1 Preface

1.1 Brief product description

In most cases, the solving of commercial problems involves processing large amounts of data. COBOL is particularly well suited to this task. COBOL programs are largely independent of the particular features of individual hardware systems. The language is laid down clearly and precisely in an official document issued by the American National Standards Institute (ANSI) under the title

"American National Standard for Information Systems

- Programming Language COBOL -

ANSI X3.23-1985" and Addendum "ANSI X3.23a-1989, Intrinsic Function Module".

This is a revised version of the 1974 standard. The internal standard functions supported as of V2.1A of the compiler are described in the above Addendum.

The German standard version DIN 66028-1986 and the international standard version ISO 1989:1985 correspond to the American National Standard. The Intrinsic Functions by ANS correspond to the international norm "ISO/IEC 1989 Amendment 1, Intrinsic Function Module".

For the purpose of description, the ANSI publication divides COBOL into a nucleus and eleven functional modules, of which five are optional (Report Writer, Communication, Debug, Segmentation, Intrinsic Functions). Each of these modules in turn contains one or two functional levels. The lower level of a module is a true subset of the higher level of the same module.

The COBOL85 (BS2000) compiler corresponds to the high subset of ANS85 as regards its language set. The optional Report Writer and Segmentation language modules are also supported in accordance with the high level of ANS85. The optional Communication and Debug language modules are not supported. In BS2000, these modules are replaced by the products UTM and AID, respectively.

1.2 Target group and summary of contents

The present manual is aimed at programmers and training personnel. It is intended as a guide to the writing and maintenance of COBOL programs and as a complement to training manuals. It is neither a COBOL textbook nor a user guide.
Readers are assumed to have a sound general knowledge of programming and some basic knowledge of COBOL.

Operation of the compiler and creation of an executable COBOL program are described in the "COBOL85 User Guide" [1].

The present manual describes the COBOL language for the Siemens COBOL compiler COBOL85 Version 2.3 for the BS2000 operating system.

The manual includes all language elements which may be used when creating COBOL programs, organized according to function, format, syntax rules, general rules, and examples:

The **function** section offers a concise, general description of the individual language elements. If several formats are involved, the functional differences between them are explained in brief.

The **format** section defines the specific arrangement of character strings and separators required for a valid clause, statement, or compound structure. The occurrence of specific strings and separators and their order of appearance as shown in the format section are decisive.

The specific notation used for describing the formats is explained under the heading "General format".

Where more than one specific arrangement is permitted, the various formats are designated as "**Format 1**, **Format 2** etc.".

The **syntax rules** section describes the particular requirements and restrictions for a given function and offers additional explanations and application guidelines.

General rules describe the use of the language structure within the program context; that is, as a function of previous and subsequent as well as superior and subordinate structures and in conjunction with references and cross-references from other language elements which, strictly speaking, are independent of the described structure. Restrictions on the order of effects at program runtime are discussed. Generally speaking, these considerations are concerned with those elements which do not appear directly in the format section.

Under **Example** you will find a concrete example of the language element that has just been described.

The structure is analogous to that used for the standard COBOL document.

Certain language elements are qualified by a colors*, as follows:

Bluish green print	Siemens Nixdorf COBOL85 compiler extensions to the COBOL language standard ANS85. These include: – implementor-defined extensions – extensions from the Journal of Development (JOD) – extensions from the X/OPEN Portability Guide
Orange print	Language elements to be avoided in new programs, since they will not be supported by future COBOL standards (obsolete elements). It is advisable to remove them from old programs.

The "Contents" table gives an overview of the general structure and organization of the manual.

The "Index" enables rapid access to desired information.

The most important terms and concepts used in this manual are defined in alphabetical order in the "Glossary".

Other manuals are referred to in the text by their abbreviated titles. The full title of each publication mentioned is given at the back of the manual under "Related publications".

* The colors were chosen so as to ensure that readers who suffer from color-blindness will nevertheless be able to distinguish the colored print from normal black print.

1.3 Changes since the last version of the manual

The following table lists the major technical innovations and changes, together with the chapters or sections in which they appear.
Less important substantive and editorial changes, updated examples, text revisions, and formatting changes such as the renumbering of rules, examples and tables are scattered throughout the manual and are not specifically itemized here.

Chapter/ section	**Topic**	**New**	**Modified**
2.3.3, 10.1.9	Special register SORT-EOW	x	
2.3.7	Completion of Fig. 2-2		X
3.8.3	Extension of VALUE clause [when SET TO FALSE IS...]	X	
3.9.9	Extension of EVALUATE statement Extension of SET statement STRING statement, DELIMITED BY...	X	
7.4.5	Extension of CALL statement format 1 and 2 New format 3 -CALL UPON SYSTEM GOBACK statement	X	
10.1.8,10.2 10.3	MERGE/SORT statement Extension of "century conversion"	X	
11	Compiler directive "Source Fixed"	x	
12	Functions for working with the "century conversion"	X	

1.4 Acknowledgment

The COBOL programming language described in this manual is based on the language defined in the standard document "American National Standard for Information Systems - Programming Language - COBOL, X.3.23-1985". In recognition of the efforts made to develop and standardize COBOL, it is customary to precede a description of COBOL with the following text:

"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 herein

FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC (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 Introduction to the COBOL language

2.1 Glossary

This section contains definitions of the terms used to describe the COBOL language in this manual. These terms do not necessarily have the same meaning for other programming languages.

The definitions are brief summaries of basic characteristics. For detailed explanations and syntax rules consult the later chapters of this manual.

Access mode

The manner in which records are to be operated upon within a file.

Actual decimal point

The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Alphabetic character

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 and space.

Alphabet-name

A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

Alphanumeric character

Any character in the computer's character set.

Alphanumeric function

A function whose value is composed of a string of one or more characters from the computer's character set.

Alternate record key

A key, other than the prime record key, whose contents identify a record within an indexed file.

Area A

Columns 8 through 11 in the COBOL reference format.

Area B

Columns 12 through 72 in the COBOL reference format.

Argument

An identifier, a literal, or an arithmetic expression that specifies a value to be used in the evaluation of a function.

Arithmetic expression

An arithmetic expression can be:

– an identifier for a numeric elementary item

– a numeric literal

– two arithmetic expressions separated by an arithmetic operator

– an arithmetic expression enclosed in parentheses.

Arithmetic operator

A single character or a fixed two-character combination which belongs to the following set:

Character	Meaning
+	Addition
−	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Ascending key

A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed decimal point

A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At end condition

An at end condition may occur:

1. During execution of a sequential READ statement for a file.
2. During execution of a RETURN statement whenever there is no logical record for the sort or merge file.
3. During execution of a SEARCH statement whenever the search terminates before any of the WHEN conditions have been satisfied.

Binary search

A method of searching a table in ascending or descending order for a particular element. The search takes place by a process of halving the searched area. At each stage of the search, the middle element is compared to see whether it is greater than, less than, or equal to the element being sought. This process of halving and comparing continues until the checked element is identical to the element being sought.

Blank lines

A blank line is one that is filled entirely with blanks in columns 7 through 72 in the COBOL reference format.

Block

A physical unit of data that is normally composed of one or more logical records or a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

Body group

Generic name for a report group, control heading or control footing.

Called program

A program which is the object of a CALL statement and is combined at program run time with the calling program to produce a run unit.

Calling program

A program which executes a CALL to another program.

Character

The basic indivisible unit of the language.

Character-string

A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

Class condition

The class condition establishes whether the contents of a data item are

– completely numeric,

– completely alphabetic,

– completely uppercase,

– completely lowercase, or

– completely made up of characters defined by means of the class-name specified in the SPECIAL-NAMES paragraph of the Environment Division.

Class-name

A user-defined word specified in the SPECIAL-NAMES paragraph of the Environment Division and naming a character set defined by the user. The class-name is entered in the class condition for purposes of checking whether a data item consists entirely of characters from this character set.

Clause

A clause is an ordered set of consecutive COBOL character strings whose purpose is to specify an attribute of an entry.

COBOL character set

Character	Meaning
0 to 9	Digit
A to Z, a to z	Letter
␣	Space (blank)
+	Plus sign
–	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
:	Colon
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol

The COBOL character set consists of 77 characters.

It includes 26 uppercase letters, 26 lowercase letters, 10 digits, the space, and 15 special characters.

COBOL word

see "Word"

Collating sequence

The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging and comparing.

Column

A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

Combined condition

A condition that is the result of connecting two or more conditions with the "AND" or the "OR" logical operator.

Comment entry

An explanatory entry in the Identification Division of a source program.

Comment line

A source program line containing an asterisk (*) or slash (/) in the indicator area of the line, i.e. in column 7 of the COBOL reference format.

Any combination of characters from the computer's character set may appear in areas A and B. The comment line serves only for documentation in a program.

The asterisk indicates a comment line. A slash indicates a comment line which causes page advance before the line is printed.

Common program

A contained program in a nested source program, whose name is provided with the COMMON attribute. Such a program can be called by the directly superordinate program and also by any "sibling program" or its "descendants".

Compile time

The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Compiler directing statement

A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation. The compiler directing statements are COPY, REPLACE and USE.

Complex condition

A condition in which one or more logical operators act upon one or more conditions.

Computer-name

A system-name that identifies the computer upon which the program is to be compiled or run.

Condition

A status of a program at run time for which a truth value can be determined. In this manual, the term "condition" (condition-1, condition-2, ...) represents either a simple condition or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-name

A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of a task switch or a user switch.

Condition-name condition

Causes a conditional variable to be tested to see whether its value matches any of the values belonging to a condition-name.

Conditional expression

A simple condition or a complex condition specified in an IF, PERFORM, EVALUATE or SEARCH statement.

Conditional statement

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

Conditional variable

A data item whose value or values is or are assigned a condition-name.

Connective

A reserved word that is used to:

– associate a data-name, paragraph-name, condition-name, or text-name with its qualifier
– link two or more operands written in a series
– form conditions (logical connectives); see "Logical operator"

Contiguous items

Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

Control break

A change in the value of a data item that is referenced in the CONTROL clause.

More generally, a change in the value of a data item that is used to control the hierarchical structure of a report.

Control break level

The relative position within a control hierarchy at which the most major control break occurred.

Control data item

A data item, a change in whose contents may produce a control break.

Control data-name

A data-name that appears in a CONTROL clause and refers to a control data item.

Control footing

A report group that is presented at the end of the control group of which it is a member.

Control group

A contiguous set of data assigned to a control data item within the control hierarchy.

For a given control data item, the control group consists of the entire sequence of control headings, control footings, and their associated report groups.

Control heading

A report group that is presented at the beginning of the control group of which it is a member.

Control hierarchy

A designated sequence of report subdivisions defined by the positional order of FINAL and the data-names within a CONTROL clause.

Conversion

The implicit transformation of numeric values from one format to another, or of index values into table element numbers and vice versa.

