2]

   kennzeichnet zusätzliche Anweisungen, die ggf. bei irregulärem bzw. regulärem Ende der Zerlegung ausgeführt werden sollen.

Die Aktivitäten von XML-Anweisung, Verarbeitungsprozedur für Ereignisse und XML-Parser sind zeitlich folgendermaßen verschränkt:



Zusammenfassend ist bei der XML-Anweisung Folgendes besonders zu beachten:

- Das XML-Dokument – genau eines – steht im angegebenen Datenfeld.

  Der wiederholte Wechsel zwischen XML-Anweisung und Verarbeitungsprozedur findet so lange statt, bis eine der folgenden Situationen auftritt:

  - Das gesamte XML-Dokument ist verarbeitet worden.
  - Die Verarbeitungsprozedur zeigt an, dass die Verarbeitung sofort beendet werden soll.
  - Ein Fehler wurde gemeldet.
- Die sequenzielle Verarbeitungsweise bedeutet für **nicht wohlgeformte** Dokumente, dass der Parser ggf. bereits Ereignisse ausgelöst hat, bevor er im Dokument auf die fehlerhafte Stelle trifft.
- Der Verarbeitungsprozedur steht die genaue Beschreibung des Ereignisses und der damit verbundenen Daten in Sonderregistern zur Verfügung.
- Das komplette Dokument wird durch Ausführung einer **einzigen** Anweisung zerlegt und verarbeitet.

- Vorhandene [NOT] EXCEPTION-Angaben werden nicht nach jedem Ereignis, sondern nur ein einziges Mal nach Ende der gesamten XML-Anweisung durchlaufen.

12.10.2.2 Sonderregister

Die Tabelle gibt einen Überblick über Namen, Inhalt und Eigenschaften der Sonderregister:

Sonderregister	Inhalt	USAGE	Länge
XML-EVENT	Name des Ereignisses	DISPLAY	fest; 30 Zeichen
XML-CODE	Fehlerschlüssel	COMP-5	fest; 9 Ziffern mit Vorzeichen
XML-TEXT	mit dem Ereignis verbundener Text	DISPLAY	variabel; maximal die Länge des in der XML-Anweisung angegebenen Datenfeldes
XML-NTEXT		NATIONAL	
XML-NAMESPACE	Namensraum eines Elements, Attributs oder einer Deklaration	DISPLAY	
XML-NNAMESPACE		NATIONAL	
XML-NAMESPACE-PREFIX	Präfix für Abkürzung eines Namensraums	DISPLAY	
XML-NNAMESPACE-PREFIX		NATIONAL	

XML-CODE dient der Kommunikation zwischen Anwendung und Parser und darf daher auch von der Anwendung geändert werden.

Die anderen Sonderregister versorgt der Parser. Die Anwendung sollte sie also nur lesen, ein Schreiben führt zu undefiniertem Verhalten. Die aktuelle Länge der in variabel langen Sonderregistern bereitgestellten Daten ist bei Bedarf mittels FUNCTION LENGTH zu bestimmen.

Der Parser versorgt bei einem Ereignis nur einen Teil der Sonderregister, abhängig von der Klasse des Datenfeldes in dem das XML-Dokument steht. Nicht versorgte Sonderregister haben immer die aktuelle Länge 0.

	bezeichner-1	
	DISPLAY	NATIONAL
XML-EVENT	X	X
XML-CODE	X	X
XML-TEXT	X	–
XML-NTEXT	_[1]	X
XML-NAMESPACE	X	–
XML-NNAMESPACE	–	X
XML-NAMESPACE-PREFIX	X	–
XML-NNAMESPACE-PREFIX	–	X

[1] Ausnahme siehe Ereignis CONTENT-NATIONAL-CHARACTER auf "Verarbeitungsprozedur"

Die Sonderregister stehen in jedem Programm zur Verfügung, das mit der entsprechenden Option übersetzt wurde. Plausible Werte in den variabel langen Registern (d.h. Längen > 0) sind jedoch nur zu erwarten, während eine XML PARSE-Anweisung aktiv ist. In XML-CODE und XML-EVENT ist der letzte Wert auch nach Beendigung der Anweisung noch verfügbar.

12.10.2.3 Verarbeitungsprozedur

Eine Verarbeitungsprozedur verhält sich beim Ablauf so, als würde sie von der XML-Anweisung über PERFORM aktiviert, ähnlich zu einer INPUT/OUTPUT PROCEDURE bei SORT, siehe auch Abschnitt „SORT-Anweisung". Es gelten daher für die Angabe in der XML-Anweisung die gleichen Regeln und Einschränkungen wie für eine PERFORM-Anweisung.