– In the case of index values (binary numbers) and table element occurrence numbers, transformation occurs according to the formula:

  index value = (occurrence number - 1) * length of table element

  Hence, conversion depends on the table used.

– In cases where USAGEs vary from one numeric data item to another.

Counter

A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Currency symbol

The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign ($).

Current record

The record which is available in the record area associated with the file.

Current record pointer

A conceptual entity that is used in the selection of the next record.

Data clause

A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Data description entry

An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data item

A unit of data (excluding literals) defined by a COBOL program or by the rules for function evaluation.

Data-name

A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats, "data-name" represents a word which cannot be subscripted, indexed or qualified unless specifically permitted by the rules for that format.

Debugging line

A debugging line is any line with "D" in its indicator area (column 7 in the COBOL reference format).

Declaratives

A set of one or more sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative sentence

A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

De-editing

The logical removal of all editing characters from a numeric edited data item in order to determine that item’s unedited numeric value.

Delimited scope statement

Any statement which includes its explicit scope terminator.

Delimiter

A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending key

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Direct indexing

With direct indexing, the index used is in the form of a direct subscript.

see "Direct subscripting"

Direct subscripting

With direct subscripting, the subscript is indicated either as an integral literal or as a data-name described as a numeric elementary item with no character positions to the right of the assumed decimal point.

Division

A set of zero, one, or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program: Identification, Environment, Data and Procedure.

Division header

A combination of words followed by a period and a space that indicates the beginning of a division. The division headers are:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION

Dynamic access

The method of switching between sequential and random access. This method can be specified for relative or indexed files only.

Editing character

A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	Space
0	Zero
+	Plus
−	Minus
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
/	Stroke (virgule, slash)

Elementary item

A data item with no further logical subdivisions.

End program header

A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program header is:
END PROGRAM program-name.

Entry

Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

Execution time

The time at which an object program is executed.

Explicit scope terminator

A reserved word which terminates the scope of a particular Procedure Division statement.

Extend mode

The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Extended access

A method of switching to and from sequential and random access. This access method may only be specified for indexed files.

External data item

A data item which is described as part of an external record in one or more programs of a run unit and which itself may be referenced from any program in which it is described.

External record

A logical record which is described in one or more programs of a run unit and whose constituent data items may be referenced from any program in which they are described.

Figurative constant

A compiler generated value referenced through the use of certain reserved words, or a user-defined constant that may be referenced by user-assigned names.

File

A collection of records.

File clause

A clause that appears as part of any of the following Data Division entries:

File description (FD)

Sort-merge file description (SD)

Report description (RD)

File connector

The storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

File description entry

An entry in the FILE SECTION of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-name

A user-defined word that names a file described in a file description entry or a sort-merge file description entry within the FILE SECTION of the Data Division.

File organization

The permanent logical file structure established at the time that a file is created.

File position indicator

A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that the number of significant digits in the relative record number is larger than the size of the relative key data item, or that an optional input file is not present, or that the end condition already exists, or that no valid next record has been established.

Format

A specific arrangement of character-strings and separators within a statement or clause.

Function

A temporary data item whose value is determined by invoking a mechanism provided by the implementor at the time the function is referenced during the execution of a statement.

Function-identifier

A syntactically correct combination of character-strings and separators that reference a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function-identifier may include a reference-modifier. A function-identifier that references an alphanumeric function may be specified anywhere in the general formats that an identifier may be specified, subject to certain restrictions. A function-identifier that references an integer or numeric function may be referenced anywhere in the general formats that an arithmetic expression may be specified (see also section 2.4.4, "Function-identifier").

Function-name

A word that names a mechanism provided by the implementor to determine the value of a function.

Global name

A name that is declared in only one program but which can be referenced by any program contained directly or indirectly in this program. Global names can be: condition-names, data-names, file-names, record-names, report-names as well as certain special registers.

Group item

A data item that is composed of subordinate data items.

High order end

The leftmost character of a string of characters.

I-O mode

The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

I-O status

A value moved to a two-character data item to inform the COBOL program of the status of an input-output operation. This value is only moved when the FILE STATUS clause has been specified in the FILE-CONTROL paragraph.

Identifier

A syntactically correct combination of character-strings and separators that names a data item, i.e. a combination of a data-name, with the appropriate qualifiers, subscripts, and reference modifiers, as required for uniqueness of reference. Identifiers of (intrinsic) functions are described separately under the term "Function-identifier".

Imperative statement

A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

Implementor-name

A name taken from the following list:

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

*) = "reserved words" within the Environment Division.

Index

A computer storage area or register, the contents of which represent the identification of a particular element in a table.

Index data item

A data item in which the value associated with an index-name can be stored.

Index-name

A user-defined word that names an index associated with a specific table.

Indexed data-name

An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed file

A file with indexed organization.

Indexed organization

The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Indicator area

Column 7 in the COBOL reference format.

Initial program

A program that is in the initial state whenever it is called within a run unit

Initial state

The state of a program when it is first called within a run unit.

Input file

A file that is opened in the input mode.

Input mode

The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-output file

A file that is opened in the I-O mode.

Input procedure

A set of statements that is executed each time a record is released to the sort file.

Integer

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point.

Where the term "integer" appears in general formats, "integer" must be a numeric literal which is an integer, and it must be neither signed nor zero unless explicitly allowed by the rules for that format.

Integer function

A function whose category is numeric and whose definition provides that all digits to the right of the decimal point in any returned value are always set to zero.

Internal data

The data that is described in a program, excluding all external data items and external files. Data-items that are defined in the LINKAGE SECTION of a program are treated as internal data.

Internal data item

A data item that is described in a program of a run unit. An internal data item can have a global name.

Internal file

A file that can only be accessed by a program of the run unit.

Invalid key condition

A condition occurring at the time of program execution when a particular value for a key of a relative or indexed file is invalid.

Key

A data item which identifies the location of one or more data items which serve to identify the ordering of data.

Key of reference

The key, either prime or alternate, currently being used to access records within an indexed file.

Key word

A reserved word or function-name whose presence is required when the format in which the word appears is used in a source program.

Level indicator

Two alphabetic characters (FD, RD, SD, DB) that identify a specific type of file.

Level-numbers

A one or two digit number which, in the range 1 through 49, indicates the position of a data item in the hierarchical structure of a logical record or which, in the case of level-numbers 66, 77 and 88, identifies special properties of a data description entry.

Library-name

A user-defined word that identifies a source program library, which may contain more than one COBOL text with various names.

Library-text

A sequence of character-strings and/or separators in a COBOL library.

Line

see "Report line"

Line number

An integer that denotes the vertical position of a report line on a page.

Line sequential organization

A sequential file organization derived from the X/Open standard.

Literal

A character-string whose value is implied by the ordered set of characters comprising the string.

Logical operator

One of the reserved words AND, OR, or NOT.

In the formation of a combined condition, AND and OR can be used as logical connectives. NOT can be used for logical negation.

Logical record

A data item at the highest level in the hierarchy (level-number 01) which does not occur in any other record.

Low order end

The rightmost character of a string of characters.

Mass storage

A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass storage file

A collection of records that is assigned to a mass storage medium.

Merge file

A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

Mnemonic-name

A user-defined word that is associated in the SPECIAL-NAMES paragraph of the Environment Division with a specified implementor-name.

Native character set

The EBCDIC character set.

Native collating sequence

The collating sequence defined in the EBCDIC character set.

Negated combined condition

The "NOT" logical operator immediately followed by a parenthesized combined condition.

Negated simple condition

The "NOT" logical operator immediately followed by a simple condition.

Nested source program

A COBOL program that contains other programs which in turn can contain further programs. It accordingly comprises an outer program with one or more programs contained in it.

Next executable sentence

The next sentence to which control will be transferred after execution of the current statement is complete.

Next executable statement

The next statement to which control will be transferred after execution of the current statement is complete.

Next record

The record which logically follows the current record of a file.

Noncontiguous items

Elementary data items, in the WORKING-STORAGE and LINKAGE SECTIONs, which bear no hierarchic relationship to other data items.

Nonnumeric item

A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric literal

A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. Quotation marks must be doubled ("") to be represented within a nonnumeric literal.

Numeric character

A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric function

A function whose class and category are numeric.

Numeric item

A data item whose value is represented by the digits "0" through "9".

Its sign, if required, must be represented by a permitted form of "+" or "−".

Numeric literal

A literal composed of one or more numeric characters that may also contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

Object program

A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word ’program’ alone may be used in place of the phrase ’object program’.

Object time

The time at which an object program is executed.

Open mode

The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file.

The precise open mode is specified in the OPEN statement either with INPUT, OUTPUT, I-O or EXTEND.

Operand

In general, an operand may be defined as "that entity which is operated upon". However, in this publication, any lowercase word (or words) that appears in a statement, paragraph, clause or entry format may be considered an operand.

Operational sign

An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional file

A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

Optional word

A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output file

A file that is opened in either the output mode or extend mode.

Output mode

The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Output procedure

A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merged order.

Padding character

An alphanumeric character used to fill the unused character positions of a physical record.

Page

A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

Page body

That part of the logical page in which lines can be written and/or spaced.

Page footing

A report group that is presented at the end of a report page and is output prior to a page advance whenever this is caused by a page advance condition.

Page heading

A report group that is presented at the beginning of a report page and is output immediately after page advance whenever this is caused by a page advance condition.

Paragraph

In the Procedure Division:

A paragraph-name followed by a period and a space and by zero, one or more sentences.

In the Identification Division and Environment Division:

A paragraph header followed by zero, one, or more entries.

Paragraph header

A reserved word placed above the paragraphs in the Identification and Environment Divisions for identification purposes.
The permissible paragraph headers are:

In the Identification Division:

PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.

In the Environment Division:

SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.

Paragraph-name

A user-defined word that identifies a paragraph in the Procedure Division. Paragraph-names must begin in area A.

Phrase

A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical record

see "Block"

Prime record key

A key whose contents uniquely identify a record within an indexed file.

Printable group

A report group that contains at least one print line.

Printable item

A data item, the extent and contents of which are specified by an elementary report entry. This elementary report entry contains a COLUMN NUMBER clause, a PICTURE clause, and a SOURCE, SUM or VALUE clause.

Procedure

A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-name

A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified) or a section-name.

Program identification area

Columns 73 through 80 in the COBOL reference format.

Program-name

A user-defined word that identifies a COBOL source program.

Pseudo-text

A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

Pseudo-text delimiter

Two contiguous equal sign (=) characters used to delimit pseudo-text.

Punctuation character

A character that belongs to the following set:

Character	BMeaning
,	Comma
;	Semicolon
.	Period
:	Colon
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
␣	Space
=	Equal sign

Qualified data-name

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier

1. A data-name which is used in a reference together with another data-name at a lower level in the same hierarchy.
2. A section-name which is used in a reference together with a paragraph-name specified in that section.
3. A library-name which is used in a reference together with a text-name associated with that library.

Random access

An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

Record

see "Logical record"

Record area

A storage area allocated for the purpose of processing the record described in a record description entry in the FILE SECTION.

Record description entry

The total set of data description entries associated with a particular record.

Record key

A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record name

A user-defined word that names a record described in a record description entry in the Data Division.

Record number

The ordinal number of a record in the file whose organization is sequential.

Reference format

Standard method for describing a statement format in a COBOL source program.

Reference modification

Definition of a data item through specification of the leftmost character position and the length of the data item.

Reference-modifier

A syntactically correct combination of character-strings and separators that defines a unique data item. Reference modifiers consist of a delimiting left parenthesis separator, the leftmost character position at which the data item begins, a colon separator, the length of the data item, and a delimiting right parenthesis separator.

Relation

see "Relational operator"

Relation character

A character that belongs to the following set:

Character	Meaning
>	Greater than
<	Less than
=	Equal to
>=	Greater than or equal to
<=	Less than or equal to

Relation condition

A condition which can yield a truth value. A relation condition causes two operands to be compared. Either of these operands may be an identifier, a literal, or an arithmetic expression.

Relational operator

A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meaning are:

Character	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to
IS GREATER THAN OR EQUAL TO IS >=	Greater than or equal to
IS LESS THAN OR EQUAL TO IS <=	Less than or equal to

Relative file

A file with relative organization.

Relative indexing

With relative indexing, the name of the table element is followed by an index in the form (index name +|− integer).

Relative key

A key whose contents identify a logical record in a relative file.

Relative organization

The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Relative record number

The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

Relative subscripting

With relative subscripting, the name of the table element is followed by a subscript in the form

```
(data-name + integer) or
(data-name - integer).
```

Report clause

A clause, in the REPORT SECTION of the Data Division, that appears in a report description entry or a report group description entry.

Report description entry

An entry in the REPORT SECTION of the Data Division that is composed of the level indicator RD, followed by a report name, followed by a set of report clauses as required.

Report file

An output file whose file description entry contains a report clause. The contents of a report file consist of records that are written under control of the Report Writer Control System.

Report footing

A report group that is presented only at the end of a report.

Report group

In the REPORT SECTION of the Data Division, a 01-level entry and its subordinate entries.

Report group description entry

An entry in the REPORT SECTION of the Data Division that is composed of the level-number 01, the optional data-name, a TYPE clause, and an optional set of report clauses.

Report heading

A report group that is presented only at the beginning of a report.

Report line

A division of a page representing one row of horizontal character positions.

Report-name

A user-defined word that names a report described in a report description entry within the REPORT SECTION of the Data Division.

Report Writer logical record

A record that consists of the Report Writer print line and associated control information necessary for its selection and vertical positioning.

Reserved word

A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Run unit

A particular set of object programs which function as a unit at object time.

Section

A section comprises a set of paragraphs or clauses. The contents are preceded by a section header. A section can be empty or contain one or more paragraphs or clauses.

Section header

A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data and Procedure Division. In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

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

In the Data Division:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
REPORT SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-name

A user-defined word which names a section in the Procedure Division.

Segment-number

A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters "0", "1", ..., "9". A segment-number may be expressed either as a one or two digit number.

Sentence

A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator

A character used to separate character-strings.

Sequence number area

Columns 1 through 6 in the COBOL reference format.

Sequence of programs

A set of COBOL source programs which are compiled with a single compiler call. Each source program in the sequence must be terminated with an end program header.

Sequential access

An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor sequence determined by the order of records in the file.

Sequential file

A file with sequential organization.

Sequential organization

A permanent logical file structure in which the records are arranged and read in the same order in which they were created.

Sign condition

The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Simple condition

Any single condition chosen from the set:

Relation condition

Class condition

Condition-name condition

Switch-status condition

Sign condition

Sort file

A collection of records to be sorted by a SORT statement.

The sort file is created and can be used by the sort function only.

Sort-merge file description entry

An entry in the FILE SECTION of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

Source program

A syntactically correct set of COBOL statements beginning at the Identification Division, a COPY statement or a REPLACE statement. The end of a source program is indicated by an end program header or by the absence of further source program lines.

Special character

A character that belongs to the following set:

Character	Meaning	Character	Meaning
+	Plus sign	.	Period (decimal point)
−	Minus sign	:	Colon
*	Asterisk	"	Quotation mark
/	Stroke (virgule, slash)	(	Left parenthesis
=	Equal sign	)	Right parenthesis
$	Currency sign	>	Greater than symbol
,	Comma (decimal point)	<	Less than symbol
;	Semicolon		

Special-character word

A reserved word which is an arithmetic operator or a relation character.

Special registers

Compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

Statement

A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Subprogram

see "Called program"

Subscript

An integer, a data-name or an arithmetic expression whose value identifies a particular element in a table or one of the data items subordinate to this element.
A subscript may be the word ALL when the subscripted identifier is used as a function argument.

Subscripted data-name

An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Sum counter

A signed numeric data item established by a SUM clause in the REPORT SECTION of the Data Division. The sum counter is used by the Report Writer in connection with summing operations.

Switch-status condition

A condition which indicates whether a user or task switch has been set to "on" or "off". The test is positive if the status of the switch corresponds to the setting given in the condition-name.

Symbolic character

A user-defined word indicating a figurative constant defined by the user.

System name

A COBOL word which is used to communicate with the operating system.

Table

A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table element

A data item that belongs to the set of repeated items comprising a table.

Text name

A user-defined word which identifies library text.

Text-word

A character or sequence of contiguous characters between margin A and margin R in a COPY library, a source program, or a pseudo-text. Text-words are:

1. Separators, except for: space, pseudo-text delimiters, and the opening and closing delimiters for nonnumeric literals.
2. Literals including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark which bound the literal. A string within a nonnumeric literal is not a separate text-word.
3. Any other sequence of characters delimited by separators, except comment lines and the word "COPY", bounded by separators.

Truth value

The representation of the result of the evaluation of a condition in terms of one of two values, "true" or "false".

Unary operator

A plus (+) or a minus (−) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1 respectively.

User-defined word

A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable

A data item whose value may be changed during execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb

A COBOL word that causes an action to be taken by the COBOL compiler and the object program.

Word

A character-string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word, or a function-name.

2.2 COBOL notation

1. Definition of a format

The specific arrangement of the elements within a clause or statement is referred to as a "general format". A clause or statement may be composed of various element types.

When more than one specific arrangement is permitted in a clause or statement, the general format is subdivided into numbered formats. Note that clauses must be written in the same sequence in which they are specified in the general format. In certain exceptional cases, departures from this rule are allowed. These cases, however, are identified as such.

The proper use of formats, the necessary application prerequisites, and the restrictions on their use are expressed in the form of rules.

2. Elements

Clauses or statements may be constructed from the following element types:

– uppercase words
– lowercase words
– uppercase and lowercase words
– level-numbers
– brackets
– braces
– connectives
– special characters

3. Words

Notation	Meaning
Uppercase	A word specially reserved for COBOL.
Uppercase, underlined	This word must be specified by the programmer as it is given in the format. It is a COBOL keyword.
Uppercase, not underlined	This word may be specified by the programmer at the location given in the format or it may be omitted. It is an optional COBOL word.
Lowercase	Generic term used to represent COBOL words, literals, picture-strings, comments, or a complete syntactical unit. It must be entered by the programmer at the the location given in the format. If more than one generic term of the same kind occurs in the same format, an appended number or letter is used to uniquely qualify that term for the descriptions.

Table 2-1: Notation used for COBOL words

An entry consisting of one or more words in uppercase followed by the words "clause" or "statement" designates a clause or statement described elsewhere in this manual. In programs, all COBOL words can appear in uppercase and lowercase as well as in lowercase only.

4. Separators

The separators listed in the following table must be used as specified in the format.

Character	Meaning	Character	Meaning
␣	Space	"	Quotation marks *)
,	Comma	(	Open parentheses
;	Semicolon	)	Close parentheses
.	Period	==	Pseudo-text delimiter
:	Colon		

Table 2-2: Separators

*) The predefined COBOL quotation mark is the double quote ("). To enable formal acceptance by the compiler of old COBOL programs in which the quotation mark is represented by the single quote or apostrophe ('), it is necessary to use a special compiler option (for details see the relevant section in the "COBOL85 User Guide" [1]).

The rules governing the use of separators are described in section 2.3.2.

5. Level indicators and level numbers

Level indicators and level numbers which occur in the format must be supplied at the appropriate point in the COBOL source program. This manual uses the form 01, 02, ..., 09 to indicate level numbers 1, 2, ..., 9.

6. Brackets [ ]

A format specification placed in square brackets may be supplied or omitted at the option of the user. If two or more items are stacked within brackets, one or none of them may be specified.

7. Braces { }

If two or more items are stacked within braces, one of the enclosed items is required. If there is only one item, the braces perform only a combining function for a subsequent ellipsis (repetition symbol).

8. Parentheses ( )

Format items appearing within parentheses refer to table item numbers (indices) which must be specified in order to differentiate the various items in a table.

9. Ellipsis ...

An ellipsis appearing in the text indicates the omission of one or more words when such an omission does not impair comprehension.

An ellipsis appearing in the format indicates that the immediately preceding unit may, if desired, be repeated any number of times after it has been specified once. A repeatable unit is either a single word or a group of words combined by brackets or braces. In the latter case, the ellipsis immediately follows the closing bracket or brace; the related opening bracket or brace determines the beginning of the unit to be repeated.

10. Space ␣

When used in examples and tables, this character refers to a space.

11. Special characters in formats

If the characters '+', '−', '>', '<', '=', '>=', '<=' appear in a format, they must be entered whenever the format is used. This applies even if these special characters are not underlined.

Example 2-1

```
          (2)             (2)
(1)        ⎧identifier-1  ⎫ (3)
ADD        ⎨              ⎬ ...
           ⎩literal-1     ⎭

                          (4) (5)     (5)
           TO {identifier-2   [ROUNDED] }...

         (6)                      (7)
        [ON SIZE ERROR  imperative statement-1]

        [NOT ON SIZE ERROR imperative statement-2]

        [END-ADD]
```

(1) COBOL keyword: the indicated form is mandatory.

(2) Braces: one of these options must be chosen.

(3) Ellipsis: the preceding entry may be repeated any number of times.

(4) Qualification: an appended number or letter is used to create a unique identification for an element.

(5) Brackets: one or more of these options may be chosen.

(6) Optional word: the word may be omitted, or specified for the sake of clarity.

(7) Lowercase words: these must be entered by the programmer.

The following language elements are valid ADD statements derived from the above format; commas and semicolons are included for better readability.

```
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.
```

2.3 Language concepts

2.3.1 COBOL character set

The basic linguistic unit is the character. The COBOL character set consists of 77 characters, including 26 uppercase letters, 26 lowercase letters, 10 digits, the space character, and 15 special characters.