Ereignisse

Innerhalb der Verarbeitungsprozedur kann die Anwendung für Ereignisse, die von Interesse sind, weitere Verarbeitungen vornehmen. Fast alle Ereignisse stellen zu diesem Zweck spezifischen Daten im Sonderregister XML-TEXT bzw. XML-NTEXT bereit. Spalte nsp in der folgenden Tabelle kennzeichnet Ereignisse, bei denen auch die Namensraum-Sonder-register versorgt sind. Dabei handelt es sich um XML-NAMESPACE und XML-NAME-SPACE-PREFIX bzw. XML-NNAMESPACE und XML-NNAMESPACE-PREFIX. Wenn kein Namensraum oder der leere Namensraum angegeben ist, ist die aktuelle Länge der Sonderregister gleich 0; bei einem Standardnamensraum ist die aktuelle Länge der Präfix-Sonderregister gleich 0.

Die Hochkommas bei den diversen Zeichenfolgen in der folgenden Tabelle dienen nur der Hervorhebung. Sie sind nicht Bestandteil der Zeichenfolge.

Name des Ereignisses (in XML-EVENT)	**Inhalt von XML-TEXT bzw. XML-NTEXT**	**nsp**
ATTRIBUTE-CHARACTERS	Wert des Attributs **ohne** die öffnenden und schließenden Literalbegrenzer	
ATTRIBUTE-NAME	Lokaler Name des Attributs	X
COMMENT	Inhalt des Kommentars, **ohne** die Zeichenfolgen '<!--' am Anfang und '->' am Ende	
CONTENT-CHARACTER	Ein einzelnes Zeichen, das einer vordefinierten Entitätenreferenz entspricht	
CONTENT-CHARACTERS	Teil-Zeichenfolge zwischen Beginn- und Ende-Tag eines Elements.Dies ist nur dann der ganze Wert, wenn er keine weiteren Elemente oder Entitätsreferenzen enthält.	
CONTENT-NATIONAL-CHARACTER	Nur in XML-NTEXT für Dokumente, die in einem alphanumerischen Datenfeld bereitgestellt sind: Ein einzelnes nationales Zeichen, das einer numerischen Entität entspricht, die nicht als alphanumerisches Zeichen darstellbar ist. XML-TEXT hat bei diesem Ereignis immer die aktuelle Länge 0.	
DEFAULTED-ATTRIBUTE-NAME	Lokaler Name eines Attributs, das im XML-Dokument nicht enthalten ist, aber in der verwendeten DTD (Document Type Definition) mit einem Vorgabewert erklärt ist.	X
DOCUMENT-TYPE-DECLARATION	Die gesamte Dokumenttypdefinition, **ein schließlich** der Zeichenfolge '<! DOCTYPE' am Anfang und '>' am Ende.	
ENCODING-DECLARATION	Zeichensatzname (wie von XHCS erwartet), der für die Interpretation des Dokuments angenommen wird. Das ist i.A. nicht der in der Encoding Declaration des XML-Dokuments angegebene Name. Vielmehr kann es ein anderer Zeichensatz sein, wenn etwa das XML-	

	Dokument mittels File Transfer übertragen und dabei konvertiert wurde, ohne gleichzeitig auch die Angabe in der Encoding Declaration anzupassen.	
END-OF-CDATA-SECTION	Die Zeichenfolge ']]>'	
END-OF-DOCUMENT	Leer, d.h. aktuelle Länge = 0	
END-OF-ELEMENT	Lokaler Name des Elements	X
EXCEPTION	Beginn des Dokuments bis einschließlich der Fehlerstelle	
NAMESPACE-DECLARATION	Leer, d.h. aktuelle Länge = 0	X
PROCESSING-INSTRUCTION-DATA	Inhalt der Steueranweisung nach dem Namen, **ohne** die Zeichenfolge '?>' am Ende. Führende Durchschüsse (white spaces) sind nicht Bestandteil, endständige jedoch schon.	
PROCESSING-INSTRUCTION-TARGET	Name der Steueranweisung (**ohne** die Zeichenfolge '<?' am Anfang)	
STANDALONE-DECLARATION	Die Zeichenfolge 'yes' oder 'no' aus der XML-Deklaration	
START-OF-CDATA-SECTION	Die Zeichenfolge '<![CDATA['	
START-OF-DOCUMENT	Das gesamte Dokument	
START-OF-ELEMENT	Lokaler Name des Elements	X
VERSION-INFORMATION	Wert aus der Versionsangabe der XML-Deklaration ohne die öffnenden und schließenden Literalbegrenzer	

Fehlerbehandlung

Der Parser teilt der Verarbeitungsprozedur gefundene Fehler durch das Ereignis 'EXCEPTION' mit. Das Sonderregister XML-CODE hat dann einen Wert ungleich 0, der den Fehler genauer beschreibt. Bei allen anderen Ereignissen hat dieses Sonderregister den Wert 0.

Der Inhalt des Sonderregisters XML-CODE beim Verlassen der Verarbeitungsprozedur steuert die Fortsetzung der XML-Anweisung. Die Verarbeitungsprozedur signalisiert durch Setzen entsprechender Werte die gewünschte Fortsetzung:

-1 Beenden der XML-Anweisung mit 'Fehler'. Dies ist bei jedem Ereignis möglich, auch wenn zuvor das Ereignis EXCEPTION nicht aufgetreten ist.