Character	Meaning
0 to 9	Digit
A to Z, a to z	Letters
␣	Space
+	Plus
–	Minus (hyphen)
*	Asterisk
/	Slash
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
:	Colon
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than

Table 2-3: COBOL character set

When nonnumeric literals, comments or comment lines are used, the character set is extended to comprise the whole set of characters of the data processing system.

The characters which are permitted for use with each type of string and as separators (delimiters) are defined in the subsections to follow.

2.3.2 Separators

A separator is a character or two contiguous characters formed according to the following rules:

1. A space is a separator. Anywhere where a space can be used as a separator or as part of a separator, it is also possible to use more than one space.

2. Commas and semicolons can only be used as separators when they are immediately followed by a space. They can be used to improve the readability of the program anywhere where a space could also be used as a separator. A comma, on the other hand, is not a separator if it is used in a PICTURE character-string.
3. A period can only be used as a separator when immediately followed by a space. It may only be used to indicate the end of a sentence, or as shown in formats.
4. Left (opening) and right (closing) parentheses are separators. When used outside of pseudo-text, they must appear as balanced pairs of left and right parentheses used to delimit subscripts, reference modifiers, arithmetic expressions, conditions or the repetition factor of a PICTURE symbol.
5. The quotation mark is a separator. An opening quotation mark must be immediately preceded by a space, a left parenthesis or an opening pseudo-text delimiter. A closing quotation mark assigned to an opening quotation mark must be immediately followed by one of the separators space, comma, semicolon, period, right parenthesis or closing pseudo-text delimiter. These immediately preceding and following separators are not part of the separator quotation mark.
6. Pseudo-text delimiters are separators. An opening pseudo-text delimiter must be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semicolon or period.
   Pseudo-text delimiters may only appear in balanced pairs delimiting pseudo-text.
7. A colon is a separator and must be specified when required in the general formats.
8. A space used as a separator may immediately precede all separators unless
   a) the reference format rules prohibit it
   b) it is followed by the closing quotation mark; in this case, a preceding space is considered part of the nonnumeric literal and not as a separator.
9. A space used as a separator may immediately follow any separator except the opening quotation mark. A space following the opening quotation mark is considered part of the nonnumeric literal and not as a separator.
10. Any character which is part of a PICTURE character-string or of a nonnumeric literal is not treated as a separator.
11. PICTURE character-strings are delimited exclusively by the separators space, comma, semicolon and period.
12. The rules governing the formation of separators do not apply to characters contained in nonnumeric literals, comment-entries or comment lines.

2.3.3 COBOL words

A word consists of 1-30 characters from the following set:

A-Z, a-z, 0-9, – (hyphen)

No distinction is made between uppercase and lowercase letters.

A word may neither begin nor end with a hyphen, must not contain space characters, and must contain at least one letter.

Words are divided into four categories:

– user-defined words

– system-names

– reserved words

– function-names

1. User-defined words

A user-defined word is a COBOL word to be supplied by the programmer according to the format for a clause or statement. It refers to particular units of data at object time. The following subsections describe the types of user-defined words employed in COBOL programs, and state the rules for writing these names.

The 17 types of user-defined words are listed and defined in Table 2-4.

All user-defined words except segment-numbers and level-numbers must be made unique. Either there must be no other user-defined word in the source program with the same sequence of characters and punctuation marks, or the word must be qualified.
With the exception of paragraph-name, section-name, level-number, and segment-number, all user-defined words must contain at least one alphabetic character. Segment-numbers or level-numbers may be identical to other segment-numbers or level-numbers, or to paragraph-names and section-names.

alphabet-name	An alphabetical name located in the SPECIAL-NAMES paragraph of the Environment Division and connected with a character set and/or collating sequence.
class-name	A name entered by the user in the CLASS clause of the SPECIAL-NAMES paragraph in the Environment Division to define a character set. This class-name can be referenced in the class condition.
condition-name	The name assigned to a specific value, set of values, or range of values which an elementary data item may assume (hence, a condition of the data item). A condition-name is defined by an 88-level entry in the FILE, LINKAGE or WORKING-STORAGE SECTION.

Table 2-4: COBOL user-defined words

data-name	A name identifying a data item in the Data Division. A data-name is defined by its appearance in a data description entry. A special data-name is an index data-name designating an index data item. An index data item is a data item whose description contains the USAGE IS INDEX clause.
file-name	A name assigned to a set of input data or output data. A file-name is defined by its appearance in the SELECT clause of the FILE CONTROL paragraph and its use as the name of an FD entry. A special file-name is a sort-file-name that names a sort-file. A sort-file-name is defined by its appearance in the SELECT clause of the FILE CONTROL paragraph and its use to name an SD entry in the FILE SECTION.
index-name	A name of an index for a particular table. An indexname is declared by its occurrence in the INDEXED BY phrase of the OCCURS clause.
level-number	A level-number indicates the position of a data item in the hierarchical structure of a record or indicates special properties of a data description entry. Level-numbers are defined by their appearance in a data description entry.
library-name	A name of an entry in the COBOL source program library. The library may contain more than one text with various names.
mnemonic-name	A fixed name, provided the programmer associated it with a particular implementor-name in the SPECIAL-NAMES paragraph of the Environment Division.
paragraph-name	A paragraph-name is used to name a paragraph in the Procedure Division. Paragraph-names are written starting at Area A.
program-name	The name used to identify the program. The program-name is defined by its use in the PROGRAM-ID paragraph of the Identification Division. It may also appear in a CALL statement of a corresponding calling program.
record-name	The name of a record. A record is declared by a 01-level entry in the FILE SECTION, LINKAGE SECTION, WORKING-STORAGE SECTION or SUB-SCHEMA SECTION.
report-name	The name of a report. A report-name is defined by its occurrence in the REPORT clause of an FD entry; it is used to name an RD entry in the REPORT SECTION.
section-name	A section-name is used to name a section in the Procedure Division. A section-name is written starting at Area A and is followed by the word SECTION.
segment-number	A number to classify sections in the Procedure Division for purposes of segmentation. It is defined by its use in a section header.
symbolic-character	A name for a figurative constant defined by the user character in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph.
text-name	Name of an entry in the COBOL source program library. The entry is copied from the library by the COPY statement.

Table 2-4: COBOL user-defined words

2. System-names

A system-name is a COBOL word which is used as an interface with the operating system environment. System-names are defined by the implementor and may vary from compiler to compiler. From the programmer's point of view, the system-names of a specific compiler are treated as reserved words.

The system names for COBOL85 are:

Computer-name	in the SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs.
Implementor-name	in the SPECIAL-NAMES paragraph and the ASSIGN clause.

3. Reserved words

COBOL includes a fixed number of reserved words, the COBOL words.

A reserved word serves a specific purpose and must be used only in the context specified in the formats; it must not occur in the source program as a user-defined word or system-name.

A complete list of reserved words is supplied on page 57ff. All reserved words marked with an asterisk (*) in this list are treated as reserved words only if DML (Data Manipulation Language) statements are being used for compilation; otherwise they may be employed as user-defined words. Compilation with DML statements occurs when

SUB-SCHEMA SECTION

is specified in the Data Division of a program (see the "UDS Reference Manual" [6]).

There are three types of reserved words:

– Required words
– Optional words
– Special purpose words

- **Required words**

  A required word is a word whose presence is required when the format in which the word appears is used in a source program.

  Required words are of two types:

Keywords

Within each format, such words are uppercase and underlined. Keywords are only allowed in the formats indicated. Keywords may be grouped as shown below:

– Verbs such as ADD, READ and CALL.

– Required words which are encountered in statement and entry formats.

– Words which have a specific functional significance, such as NEGATIVE, SECTION, etc.

Some keywords may be abbreviated (e.g. PIC for PICTURE).

Special character words

These are the arithmetic operators and relation characters (see section 2.1, "Glossary").

- **Optional words**

  Within each format, uppercase words which are not underscored are called "optional words". These words may be used at the option of the user. The presence or omission of an optional word has no effect on the meaning of the COBOL statement. However, an optional word must not be misspelled or replaced with another word.

- **Special purpose words**

  There are two types of special purpose words:

  – special registers

  – figurative constants

Special registers

Special registers are data items in which information produced with the use of certain COBOL features is stored. The attributes of these registers are predefined, and each register has a fixed name. Thus, the programmer does not have to define these registers in the Data Division. The eleven special registers are listed in Table 2-5.

Register name	Description	Use
TALLY	5-digit unsigned data item with COMPUTATIONAL phrase (see "USAGE clause", page 192)	TALLY may be used wherever a data item with an integral value can occur. For example, if the current value of TALLY is 3, the following statementsare equivalent: ADD 3 TO ALPHA. ADD TALLY TO ALPHA.
LINE-COUNTER PAGE-COUNTER PRINT-SWITCH CBL-CTR	Used by the Report Writer (see chapter 8, "Report writer").	See chapter 8, "Report writer".
LINAGE-COUNTER	A 4-byte data item containing an unsigned integer whose value is less than or equal to integer-1 or the data item referenced by dataname-1 in the LINAGE clause	A LINAGE-COUNTER register is generated by the compiler for each file whose file description entry contains a LINAGE-clause (see "LINAGE clause", page 386).
RETURN-CODE	8-digit signed data item with COMPUTATIONAL and SYNCHRONIZED phrase (corresponds to PIC S9(8) COMP-5 SYNC).	This data item exists only once for each program system. The user can use this item to exchange information between COBOL modules which were compiled separately but linked into a single object porgram. This item can also be used to store the return value of a non-COBOL subprogram. When a COBOL subprogram terminates, the contents of the item can be made available to the calling non-COBOL program as a function value. If the contents of the RETURN-CODE special register are not 0 after the execution of STOP RUN, the operating system is informed that the program terminated abnormalley.
SORT-RETURN SORT-FILE-SIZE SORT-CORE-SIZE SORT-MODE-SIZE SORT-EOW	Used by the sort section (see chapter 10, "Sorting of records").	See chapter 10, "Sorting of records".

Table 2-5: COBOL special registers

Figurative constants

The values of figurative constants are produced by the compiler and are indicated by the reserved words listed in Table 2-6. Figurative constants must not be enclosed in quotation marks. The singular and plural forms of a figurative constant are equivalent and may be used optionally.

The figurative constant [ALL] symbolic-character stands for one or more of the characters specified as the value of symbolic-character in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph.

If a figurative constant represents a string of one or more characters, the compiler determines the length of the string according to the following rules:

1. If a figurative constant is specified in a VALUE clause or associated with another data item (e.g. moved to or compared with another data item), it is first duplicated to the right until the resultant string has at least as many character positions as the other data item.

   If this character-string has more character positions than the other data item following the duplication operation, the extra positions will be truncated from the right.

   Extension or truncation of the character-string of figurative constants takes place prior to and independently of any application of the JUSTIFIED clause to the other data item.

2. The character-string always has a length of 1 whenever the figurative constants ZERO, SPACE, HIGH-VALUE, LOW-VALUE and QUOTE (including their plurals) are not brought into contact with another data item, particularly whenever they occur in a DISPLAY, STOP, STRING or UNSTRING statement.