0 Fortsetzung der XML-Anweisung. Wenn bereits ein Fehler gemeldet wurde, ist die Fortsetzung nur bei 'leichten' Fehlern möglich. Sie beschränkt sich aber auf die Suche nach weiteren Fehlern. Bei 'schweren' Fehlern wird die XML-Anweisung trotzdem immer abgebrochen.

andere Werte	Das weitere Verhalten ist undefiniert.

Vorhandene EXCEPTION- bzw. NOT EXCEPTION-Klauseln werden nach Ende der XML-Anweisung ausgeführt, abhängig davon, wie die Anweisung beendet wurde:

NOT EXCEPTION	Bei regulärer Beendigung
EXCEPTION	Bei irregulärer Beendigung wegen Fehlern. Das schließt den aktiven Abbruch durch Setzen des XML-CODE-Sonderregisters in der Verarbeitungsprozedur mit ein.

Beispiel 12-57 Ereignisse und Inhalt der Sonderregister

```
DATA DIVISION.
WORKING-STORAGE SECTION.
...
01 dokument.
 02     VALUE "<?xml version='1.1'?>".
 02     VALUE "<dok xmlns:r='http://www.efg'>".
 02     VALUE "111".
 02     VALUE "<a att='XX'/>".
 02     VALUE "<![CDATA[tse-da-da segdschn]]>"".
 02     VALUE "222".
 02     VALUE "<r:b>".
 02     VALUE "yyyyy   ".
 02     VALUE "</r:b>".
 02     VALUE "</dok>".
```

Die Zerlegung des Dokuments durch die XML-Anweisung führt zu nachstehender Folge von Ereignissen und Inhalten der Sonderregister:

XML-EVENT	XML-TEXT	XML-NAME-SPACE	XML-NAME-SPACE-PREFIX
VERSION-INFORMATION	1.1		
START-OF-DOCUMENT	<?xml version='1.1'?> <dok xmlns:r='http://www.efg'>111<a att='XX'/><![CDATA [tse-da-da segdschn]]> 222<r:b>yyyyy'BLANK"BLANK"BLANK'</r:b></dok>		
START-OF-ELEMENT	dok		
NAMESPACE-DECLARATION		http://www.efg	r
	111		

CONTENT-CHARACTERS			
START-OF-ELEMENT	a		
ATTRIBUTE-NAME	att		
ATTRIBUTE-CHARACTERS	XX		
END-OF-ELEMENT	a		
START-OF-CDATA-SECTION	<![CDATA[		
CONTENT-CHARACTERS	tse-da-dasegdschn		
END-OF-CDATA-SECTION	]]>		
CONTENT-CHARACTERS	222		
START-OF-ELEMENT	b	http://www.efg	r
CONTENT-CHARACTERS	yyyyy'BLANK"BLANK"BLANK'		
END-OF-ELEMENT	b	http://www.efg	r
END-OF-ELEMENT	dok		
END-OF-DOCUMENT			

Anmerkungen:

- Der Wert (111222) des Elements 'dok' ist durch das eingeschobene Element 'a' in zwei Teile aufgespalten. Aufgrund der sequenziellen Verarbeitung des XML-Dokuments liefert der Parser die Teilstücke getrennt – unterbrochen durch die Ereignisse zu Element 'a' – jeweils mit dem Ereignis CONTENT-CHARACTERS.
- Ein leerer Eintrag zeigt an, dass das Sonderregister bei dem Ereignis die aktuelle Länge 0 hat.

Beispiel 12-58 XML-Anweisung und Verarbeitungsprozedur

```
PROCEDURE DIVISION.
...
  XML PARSE dokument
    PROCESSING PROCEDURE IS ereignis
    ON EXCEPTION ...
    NOT ON EXCEPTION ...
  END-XML.
...
ereignis.
  EVALUATE XML-EVENT
    WHEN   "VERSION-INFORMATION"
      IF XML-TEXT NOT = "1.0"
         DISPLAY "XML Version nicht 1.0"
         MOVE -1 TO XML-CODE
      END-IF
WHEN   "CONTENT-CHARACTERS"
```

```
  ...
WHEN   "START-OF-ELEMENT"
IF FUNCTION LENGTH(XML-TEXT) > 31
   DISPLAY "Elementname zu lang"
   MOVE -1 to XML-CODE
ELSE
   MOVE XML-TEXT TO ...
END-IF
WHEN   "END-OF-ELEMENT"
  ...
WHEN   OTHER
  CONTINUE
END-EVALUATE.
```

Anmerkungen:

- Das Programm möchte nur XML-Dokumente der Version 1.0 verarbeiten und für alle anderen Versionen die Verarbeitung beenden: Wenn das Sonderregister XML-CODE mit dem Wert -1 in der Verarbeitungsprozedur versorgt wird, signalisiert dies den Wunsch nach sofortigem Abbruch der XML-Anweisung.
- -1 im Sonderregister XML-CODE bei Rückkehr zur XML-Anweisung bedeutet irreguläres Ende und Fortsetzung mit den Anweisungen bei der ON EXCEPTION-Angabe.
- Das Programm verarbeitet nur Elemente und deren Werte. Andere Ereignisse 'verschluckt' die Verarbeitungsprozedur, ohne dafür etwas zu tun (WHEN OTHER).
- Das Programm möchte nur Elementnamen mit maximal 31 Zeichen akzeptieren. Wenn im Dokument längere Elementnamen vorkommen, wird die XML-Anweisung sofort beendet. Die Länge eines im Sonderregister XML-TEXT bereitgestellten Elementnamen entspricht der aktuellen Länge des Sonderregisters: sie wird von FUNCTION LENGTH geliefert.
- Auch wenn für einzelne Sonderregister keine Definition mit herkömmlichen COBOL-Sprachmitteln möglich ist, dürfen sie in COBOL-Anweisungen verwendet werden. Sie verhalten sich wie alphanumerische bzw. nationale Gruppen variabler Länge, z.B. als Sendefeld in der MOVE-Anweisung.

12.10.3 XML Common Syntactic Constructs

Die in einem XML-Dokument verfügbaren Konstrukte sind an der Schnittstelle in COBOL teilweise gar nicht mehr sichtbar, teilweise ist die Sichtbarkeit von der Art der Verarbeitung abhängig.

Konstrukt / Eigenschaft	**Strukturorientiert**	**Ereignisorientiert**
Numerische Zeichenentität (numeric character reference)	ersetzt durch das entsprechende Zeichen	
	Wenn es im Zielzeichensatz keine Darstellung gibt, wird das Ersatzzeichen geliefert.	Wenn es im Zielzeichensatz keine Darstellung gibt, wird die nationale Darstellung geliefert, d.h. das Ereignis CONTENT-NATIONAL-CHARACTER.
Vordefinierte Entität (predefined entity)	ersetzt durch das entsprechende Zeichen	
Allgemeine Entität (general entity)	ersetzt entsprechend der verfügbaren Definition	
Interne Dokument Typ Definition (internal Document Type Definition (DTD))	für Definition von Entitäten herangezogen	
	zum Validieren des Dokuments nicht herangezogen	
	nicht geliefert[1]	in einem Stück geliefert
Externe Dokument Typ Definition (external DTD)	für Definition von Entitäten herangezogen	
	zum Validieren des Dokuments nicht herangezogen	
	nicht geliefert	
Attribut mit Vorgabewert aus DTD (defaulted attribute)	wird geliefert; nicht von den Attributen unterscheid bar, die im XML-Dokument direkt an gegeben sind	wird geliefert; von den Attributen unterscheidbar, die im XML-Dokument direkt angegeben sind (Ereignis = DEFAULTED-ATTRIBUTE-NAME)
Schema	nicht unterstützt	
XML-Deklaration (text declaration)	wird interpretiert	
	nicht geliefert[2]	in drei Stücken geliefert: erst VERSION-INFORMATION, dann ENCODING-DECLARATION, zuletzt STANDALONE-DECLARATION
Steueranweisungen (processing instructions)	nicht geliefert[1]	in zwei Stücken geliefert: erst PROCESSING-INSTRUCTION-TARGET, dann PROCESSING-INSTRUCTION-DATA
Kommentare (comments)	nicht geliefert[1]	in einem Stück geliefert

CDATA-Abschnitt (CDATA sections)	Bestandteil der gelieferten Werte; nicht mehr erkennbar	in drei Stücken geliefert: erst START-OF-CDATA-SECTION, dann CONTENT-CHARACTERS, zuletzt END-OF-CDATA-SECTION
Namensraumdeklaration (namespace declaration)	nicht geliefert; bei der Verwendung in Element- bzw. Attribut-Namen jedoch interpretiert	in einem Stück geliefert
Aufgesplitterte Werte (Daten im mixed content)	zu einem einzigen Wert zusammenge- fasst	stückweise geliefert; Zerstückelung kann auch durch Entitätenreferenzen hervorgerufen werden.
Behandlung von Zeilenende, Dateisatzstruktur etc.	optionengesteuert unterdrückt oder normiert weitergegeben (betrifft externe Entitäten und Dokumente in Dateien)	optionengesteuert unterdrückt oder normiert weitergegeben (betrifft nur externe Entitäten; das Dokument selbst steht schon im Speicher.)
Wohlgeformtheit des Dokuments	Verstöße vor der Verarbeitung festge- stellt	Verstöße erst im Verlauf der Verarbei- tung erkannt
Bestimmung des Zeichensatzes in dem das Dokument vorliegt	aus der Definition des Speicherbereichs; Wenn der Speicherbereich alphanumerisch ist, wird die genaue EBCDIC-Variante aus dem Inhalt, d.h. einer ggf. vorhandenen Zeichensatzdeklaration (encoding declaration) ermittelt.	
	aus Dateiattribut und Dateiinhalt	

[1] Keine Sprachmittel für deren Beschreibung in COBOL vorgesehen

[2] Sprachmittel vorgesehen, um den Zeichensatznamen aus der Encoding Declaration zu liefern, derzeit aber noch nicht unterstützt

12.11 Testhilfen

Als Testhilfen des Compilers kann der Benutzer Testhilfezeilen und einen Übersetzungszeit-Schalter für Testhilfezeilen in Anspruch nehmen.