3. If the figurative constant ALL literal is not brought into contact with another data item, the length of the character-string is equal to the length of literal.

A figurative constant can be used wherever literal occurs in a format, except for the following cases:

1. If literal is restricted to numeric literals, the only figurative constant allowed is ZERO (ZEROS, ZEROES).

2. The figurative constant ALL literal cannot be brought into contact with numeric or numeric-edited data items.

3. Apart from its use in the figurative constant ALL literal, the word ALL has no function; it serves only to enhance readability.

If the figurative constants HIGH-VALUE[S] or LOW-VALUE[S] are used in a source program (except for the ALPHABET clause), the character currently associated with this constant is dependent on the collating sequence defined for the program and belonging to the character set (see "OBJECT-COMPUTER paragraph", page 124 and "SPECIAL-NAMES paragraph", page 125).

Each reserved word used to assign a value to a figurative constant constitutes a character-string of its own; if the word ALL is used it constitutes two character-strings.

If alphabet-name-2 is specified in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph or in the CODE-SET clause of a data description entry (see "CODE-SET clause", page 382), the character code type is defined by the ALPHABET clause.

If the IN phrase is omitted, symbolic-character-1 stands for the character whose position within the collating sequence of the hardware-specific character set is indicated by integer-1.

If the IN phrase is used, integer-1 refers to the character set named by alphabet-name-2.

The internal representation of symbolic-character-1 is identical to that of the corresponding character in the hardware-specific character set.

Table 2-6 lists the figurative constants and indicates the values they represent.

Figurative constant	Corresponding value	Example *)
[ALL] ZERO or [ALL] ZEROS or [ALL] ZEROES	One or more occurrences of the character 0 (X' F0') or binary zero (X' 00'), depending on the description of the data item.	Statement: MOVE ZEROS TO FIELD. Contents of FIELD: – If FIELD is a binary item: X' 00000000' – If FIELD is an external decimal item: X' F0F0F0F0' (= C' 0000') – If FIELD is an internal decimal item: X' 0000000F' .
[ALL] SPACE or [ALL] SPACES	One or more occurrences of the character space (X' 40').	Statement: MOVE SPACE TO FIELD. Contents of FIELD: X‘40404040‘(= C‘␣␣␣␣‘)
[ALL] HIGH-VALUE or [ALL] HIGH-VALUES	With COLLATING SEQUENCE unspecified: One or more occurrences of the character that has the highest value in the EBCDIC collating sequence (X' FF').	Statement: MOVE HIGH-VALUE TO FIELD. Contents of FIELD: X‘FFFFFFFF‘(=C‘~~~~‘)
	With COLLATING SEQUENCE specified: The character with the highest position in the program collating sequence.	Entry in SPECIAL-NAMES paragraph: ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 194. The highest position belongs to the character at the 194th position of the EBCDIC character set, i.e. the character A. A is assigned to HIGH-VALUE.
[ALL] LOW-VALUE or [ALL] LOW-VALUES	With COLLATING SEQUENCE unspecified: One or more occurrences of the character that has the lowest value in the EBCDIC collating sequence (X' 00').	Statement: MOVE LOW-VALUE TO FIELD. Contents of FIELD: X' 00000000'
	With COLLATING SEQUENCE specified: The character with the lowest position in the program collating sequence.	Entry in SPECIAL-NAMES paragraph: ALPHABET ALPHATAB IS "0" "1" "2". The lowest position belongs to the character 0. 0 is assigned to LOW-VALUE.

Table 2-6: COBOL figurative constants and values

Figurative constant	Corresponding value	Example *)
[ALL] QUOTE or [ALL] QUOTES	One or more occurrences of the quotation mark (X' 7F'). Note: The word QUOTE (QUOTES) cannot be used in place of a quotation mark to enclose a nonnumeric literal.	Data description entry: 02 FIELD PIC X VALUE QUOTE. Contents of FIELD: X'7F' or X'7D', depending on the current quotation mark (see COBOL character set, page 46).
ALL literal	One or more occurrences of the string of characters composing the literal. The literal must be nonnumeric.	Statement: MOVE ALL "A" TO ALPHA. Contents of ALPHA: C' AAAA' Statement: MOVE ALL "12" TO ALPHA. Contents of ALPHA: C' 1212' Statement: MOVE ALL "ABC" TO ALPHA. Contents of ALPHA: C' ABCA'
[ALL] symbolic-character	One or more repetitions of the character specified as the value of symbolic-character in the SYMBOLIC-CHARACTERS clause of the SPECIAL-NAMES paragraph.	Description: SYMBOLIC C0 IS 193 Statement: MOVE ALL C0 TO ALPHA. Contents of ALPHA: X' C0C0C0C0'

Table 2-6: COBOL figurative constants and values

*) In these examples it is assumed that, unless otherwise specified, ALPHA is a 4-byte area with the data format DISPLAY.

The following table contains all the reserved words.

All words marked with * are treated as reserved words only if DML (Data Manipulation Language) statements are being used for compilation; otherwise they may be employed as user-defined words. Compilation with DML statements occurs when SUB-SCHEMA SECTION is specified.

```
<                       *CASE              *CURRENT               END-ACCEPT
<=                       CBL-CTR            DATA                  END-ADD
+                        CD                *DATABASE-EXCEPTION    END-CALL
*                        CF                 DATABASE-KEY          END-COMPUTE
**                       CH                 DATABASE-KEY-LONG     END-DELETE
-                        CHARACTER          DATE                  END-DISPLAY
/                        CHARACTERS         DATE-COMPILED         END-DIVIDE
>                        CHECKING           DATE-WRITTEN          END-EVALUATE
>=                       CLASS              DAY                   END-IF
:                        CLOCK-UNITS        DAY-OF-WEEK           END-MULTIPLY
=                        CLOSE             *DB                    END-OF-PAGE
ACCEPT                   CODE               DE                    END-PERFORM
ACCESS                   CODE-SET           DEBUGGING             END-READ
ADD                      COLLATING          DEBUG-CONTENTS        END-RECEIVE
ADVANCING                COLUMN             DEBUG-ITEM            END-RETURN
AFTER                    COMMA              DEBUG-LINE            END-REWRITE
ALL                      COMMIT             DEBUG-NAME            END-SEARCH
ALPHABET                 COMMON             DEBUG-SUB-1           END-START
ALPHABETIC               COMMUNICATION      DEBUG-SUB-2           END-STRING
ALPHABETIC-LOWER         COMP               DEBUG-SUB-3           END-SUBTRACT
ALPHABETIC-UPPER         COMP-1             DECIMAL-POINT         END-UNSTRING
ALPHANUMERIC             COMP-2             DECLARATIVES          END-WRITE
ALPHANUMERIC-EDITED      COMP-3             DELETE                ENDING
ALSO                     COMP-5             DELIMITED             ENTER
ALTER                    COMPUTATIONAL      DELIMITER             ENTRY
ALTERNATE                COMPUTATIONAL-1    DEPENDING             ENVIRONMENT
AND                      COMPUTATIONAL-2    DESCENDING            ENVIRONMENT-NAME
ANY                      COMPUTATIONAL-3    DESTINATION           ENVIRONMENT-VALUE
ARE                      COMPUTATIONAL-5    DETAIL                EOP
AREA                     COMPUTE            DISABLE               EQUAL
AREAS                    CONFIGURATION      DISC                 *ERASE
ARGUMENT-NUMBER         *CONNECT           *DISCONNECT            ERROR
ARGUMENT-VALUE           CONSOLE            DISPLAY               ESI
ASCENDING                CONTAINS           DIVIDE                EVALUATE
ASSIGN                   CONTENT            DIVISION              EVERY
AT                       CONTINUE           DOWN                  EXCEPTION
AUTHOR                   CONTROL           *DUPLICATE            *EXCLUSIVE
                         CONTROLS           DUPLICATES            EXIT
BEFORE                   CONVERTING         DYNAMIC               EXTEND
BEGINNING                COPY                                     EXTENDED
BINARY                   CORR               EBCDIC                EXTERNAL
BLANK                    CORRESPONDING      EGI
BLOCK                    COUNT              ELSE                  FALSE
BOTTOM                   CREATING           EMI                   FD
BY                       CSP               *EMPTY                *FETCH
CALL                     C01...C11          ENABLE                FILE
CANCEL                   CURRENCY           END                   FILE-CONTROL
```

FILE-LIMITS
FILLER
FINAL
*FIND
*FINISH
FIRST
FOOTING
FOR
*FREE
FROM
FUNCTION

GENERATE
*GET
GIVING
GLOBAL
GO
GOBACK
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

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

JUST
JUSTIFIED

*KEEP
KEY
KEY-YY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
*LIMITED
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

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

NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
NUMERIC-EDITED

OBJECT-COMPUTER
*OCCURENCE
OCCURS
OF
OFF
OMITTED
ON
*ONLY
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
*OWNER

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

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
*READY
*REALM
*REALM-NAME
RECEIVE
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REFERENCE
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPEATED
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
*RESULT
*RETAINING
*RETRIEVAL
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROLLBACK
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
*SELECTIVE
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET

*SET-SELECTION
*SETS
SIGN
SIZE
SORT
SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-MERGE
SORT-MODE-SIZE
SORT-RETURN
SORT-TAPE
SORT-TAPES
*SORTED
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STATUS
STOP
*STORE
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
*SUB-SCHEMA
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED
SYSIPT
SYSOPT
*SYSTEM

TABLE
TALLY
TALLYING
TAPE
TAPES
*TENANT
TERMINAL
TERMINATE
TEST
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TRUE
TRY
TSW-1...TSW-31
TYPE

UNIT
UNITS
UNSTRING
UNTIL
UP
*UPDATE
UPON
USAGE
*USAGE-MODE
USE
USING
USW-1...USW-31

VALUE
VALUES
VARYING
*VIA

WHEN
WITH
*WITHIN
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

Optionally reserved words

If the compiler option MARK-NEW-KEYWORDS is set, the following words are marked to indicate new `reserved words` in the future standard. For the words in *italics*, this applies only within a certain context.