Testhilfezeilen

Eine Testhilfezeile ist eine Programmzeile, in der in Spalte 7 (Anzeigenbereich) der Buchstabe „D“ steht.
Jede Testhilfezeile, die zwischen Rand A und Rand R ausschließlich Leerzeichen enthält, wird als Leerzeile behandelt.
Der Inhalt einer Testhilfezeile muss syntaktisch richtig innerhalb des Programms sein, unabhängig davon, ob sie als Kommentar betrachtet wird oder nicht.
Sind alle COPY-Anweisungen bearbeitet, wird eine Testhilfezeile als Programmtext behandelt, falls ihr eine WITH DEBUGGING MODE-Klausel vorangeht. Andernfalls wird sie wie eine Kommentarzeile behandelt.
Aufeinanderfolgende Testhilfezeilen sind erlaubt.

> ¡ Testhilfezeilen sind nur im Fixed-Format erlaubt.

Übersetzungszeit-Schalter

Die im SOURCE-COMPUTER-Paragrafen angegebene WITH DEBUGGING MODE-Klausel dient als Übersetzungszeit-Schalter für die Testhilfezeilen.
Ist die WITH DEBUGGING MODE-Klausel angegeben, werden alle nachfolgenden Testhilfezeilen gemäß den oben beschriebenen Regeln übersetzt. Ist sie nicht angegeben, werden die Testhilfezeilen vom Compiler als Kommentar behandelt.

13 Segmentierung

In diesem Kapitel werden folgende Themen behandelt:

13.1 Allgemeine Beschreibung

Die COBOL-Segmentierung ermöglicht es dem Programmierer, zur Übersetzungszeit die zur Überlagerung eines Programms erforderlichen Angaben zu machen. Nur in der PROCEDURE DIVISION kann segmentiert werden. Die PROCEDURE DIVISION und die ENVIRONMENT DIVISION sind deshalb vorgesehen, die erforderlichen Angaben zur Segmentierung des Programms festzulegen.

Hinweis

Für die Erzeugung von mehrfachbenutzbarem Code ist die Segmentierung überflüssig. Mehrfachbenutzbarer Code lässt sich auf einfache Weise mit einer Steueranweisung an den Compiler erzeugen (siehe Handbuch „COBOL2000 Benutzerhandbuch“ [1]).

13.1.1 Organisation

Obwohl es nicht vorgeschrieben ist, wird die PROCEDURE DIVISION einer Übersetzungseinheit üblicherweise als eine Folge von mehreren Kapiteln geschrieben, von denen jedes aus einer Anzahl von Anweisungen besteht, die gemeinsam eine bestimmte Funktion ausführen. Wird die Segmentierung benutzt, muss die ganze PROCEDURE DIVISION in Kapitel eingeteilt werden. Zusätzlich muss bei jedem Kapitel angegeben werden, ob es zum festen Teil oder zu einem der unabhängigen Segmente des Programms gehören soll. Segmentierung ändert jedoch nichts an der Notwendigkeit, Prozedurnamen durch Kennzeichnung eindeutig zu machen.

13.1.2 Fester Teil des Programms

Unter dem festen Teil des Programms versteht man den Teil, der logisch so behandelt wird, als würde er immer resident im Speicher stehen. Dieser Teil setzt sich aus zwei Arten von Segmenten zusammen: den permanenten Segmenten und den überlagerbaren festen Segmenten.

a. Ein permanentes Segment ist ein Segment im festen Teil, das durch keinen anderen Programmteil überlagert werden kann.
b. Ein überlagerbares festes Segment ist ein Segment im festen Teil, das ein anderes Segment überlagern kann und durch ein anderes überlagerbares festes Segment oder durch ein unabhängiges Segment überlagert werden kann. Vom Ablauf her gesehen wird es jedoch als resident im Internspeicher betrachtet. Wenn ein überlagerbares festes Segment aufgerufen wird, wird es in dem Zustand zur Verfügung gestellt, in dem es zuletzt verwendet worden ist, wobei durch ALTER modifizierte Sprungziele von GO TO-Anweisungen nicht in den Initialzustand zurückgesetzt werden.

Die Anzahl der permanenten Segmente im festen Teil kann durch die SEGMENT-LIMIT-Klausel verändert werden.