ABSENT	END-INVOKE	*LOCALE*	*REVERSE-VIDEO*
ACTIVE-CLASS	*EOL*	*LOCALIZE*	
ADDRESS	*EOS*	LOCALE-STORAGE	SCREEN
ALIGNED	*ERASE*	*LOWLIGHT*	*SECONDS*
ALLOCATE	EXCEPTION-OBJECT		*SECURE*
ALLOW	*EXPANDS*	*MANUAL*	SELF
ARITHMETIC		METHOD	SHARING
AS	FACTORY	METHOD-ID	*SIGNED*
ATTRIBUTE	FLOAT-LONG		SOURCES
AUTO	FLOAT-SHORT	NATIONAL	STANDARD-3
	FOREGROUND-COLOR	NATIONAL-EDITED	*STEP*
B-AND	FORMAT	*NONE*	*STRONG*
B-NOT	FREE[1)]	*NORMAL*	SUPER
B-OR	*FULL*	NULL	*SYMBOL*
B-XOR	FUNCTION-ID	*NUMBERS*	
BACKGROUND-COLOR			TIMEOUT
BELL	GET[1)]	OBJECT	TYPEDEF
BINARY-CHAR		*ONLY*	
BINARY-DOUBLE	*HIGHLIGHT*	OPTIONS	*UCS-2*
BINARY-LONG		OVERRIDE	*UCS-4*
BINARY-SHORT	*IGNORING*		*UNDERLINE*
BIT	INHERITS	*PARAGRAPH*	UNIVERSAL
BLINK	*IINITIALIZED*	PRESENT	UNLOCK
BOOLEAN	INTEGER	*PREVIOUS*	*UNSIGNED*
	INTERFACE	PROGRAM-POINTER	*UTF-8*
CALL-CONVENTION	INTERFACE-ID	PROPERTY	*UTF-16*
CENTER	*INTRINSIC*	PROTOTYPE	
CLASS-ID	INVOKE		VALID
COL		RAISE	VALIDATE
COLS	*LC-ALL*	RAISING	
COLUMNS	*LC-COLLATE*	*RECURSIVE*	*YYYYDDD*
CONSTANT	*LC-CTYPE*	*RELATION*	*YYYYMMDD*
CRT	*LC-CURRENCY*	REPOSITORY	
CURSOR	*LC-MESSAGES*	*REQUIRED*	
CYCLE	*LC-MONETARY*	RESERVED	
	LC-NUMERIC	*RETRY*	
DEFAULT	*LC-TIME*	RETURNING	

[1] Previously only DML keyword

4. Function-names

A function-name is a word that is one of a specified list of words which may be used in COBOL source programs. The same word, in a different context, may appear in a program as a user-defined word (see "Function-name" in section 12.1).

2.3.4 Literals

A literal is a character-string whose value is determined by the characters of which it is composed; alternatively, the string may represent a reserved word which corresponds to a figurative constant. Literals are either numeric or nonnumeric.

1. Nonnumeric literals

A nonnumeric literal is a character-string consisting of from 1 to 180 characters, enclosed in quotes. The value of the nonnumeric literal in the object program is the sequence of individual characters itself, without delimiting quotes. The literal may contain any characters from the EBCDIC character set, except quotation marks. To represent a quotation mark within a literal, two contiguous quotation mark characters (") must be used.

Example 2-2

```
"CHARACTER"
"153.78"
"ADAM ""BDAM"" CDAM"
```

2. Numeric literals

There are two types of numeric literals: fixed-point literals and floating-point literals.

- **Numeric fixed-point literals**

  A fixed-point numeric literal is a string of characters chosen from the following set: the digits 0-9, the plus sign, the minus sign, and the decimal point.

  Fixed-point numeric literals must be formed according to the following rules:

  1. The literal may contain 1 to 18 digits.
  2. The literal may contain only one sign character. If a sign is used, it must be the leftmost character of the literal. An unsigned literal is assumed to be positive.
  3. The literal may contain only one decimal point. The decimal point may appear anywhere in the literal, except as the rightmost character. A decimal point designates an assumed decimal point location. (The assumed decimal point in any numeric literal or data item is the position where the compiler and the generated program assume the decimal point to be, though no internal memory position is reserved for a separate decimal point character.) A literal with no decimal point is an integer.

  The term **integer** is used to describe a numeric literal which is unsigned and greater than zero and which has no character positions to the right of the assumed decimal point.

Example 2-3

(Here, the assumed decimal point is represented by the character V.)

Literal	Location of assumed point	Internal sign	No. of digit positions assigned
+123 3.765 −45.7	123V 3V765 45V7	+ + −	3 4 3

- **Numeric floating-point literals**

  A numeric floating-point literal must have the following format:

  mantissa exponent

  The mantissa consists of an optional sign followed by 1 to 16 digits with a decimal point. The decimal point may be specified anywhere in the mantissa.

  The exponent consists of the symbol E, followed by an optional sign and then by one or more digits (the exponent 0 can be written as 0 or 00).

  The literal must not contain blanks. The exponent must be specified immediately to the right of the mantissa.

  The sign is the only optional character in the format. An unsigned mantissa or an unsigned exponent is interpreted as positive.

  The value of the literal is the product of the mantissa and the power of 10 given by the exponent.

  The absolute value of a number represented by a floating-point literal must not exceed 7.2*10 .

  Example 2-4

  $+1.5E-2=1.5*10^{-2}$

2.3.5 PICTURE character-string

A PICTURE character-string consists of certain combinations of characters from the COBOL character set, which are used as symbols (see PICTURE clause, page 164).

Any punctuation character within a PICTURE character-string is not interpreted as a punctuation character but rather as a symbol used in that PICTURE character-string.

2.3.6 Comment-entry

A comment-entry in the Identification Division is an entry consisting of any combination of characters from the character set of the data processing system.

2.3.7 Concept of computer-independent data description

To make data as computer-independent as possible, the characteristics and properties of the data are described in terms of a standard data format rather than a machine-oriented format. This standard data format is derived from general data processing applications and uses the decimal system to represent numbers (regardless of how the computer system represents numbers internally), and all remaining characters of the COBOL character set to specify nonnumeric data items.

1. Concept of logical record and file

The logical characteristics of a record or a file differ from the way in which the data is physically stored in the computer.

- **Physical aspects of a file**

  The physical aspects of a file are determined by the way in which the data is stored on the input or output medium. They include such features as:

  – the grouping of logical records, taking into account the physical limitations of the storage medium.

  – the manner in which a file may be identified.

- **Conceptual characteristics of a file**

  The conceptual characteristics of a file are determined by the structures which the user specifies by data definitions. The input-output statements in a COBOL program refer to logical records.

  It is extremely important to distinguish between a physical record and a logical record.

– A **physical record** (or **block**) is a unit of information whose size and recording mode provide for optimum data storage on an input or output medium for a particular computer installation. The size of a physical record is machine-dependent and bears no direct relationship to the size of the logical file information.

– A **logical record** (or simply **record**) is a group of related data which can be uniquely identified and treated as a unit, and can be read from or written to a file. A block may contain several records.

  The term "record" is not restricted to data stored on an external data medium, but can be applied to the definition of working storage for data created internally during program execution.

In this manual, references to "records" always mean to logical records.

2. Level concept

The level concept permits the structuring of a logical record. Data processed by a COBOL program can be described as elementary items, group items, records and files (for file description see chapters 4 through 6).

- **Elementary items**

  An elementary item is the smallest unit of data bearing a name, i.e. it is not divisible into further elementary items. An elementary item is described with a PICTURE clause (except in the case of description with COMP-1, COMP-2, or index data items).

  The length of an elementary item must not exceed 65535 bytes.

- **Group items**

  Several elementary items combined form a group item. Thus, a number of elementary items may be addressed simultaneously under the name of the group item. Each group consists of an elementary item or a series of elementary items. Groups, in turn, may be combined to form two or more group items. Consequently, an elementary item may belong to more than one group item (see Figure 2-1, Figure 2-2). The name of a group item must not be described with the PICTURE clause.

- **Records**

  A record is a data item which is not subordinate to another data item. It consists of one or more group items with one or more elementary items, or it is itself an elementary item. The description of a record must start in Area A.

- **Level-numbers**

  Data is divided into various levels. These levels are indicated by means of level-numbers. The numbers 01 to 49 are allowed as level-numbers. In addition, there are special level-numbers: 66, 77, and 88. In a source program, every level-number must be given a separate entry.

Since a record represents the largest organizational unit, level-numbers for records start at 01. Hierarchically subordinate items are assigned numerically higher level-numbers (from 02 to 49). The level-number of a subordinate data item must be greater than that of a higher-ranking data item by one or more units. Once an elementary item has been described, only those level-numbers which have already appeared in the record description entry are permitted.

Example 2-5

right:

```
01  DATA RECORD.
    05  GROUP-ITEM-1.
        10  ELEMENTARY-ITEM-11 ...
        10  ELEMENTARY-ITEM-12 ...
        10  ELEMENTARY-ITEM-13 ...
    05  GROUP-ITEM-2.
        10  ELEMENTARY-ITEM-21 ...
        10  ELEMENTARY-ITEM-22 ...
```

wrong:

```
01  DATA RECORD.
    05  GROUP-ITEM-1.
        10  ELEMENTARY-ITEM-11 ...
        10  ELEMENTARY-ITEM-12 ...
        10  ELEMENTARY-ITEM-13 ...
    03  GROUP-ITEM-2.
        10  ELEMENTARY-ITEM-21 ...
        10  ELEMENTARY-ITEM-22 ...
```

There are three types of data for which no level concept exists. These are assigned the level numbers 66, 77, and 88:

– Level number 66 is given to the names of data items described with the RENAMES clause (see "RENAMES clause", page 183).

– Level number 77 is given to structure-independent data items of the WORKING-STORAGE SECTION or LINKAGE SECTION (see "Level number", page 143).

– Level number 88 is given to the explanation of condition-names (see "VALUE clause", page 202).

Level numbers 01 and 77 must be located in Area A of the source program; all other level numbers may begin in Area A or Area B.

For further rules see under "Level number" (page 146).

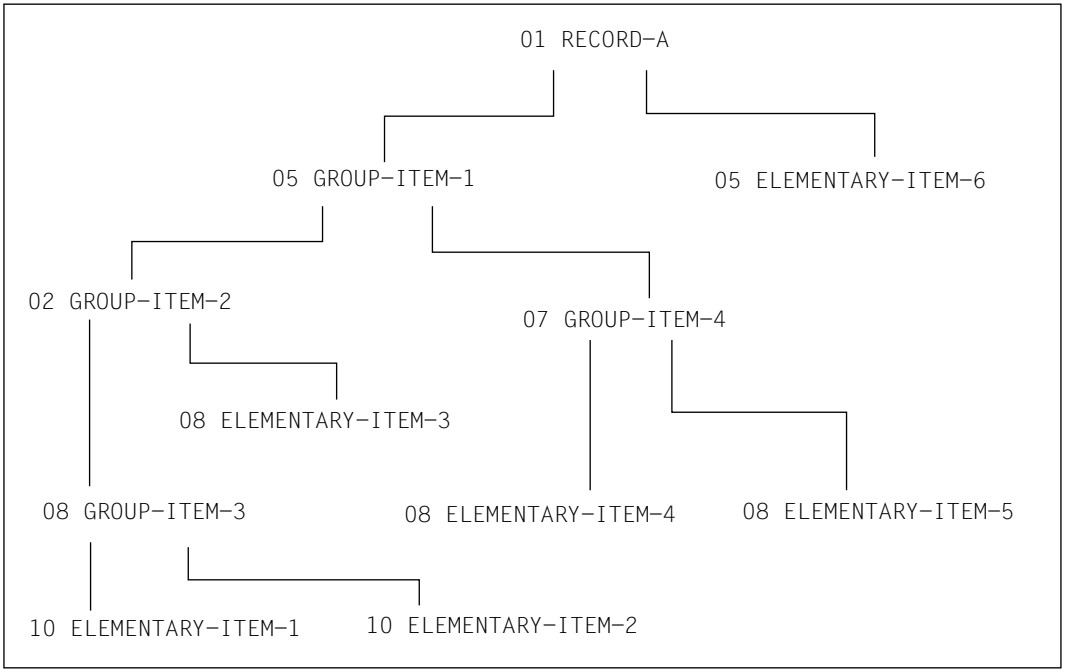


Fig. 2-1: Relationship between group items and elementary items in a record

```
01 RECORD-A.
   05 GROUP-ITEM-1.
      07 GROUP-ITEM-2.
         08 GROUP-ITEM-3.
            10 ELEMENTARY-ITEM-1...
            10 ELEMENTARY-ITEM-2...
         08 ELEMENTARY-ITEM-3...
      07 GROUP-ITEM-4.
         08 ELEMENTARY-ITEM-4...
         08 ELEMENTARY-ITEM-5...
   05 ELEMENTARY-ITEM-6...
```