13.1.3 Unabhängige Segmente

Ein unabhängiges Segment ist der Teil des Programmes, der andere Segmente überlagern kann und durch ein überlagerbares festes Segment oder durch ein anderes unabhängiges Segment überlagert werden kann. Werden Prozeduren innerhalb eines unabhängigen Segments durch eine PERFORM- oder GO TO-Anweisung außerhalb dieses unabhängigen Segments angesprochen, wird ein unabhängiges Segment dem Programm immer in seinem Initialzustand zur Verfügung gestellt. Durch ALTER modifizierte Sprungziele von GO TO-Anweisungen werden in den Initialzustand zurückgesetzt.

13.2 Allgemeine Regeln für die Segmentierung

1. Wenn die zu einem Segment gehörenden Kapitel, d.h. Kapitel mit der gleichen Segmentnummer, verstreut in der Übersetzungseinheit auftreten, ist es notwendig dass sie vom Compiler umgeordnet werden. Der Compiler gewährleistet jedoch, dass der logische Datenfluss der Übersetzungseinheit erhalten bleibt. Er fügt auch die zum Laden eines Segmentes notwendigen Anweisungen ein und/oder stellt gegebenenfalls den Initialzustand eines Segmentes wieder her. Innerhalb einer Übersetzungseinheit kann zu jedem beliebigen Paragrafen in einem Kapitel verzweigt werden, d.h. es ist nicht erforderlich, auf den Anfang eines Kapitels zu verzweigen.
2. Während des Programmablaufs bleiben nur die festen Segmente im Internspeicher resident. Sowohl die überlagerbaren festen Segmente als auch die unabhängigen Segmente können einander überlagern.
3. Die folgenden Gesichtspunkte sollten beim Einteilen der Segmente berücksichtigt werden:

   Logische Erfordernisse:

   Kapitel, die zu jeder Zeit verfügbar sein müssen oder sehr häufig aufgerufen werden, sollten als permanente Segmente eingeteilt werden. Kapitel, die weniger häufig verwendet werden, sollten entweder zu einem der überlagerten festen Segmente oder zu einem der unabhängigen Segmente gehören. Dies hängt jeweils von den logischen Erfordernissen des Programmablaufs ab.

   Beziehungen zu anderen Kapiteln:

   Kapitel, die häufig miteinander in Verbindung stehen, sollten dieselbe Segmentnummer erhalten. Alle Kapitel mit derselben Segmentnummer bilden ein Programmsegment für sich.
4. Wird die Segmentierung angewendet, sind für die ALTER-, PERFORM-, MERGE- und SORT-Anweisungen sowie aufgerufene Programme folgende Regeln zu beachten:

   ALTER-Anweisung:

   a. Eine GO TO-Anweisung in einem Kapitel, dessen Segmentnummer 50 oder größer ist, darf nicht von einer ALTER-Anweisung in einem Kapitel mit einer anderen Segmentnummer angesprochen werden.
   b. Eine GO TO-Anweisung in einem Kapitel, dessen Segmentnummer kleiner als 50 ist, darf von einer ALTER-Anweisung in jedem beliebigen Kapitel angesprochen werden, selbst dann, wenn die von der ALTER-Anweisung angesprochene GO TO-Anweisung in einem Programmsegment steht, das noch nicht zum Ablauf aufgerufen worden ist.

   PERFORM-Anweisung:

   a. Eine PERFORM-Anweisung, die innerhalb eines Kapitels geschrieben wird, des-sen Segmentnummer kleiner ist als die Segmentnummer in der SEGMENT-LIMIT-Klausel, kann innerhalb ihres Bereichs nur folgende Kapitel haben:
      - Kapitel mit Segmentnummern kleiner als 50.
      - Kapitel, die vollständig in einem einzelnen Segment mit einer Segmentnummer größer als 49 liegen.

      Der Compiler erlaubt jedoch, dass die PERFORM-Anweisung innerhalb ihres Bereichs Kapitel mit einer beliebigen Segmentnummer ansprechen kann.
   b. Eine PERFORM-Anweisung, die innerhalb eines Kapitels geschrieben wird, dessen Segmentnummer gleich oder größer ist als die Segmentnummer in der SEGMENT-LIMIT-Klausel, kann innerhalb ihres Bereichs nur folgende Kapitel haben:
      - Kapitel mit derselben Segmentnummer wie das Kapitel, das die PERFORM-Anweisung enthält.
      - Kapitel mit Segmentnummern, die kleiner sind als die Segmentnummer in der SEGMENT-LIMIT-Klausel.

      Der Compiler erlaubt jedoch, dass die PERFORM-Anweisung innerhalb ihres Bereichs Kapitel mit einer beliebigen Segmentnummer ansprechen kann.