Fig. 2-2: Group items and elementary items in a record

Figure 2-1 shows the structure of a sample record; Figure 2-2 demonstrates how to use level numbers to represent this structure in the record description entry. In this example, GROUP-ITEM-3 and ELEMENTARY-ITEM-3 are a subordinate part of GROUP-ITEM-2; similarly, GROUP-ITEM-2 and GROUP-ITEM-4 are a subordinate part of GROUP-ITEM-1.

3. Data classes

The five categories of data items (see "PICTURE clause") are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric items, the classes and categories are synonymous. The alphanumeric class includes the categories "alphanumeric edited", "numeric edited" and "alphanumeric" (without editing).

Each elementary item fits into one of the classes and also into one of the categories.

Group items are always classified as alphanumeric at object time, regardless of the class of the elementary items subordinate to them.

Table 2-7 below illustrates the relationship between the classes and categories of data items.

Level of item	Class	Catagory
elementary	alphabetic	alphabetic
	numeric	numeric
	alphanumeric	numeric edited alphanumeric edited alphanumeric
group	alphanumeric	alphabetic numeric numeric edited alphabetic edited alphanumeric

Table 2-7: Classes and categories of elementary and group items

4. Data categories

The following subsections describe the data items in the various categories, as shown above under "Data classes". The category of a data item is determined by the type of the PICTURE and USAGE clauses which are present in the description element (for further details see "PICTURE clause", page 164 and "USAGE clause", page 192).

If the data item is a function, it is class and category alphanumeric or numeric (see chapter 12, "Intrinsic functions").

- **Alphabetic data items**

  An alphabetic data item is a data item whose contents, if represented in standard format, can be any combination of the 52 uppercase and lowercase letters of the alphabet, plus the space character. Each alphabetic character is stored in its own byte in working storage.

  The PICTURE character-string for alphabetic items contains only the symbol A. The data format of alphabetic items is always DISPLAY.

- **Numeric items**

  There are two types of numeric data items, fixed-point items and floating-point items. For the internal representation of numeric items, see Table 3-5, page 200.

  Fixed-point data items

  A fixed-point data item is a numeric data item in which the operational decimal point is assumed to be present in every value or to be maintained at a fixed position relative to the beginning or end of the storage area reserved for the data item. The contents of a fixed-point data item must be comprised of the digits 0 through 9, provided the SIGN clause is not specified. If the SIGN clause is specified, the contents may contain a +, − or other representations of the sign in addition to the above-mentioned digits. If the picture-string contains an S for a fixed-point data item, the contents of the data item are treated as positive or negative, depending on the operational sign. If the picture-string does not contain an S, the contents of the data item are treated as an absolute value. treated as an absolute value.

  Picture-strings for fixed-point items may contain the symbolic characters 9, P, S and V only.

  COBOL recognizes three types of fixed-point numbers:

external decimal	(USAGE IS DISPLAY)
binary	(USAGE IS COMPUTATIONAL or COMPUTATIONAL-5 or USAGE IS BINARY)
internal decimal	(USAGE IS COMPUTATIONAL-3 or USAGE IS PACKED DECIMAL)

  The differences between these three types are described under "USAGE clause" (page 192).

  Floating-point data items

  A floating-point data item is a numeric data item whose decimal point is movable, i.e. it can be located at various positions and is defined by specifying a base-10 exponent. The base thus raised to a particular power serves as the coefficient of a fixed-point number (mantissa) thereby representing the floating-point number.

  Floating-point data items are only used for data whose potential value range is too large for fixed-point representation.

  There are two kinds of floating-point data items: external floating-point data items and internal floating-point data items.

External floating-point data items

The picture-string for an external floating-point data item can contain the characters 9, . (decimal point), V, E, + and −. Except for V, each character occupies one byte of memory, and is contained in each printout (see "PICTURE clause", page 164 and "USAGE clause", page 192 for further details).

The data format of an external floating-point data item is always DISPLAY.

Internal floating-point data items

There are two kinds of internal floating-point data items:

– single-precision (USAGE IS COMPUTATIONAL-1), with a length of 4 bytes and

– double-precision (USAGE IS COMPUTATIONAL-2), with a length of 8 bytes.

Both encompass the same value range. A single-precision data item permits precision to 7 decimal digits. A double-precision data item allows precision to 16 decimal digits.

A PICTURE clause is prohibited for internal floating-point data items; the length of this kind of data item is determined by its USAGE clause.

- **Alphanumeric items**

  An alphanumeric item is one whose contents, when represented in standard data format, are any characters from the EBCDIC set.
  Its picture-string is restricted to combinations of the symbols A, X, and 9. The item is treated as if its picture-string contained all X's.
  A picture-string which contains all A's or all 9's does not define an alphanumeric item.

  The data format of an alphanumeric item is always DISPLAY.

- **Numeric edited items**

  A numeric edited item describes the editing of a numeric value. When a numeric edited item is a receiving item for a MOVE, the data coming into the item is edited according to the picture-string specified.
  The picture-string of a numeric edited item is restricted to certain combinations of the symbols B, / (slash), P, V, Z, 0 (zero), 9, , (comma), . (decimal point), *, +, −, CR, DB and $ (currency sign). The allowable combinations are determined from editing rules and the order or precedence of symbols (see the "PICTURE clause", page 164). The maximum number of digits that may be represented in a picture-string for a numeric edited item is 18.
  Data is stored one character per byte. The contents of a character position that represents a digit must be one of the numerals 0 through 9.

  The data format of a numeric edited data item is always DISPLAY.

- **Alphanumeric edited items**

  An alphanumeric edited item describes the editing of an alphanumeric value. When an alphanumeric edited item is a receiving item for a MOVE statement, the data being moved into the item is edited according to the PICTURE character-string specified for the item.

  The picture-string of an alphanumeric edited item is restricted to certain combinations of the following characters: A, / (slash), X, 9, 0 (zero), and B (see "PICTURE clause", page 164).

  The contents of an alphanumeric edited item, when represented in standard data format, are allowable characters chosen from the EBCDIC character set.

  The data format of an alphanumeric edited item is always DISPLAY.

5. Algebraic signs

There are two categories of algebraic signs:

– operational signs, which are associated with signed numeric items and signed numeric literals to specify their algebraic properties

– editing signs, which occur e.g. in edited reports in order to indicate the sign of a data item.
  Editing signs are inserted in a data item by means of the sign control character of the relevant picture-string (see "PICTURE clause", page 164).

6. Alignment of data

The alignment of data within elementary data items depends on the category of the receiving item. The alignment within group items is the same as for alphanumeric receiving items.

- **Numeric data items**

  If the receiving item is described as a numeric item, the data being sent is aligned on the decimal point and is moved to the character positions of the receiving item. If the data being sent is shorter than the receiving item, the unused character positions are filled with zeros. If the data being sent is longer than the receiving item, it is truncated from the left or right as appropriate.
  If an assumed decimal point is not supplied explicitly, the receiving item is treated as if it had an assumed decimal point immediately following its rightmost character; alignment and moving are as described above.

- **Numeric edited data items**

  If the receiving item is a numeric edited item, alignment and moving of the data being sent take place as in the case of numeric receiving items; leading zeros can be replaced by other characters through special editing specifications.

- **Alphanumeric, alphanumeric edited, and alphabetic data items**

  If the receiving item is alphanumeric (other than numeric edited), alphanumeric edited, or alphabetic, then the data being sent is moved from left to right into the character positions of the receiving item. If the data being sent is shorter than the receiving item, the unused character positions are filled with spaces. If the data being sent is longer than the receiving item, the excess characters of the data being sent are truncated.
  If the JUSTIFIED clause is specified for the receiving item, refer to the description of the "JUSTIFIED clause" (page 153).

- **Data item alignment for accelerated program execution**

  Particular data (in arithmetic or subscripting operations) can be processed more rapidly if the data is aligned on natural boundaries (halfword, word, doubleword).
  The object program requires additional machine instructions for accessing and storing data if parts of two or more data items occur between two adjacent natural boundaries or if certain natural boundaries divide a single item.
  Data items whose alignment on these natural boundaries is such that they do not require additional machine instructions, are defined as "synchronized".

  The user has two means of achieving this form of alignment:

  – using the SYNCHRONIZED clause (see "SYNCHRONIZED clause", page 189),

  – suitably organizing the data, allowing for the natural boundaries.
    See the next section for details.

2.3.8 Implementor-dependent representation and alignment of data

1. Data formats

- **Standard data format**

  The standard format used to store data items with USAGE DISPLAY in internal memory is as follows:

  Each **character position** (as specified by the picture-string) is represented by one byte.

  Each **character** is internally represented by the appropriate code from the EBCDIC character set.

  The EBCDIC character set is described in section 2.9 (page 108).

- **Other data formats**

  The internal representations of internal decimal, binary and floating-point (long or short) data formats are described under "USAGE clause".

2. Alignment by insertion of slack bytes

There are two types of slack bytes:

– Intra-record slack bytes (slack bytes within records) are unused character positions which precede every aligned data item in the record.

– Inter-record slack bytes (slack bytes between records) are unused character positions which are inserted between blocked logical records.

- **Intra-record slack bytes**

  For an output file or in the WORKING-STORAGE section, the compiler inserts slack bytes within records to ensure that all aligned data items are justified on the appropriate boundaries. For an input file or in the LINKAGE section, the compiler expects any required slack bytes to be present in order to ensure proper alignment of a data item declared as SYNCHRONIZED.

  Since it is very important for the user to know the length of a record in a file, the algorithm that the compiler uses to determine whether slack bytes are required and, if they are required, how many slack bytes are to be added, is described as follows:

  The number of occupied bytes in all elementary data items which precede a data item in a record is computed, including any slack bytes previously added.