   SORT-/MERGE-Anweisung:

a. Wird eine SORT- oder MERGE-Anweisung innerhalb eines Kapitels verwendet, das kein unabhängiges Segment ist, müssen sich die Eingabeprozeduren oder Ausgabeprozeduren, die von dieser SORT- oder MERGE-Anweisung angesprochen werden,
   - ganz innerhalb unabhängiger Segmente oder
   - ganz in einem einzigen unabhängigen Segment befinden.

b. Wird eine SORT- oder MERGE-Anweisung innerhalb eines unabhängigen Segments verwendet, müssen sich die Eingabeprozeduren oder Ausgabeprozeduren, die von dieser SORT- oder MERGE-Anweisung angesprochen werden,
   - ganz innerhalb unabhängiger Segmente oder
   - ganz in dem gleichen unabhängigen Segment wie diese SORT- oder MERGE-Anweisung befinden.

Diese Einschränkungen gelten nicht für den in diesem Handbuch beschriebenen Compiler.

Aufgerufene Programm

Ein Programm, das durch eine CALL-Anweisung aufgerufen wurde, darf die Einsprungstellen nur im permanenten Segment haben.

13.3 Sprachelemente

Es gibt zwei COBOL-Sprachelemente, um die Segmentierung zu steuern. Beide Sprachelemente sind wahlweise und sollten nur angegeben werden, wenn die Segmentierung verwendet wird:

SEGMENT-LIMIT-Klausel	angegeben im OBJECT-COMPUTER-Paragrafen in der ENVIRONMENT DIVISION
Segmentnummer	angegeben in der jeweiligen Kapitelüberschrift in der PROCEDURE DIVISION

13.3.1 Sprachelemente ENVIRONMENT DIVISION

In diesem Kapitel werden folgende Themen behandelt:

- SEGMENT-LIMIT-Klausel

13.3.1.1 SEGMENT-LIMIT-Klausel

Funktion

Die SEGMENT-LIMIT-Klausel gibt dem Anwender die Möglichkeit, die Anzahl der permanenten und überlagerbaren festen Segmente in seinem Programm zu ändern. Diese Segmente behalten dennoch die logischen Eigenschaften der Segmente des festen Teils (Segmentnummer 0 bis 49) bei, und die gesamte Anzahl der Segmente des festen Teils bleibt konstant.

Format

```
SEGMENT-LIMIT IS segmentnummer
```

Syntaxregeln

1. Die SEGMENT-LIMIT-Klausel wird im OBJECT-COMPUTER-Paragrafen angegeben. Sie ist eine wahlfreie Klausel.
2. segmentnummer muss eine vorzeichenlose, ganze Zahl mit den Werten von 1 bis einschließlich 50 sein.
3. Wenn die SEGMENT-LIMIT-Klausel angegeben ist, werden nur die Segmente als permanent betrachtet, deren Segmentnummern kleiner als segmentnummer in der SEGMENT-LIMIT-Klausel sind.
4. Die Segmente mit Segmentnummern von Segmentgrenze bis 49 werden als überlagerbare feste Segmente betrachtet.
5. Wird die SEGMENT-LIMIT-Klausel nicht angegeben, werden alle Segmente mit Segmentnummern von 0 bis 49 als permanente Segmente des Programms betrachtet.

Allgemeine Regel

1. Im Idealfall sollten alle Programmsegmente mit Segmentnummern von 0 bis 49 als permanente Segmente behandelt werden. Wenn jedoch der verfügbare Internspeicher nicht ausreicht, um alle permanenten Segmente und das größte überlagerbare Segment aufzunehmen, wird es notwendig sein, die Anzahl der permanenten Segmente zu verringern. Dies kann durch nachträgliches Einfügen einer SEGMENT-LIMIT-Klausel mit geeigneten Angaben, ohne sonstige Programmänderung, erfolgen.

Beispiel 12-1

```
SEGMENT-LIMIT IS 40
```

Diese Klausel zeigt an, dass alle Segmente mit den Segmentnummern von 0 bis 39 als permanente Segmente des Programms betrachtet werden. Segmente mit Segmentnummern von 40 bis 49 sind überlagerbare feste Segmente.

13.3.2 Sprachelemente PROCEDURE DIVISION

In diesem Kapitel werden folgende Themen behandelt:

- Segmentnummer

13.3.2.1 Segmentnummer

Funktion

Die Einteilung der Kapitel wird durch ein System von Segmentnummern verwirklicht. Die Segmentnummer ist in der Kapitelüberschrift enthalten.

Format

```
kapitelname SECTION [segmentnummer].
```

Syntaxregeln

1. Der Kapitelname benennt das Kapitel.
2. Die Segmentnummer zeigt an, ob das Kapitel zu einem permanenten, einem überlagerbaren festen oder einem unabhängigen Segment gehört.
3. segmentnummer muss eine vorzeichenlose ganze Zahl mit dem Wert 0 bis einschließlich 99 sein.
4. Alle Kapitel, die dieselbe Segmentnummer haben, bilden ein Programmsegment mit dieser Nummer.
5. Segmente mit den Segmentnummern 0 bis 49 gehören zum festen Teil des Zielprogramms. Die SEGMENT-LIMIT-Klausel zeigt an, welche dieser Segmente als permanente und welche als überlagerbare feste Segmente behandelt werden.
6. Segmente mit den Segmentnummern 50 bis 99 sind unabhängige Segmente.
7. Wird die Segmentnummer in der Kapitelüberschrift nicht angegeben, wird die Segmentnummer 0 angenommen.
8. Kapitel in den Prozedurvereinbarungen der PROCEDURE DIVISION dürfen in ihren Kapitelüberschriften keine Segmentnummern enthalten. Sie werden wie permanente Segmente mit der Segmentnummer 0 behandelt.

Allgemeine Regel

1. Wenn ein Prozedurname in einem unabhängigen Segment durch eine PERFORM-oder GO TO-Anweisung, die in einem Segment mit einer anderen Segmentnummer enthalten ist, aufgerufen wird, wird das aufgerufenen Segment für jede Ausführung der PERFORM-oder GO TO-Anweisung in seinem Initialzustand zur Verfügung gestellt. Bei einer PERFORM-Anweisung mit Wiederholungen wird der Initialzustand nur beim ersten Durchlaufen der PERFORM-Anweisung hergestellt.

14 Zusammenfassung der obsoleten Elemente

In diesem Kapitel werden die Sprachmittel zusammenfassend aufgezählt, die in neuen Programmen nicht verwendet werden sollen, weil sie von künftigen COBOL-Normen nicht mehr unterstützt werden. Die Entfernung dieser obsoleten Elemente aus alten Programmen ist ratsam.

- MEMORY SIZE-Klausel:

  Dient nur der Dokumentation (siehe Abschnitt „OBJECT-COMPUTER-Paragraf").
- MULTIPLE FILE TYPE-Klausel:

  Wird benötigt, falls sich auf einer Magnetbandspule mehr als eine Datei befindet (siehe Abschnitt „MULTIPLE FILE TAPE-Klausel").
- RERUN-Klausel:

  Zeigt an, wann und wohin Wiederanlaufpunkte auszugeben sind (siehe Abschnitt „RERUN-Klausel").
- DATA RECORDS-Klausel:

  Dient nur der Dokumentation und bezeichnet Namen der Datensätze der Datei (siehe Abschnitt „DATA RECORDS-Klausel").
- LABEL RECORDS-Klausel:

  Dient der Angabe der Namen und Werte der in einer Datei enthaltenen Kennsätze (siehe Abschnitt „LABEL RECORDS-Klausel").
- VALUE OF-Klausel:

  Vereinbart die Beschreibung von Datenfeldern eines Dateikennsatzes (siehe Abschnitt „VALUE OF-Klausel").
- ALTER-Anweisung:

  Dient der Modifizierung von GOTO-Anweisungen (siehe Abschnitt „ALTER-Anweisung").
- REVERSED-Angabe:

  Dient dem Einlesen von Datensätzen einer Datei in umgekehrter Reihenfolge (siehe Abschnitt „OPEN-Anweisung").
- STOP literal-Angabe:

  Gibt das Literal an den zugeordneten Haupt- oder Nebenbedienplatz aus, so dass nur der Anlagenbediener das Programm fortsetzen kann (siehe Abschnitt „STOP-Anweisung").
- Segmentierung:

  Segmentierung in der PROCEDURE DIVISION, um zur Übersetzungszeit die zur Überlagerung eines Programms erforderlichen Angaben zu machen (siehe Kapitel „Segmentierung").
  - SEGMENT-LIMIT-Klausel:

    Ermöglicht es, die Anzahl der permanenten und überlagerbaren festen Segmente im Programm zu ändern.
  - Segmentnummer:

    Erlaubt die Einteilung der Kapitel durch ein System von Segmentnummern.

15 Literatur

Die Handbücher sind online unter http://manuals.ts.fujitsu.com zu finden oder in gedruckter Form gegen gesondertes Entgelt unter http://manualshop.ts.fujitsu.com zu bestellen.

[1] **COBOL2000** (BS2000/OSD))**COBOL-Compiler**
Benutzerhandbuch

[2] **CRTE (BS2000/OSD)**
Common Runtime Environment
Benutzerhandbuch

[3] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

[4] **AID** (BS2000)
Advanced Interactive Debugger
Testen von COBOL-Programmen
Benutzerhandbuch

[5] **AID** (BS2000/OSD)
Testen auf Maschinencode-Ebene
Benutzerhandbuch

[6] **UDS/SQL** (BS2000/OSD)
Anwendungen programmieren
Benutzerhandbuch

[7] **openUTM** (BS2000/OSD, UNIX, Windows)
Anwendungen programmieren mit KDCS für COBOL, C und C++
Benutzerhandbuch

[8] **SORT** (BS2000/OSD)
Benutzerhandbuch

[9] **BS2000/OSD-BC**
Einführung in das DVS
Benutzerhandbuch

[10] **EDT** (BS2000/OSD)
Anweisungen
Benutzerhandbuch

[11] **COBOL85** (BS2000)
COBOL-Compiler
Tabellenheft

[12] **BS2000/OSD**
Softbooks Deutsch
CD-ROM