This sum is to be divided by m, where:

m = 2 for COMPUTATIONAL or COMPUTATIONAL-5 or BINARY data items with a length of 4 digits or less;

m = 4 for COMPUTATIONAL or COMPUTATIONAL-5 or BINARY data items with a length of 9 digits or less;

m = 4 for COMPUTATIONAL-1 data items,

m = 8 for COMPUTATIONAL-2 data items with a length of 18 digits or less,

m = 4 for index data items.

If the remainder r of this division is equal to zero, no slack bytes are required. If the remainder is unequal to zero, the number of slack bytes to be added is equal to m − r.

These slack bytes are added to each record immediately following the elementary item that precedes the BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-5 or INDEX data item. They are declared as though they were a data item with a level number equal to that of the data item immediately preceding the aligned data item, and must be included in the size of the group where they are contained.

Example 2-6

Slack bytes within records

```
01  A.
    02   B  PICTURE X(5).
    02   C.
         03 D  PICTURE XX.
        [03 slack byte PICTURE X. Inserted by the compiler.]
         03 E  PICTURE S9(6) COMP SYNCHRONIZED.
```

Slack bytes are also added by the compiler when a group item is described with an OCCURS clause and contains an aligned data item defined with USAGE as BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-5 or INDEX. To decide whether to add slack bytes, the following steps are performed:

– The compiler calculates the size of the group including all intra-record slack bytes required.

– This sum is divided by the largest m required by any elementary item within the group.

– If the remainder r of this division is equal to zero, no slack bytes will be needed. If r is unequal to zero, m-r slack bytes must be added.

Insertion of slack bytes takes place at the end of each occurrence of the group item which contains the OCCURS clause, in order to ensure that all occurrences of table items begin at the same kind of boundary. In example 2-7, all occurrences of D begin one byte beyond a double-word boundary.

Example 2-7

Occurrences of slack bytes in tables

```
01 A.
   02 B  PICTURE X.
   02 C  OCCURS 10 TIMES.
      03 D PICTURE X.
     [03 slack bytes PICTURE XX. Inserted by the compiler.]
      03 E PICTURE S9(4)V99 COMP SYNC.
      03 F PICTURE S9(4) COMP SYNC.
      03 G PICTURE X(5).
     [03 slack bytes PICTURE XX. Inserted by the compiler.]
```

If aligned data items defined as BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-5 or INDEX follow an entry with an OCCURS DEPENDING clause, then slack bytes are added on the basis of the item which is repeated with the maximum number. If the length of this item is not divisible by the m required by the data, then only certain values of the data-name used in the DEPENDING phrase produce a correct alignment of the items. The programmer should be aware of this situation and try to avoid it. These values are ones in which the length of the data item, multiplied by the number of occurrences plus the number of slack bytes calculated on the basis of the maximum number of occurrences, is divisible by m with no remainder.

Example 2-8

Occurrences of slack bytes in tables with the DEPENDING phrase

```
01  A.
    02  B   PICTURE 99.
    02  C   PICTURE X OCCURS 50 TO 99 TIMES
            DEPENDING ON B.
   [02  slack bytes PICTURE X. Inserted by the compiler.]
    02  D   PICTURE S99 COMP SYNC.
```

In this example, when references to D are required, B is restricted to odd values.

```
01  A.
    02  B   PICTURE 999.
    02  C   PICTURE XX OCCURS 20 TO 99 TIMES
            DEPENDING ON B.
   [02  slack bytes PICTURE X. Inserted by the compiler.]
    05  D   PICTURE S99 COMP SYNC.
```

In this example, all values of B provide correct references to D.

- **Inter-record slack bytes**

  When records that contain aligned data items are to be blocked, the programmer must ensure that all records following the first record in the input-output storage area are properly boundary-aligned. This is only necessary, however, in cases where data is to be processed blockwise (locate mode). COBOL85 does not use this mode.

2.4 Uniqueness of references

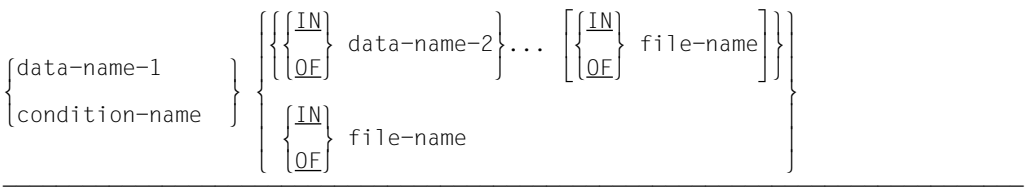
2.4.1 Qualification

Function

Every user-defined name explicitly referenced in a COBOL source program must be unique. A name is unique when there is no other name consisting of the same sequence of characters and hyphens, or the name occurs in a hierarchy of names, so that it can be referenced unambiguously. This occurs by specifying one or more names on a higher level of the hierarchy. The higher levels are called qualifiers, and the process that causes the name to be unique is called qualification. A name must be qualified sufficiently to be unique; however, it is not absolutely necessary to specify all levels of the hierarchy. Within the Data Division, all data names used for qualification purposes must be given a level number or a level identifier. Thus, two identical data names cannot be subordinate elements of a single group item, unless they can be uniquely qualified. In the Procedure Division, two identical paragraph names are only allowed to occur in the same section if they are not referenced. If a paragraph name is referenced, it must be unique, i.e. it must be qualified when it occurs in more than one section.

In the qualification hierarchy, the names belonging to a level identifier are the most important, followed by the names belonging to level 01, then those belonging to level 02 to 49. A section name is the only qualifier available for paragraph names. The uppermost name in the hierarchy must be unique, and cannot be qualified. Subscripted or indexed data names and conditional variables, as well as procedure names and data names, can be made unique by means of qualification. The name of a conditional variable can be used as a qualifier for each of its condition names.

Format 1



Format 2

```
paragraph-name   {IN} section-name
                 {OF}
```

Format 3

```
text-name {IN} library-name
          {OF}
```

Format 4

```
LINAGE-COUNTER  {IN} file-name
                {OF}
```

Format 5

```
{PAGE-COUNTER}  {IN} report-name
{LINE-COUNTER}  {OF}
```

Format 6

```
                {{IN} data-name-2 [{IN} report-name]}
data-name-1     {{OF}             [{OF}            ]}
                {{IN} report-name                   }
                {{OF}                               }
```

Syntax rules

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.
2. The same name must not appear on more than one level of the hierarchy.
3. A data name must not be subscripted or indexed when used as a qualifier.

General rules

1. A data-name or a condition-name, if assigned to more than one data item within the source program, must be qualified whenever it is referenced in the Procedure, Environment or Data Division (except in the REDEFINES clause, where qualification is not needed and may be used).
2. A paragraph-name is only allowed to occur more than once within a section if it is not referenced. If it is referenced, it is only allowed to occur once within a section, or must be qualified when it occurs in more than one section. When a paragraph-name is qualified by a section-name, the word SECTION must not be used. A paragraph-name, when referenced from within the same section, need not be qualified.
3. A name may be qualified even when qualification is not required; if uniqueness may be ensured by more than one combination of qualifiers, then each such combination is permitted. The total set of the qualifiers for a given data-name must not be identical to a subset of qualifiers for another data-name.
4. If more than one COBOL library is available to the compiler at compile time, then every time text-name is referenced it must be qualified by library-name.
5. If data-name is qualified in a contained or containing program of a nested program, the same data-name must not be used for a unit of data (record or data item) that is declared as external or global in one of the group of nested programs.

2.4.2 Subscripting

Function

Subscripts are used when an individual element is to be accessed within a table (see OCCURS clause, page 155).

Format 1

describes subscripting without qualification.

```
{data-name     }
{              } ({subscript-1... )
{condition-name}
```

Format 2

```
{data-name     } {IN} data-name-1 [{IN} data-name-2] ... ({subscript-1}... )
{              } {  }              [{  }             ]
{condition-name} {OF}              [{OF}             ]
```

Describes subscripting with qualification.

For the explanation of and rules for qualification see "Qualification" (page 77).

Syntax rules for both formats

1. data-name is the name of the table element. Its data description entry must either contain an OCCURS clause, or it must be subordinate to a data item which contains an OCCURS clause.
2. subscript-1... may be represented by
   - an integer literal
   - a data-name with a positive integer as its value
   - relative subscripting
   - an arithmetic expression with the value of a positive integer
   - the word ALL.

The data-name itself may be qualified but not indexed. ALL may be specified only if the subscripted identifier is specified as a function argument.

3. One subscript must be specified for each OCCURS clause which is subordinate to data-name. Since a table may have up to seven dimensions, references to an element in a table may require up to seven subscripts.
4. The subscript is enclosed in parentheses. The left parenthesis immediately follows the spaces after the name of the table element (data-name). When more than one subscript appears within a set of parentheses, these subscripts may be separated either by commas followed by at least one space, or by spaces only.
   In the case of relative subscripting, the operational signs between data-name and integer must also be delimited by spaces.
5. The subscript, or set of subscripts, identifies the table element which is to be referenced. A data-name to which one or more subscripts have been added is called a subscripted data-name or identifier.
6. When more than one subscript is used, they are entered proceeding from the outermost to the innermost table.

General rule for both formats

The **subscript** may contain a plus sign. The lowest valid subscript is 1. Consequently, neither zero nor negative numbers are permitted for subscripting. The highest allowable subscript value, in any particular case, is the maximum number of occurrences of the item, as specified in the OCCURS clause.

2.4.3 Indexing

Function

Indices are used when an individual element is to be accessed within a table (see OCCURS clause, page 155).

Format 1

Describes indexing without qualification:

```
{data-name-1   }
{              } ( {index-1 [{+} integer]}... )
{condition-name}             [{-}        ]
```

Format 2

Describes indexing with qualification:

```
{data-name-1   } {IN} data-name-2 [{IN} file-name-1]...
{              } {  }             [{  }
{condition-name} {OF}             [{OF}

        ({index-1 [{+} integer]}... )
                   [{-}        ]
```

For a description of qualification with associated rules see "Qualification" (page 77).

Syntax rules for both formats

1. data-name-1 is the name of a table element.

   If a data-name-1 is used with an index-name, then the data description entry of data-name-1 must either itself contain an OCCURS clause with an INDEXED BY phrase, or data-name-1 must be subordinate to a group item containing an OCCURS clause with an INDEXED BY phrase.

   For example, the reference

   ```
   TOTAL (INDEXA, INDEXB),
   ```

   implies that TOTAL belongs to a structure with two levels of OCCURS clauses, each with an INDEXED BY phrase specified.

2. The index is enclosed in parentheses. The left parenthesis immediately follows the space after the name of the table element (data-name). When more than one index-name appears within a set of parentheses, these index-names may be separated either by commas followed by at least one space, or by spaces only.
3. When the + integer or − integer phrase is used, the + and − characters must be preceded and followed by spaces.
4. Index-names are written proceeding from the outermost to the innermost table.
5. The lowest valid occurrence number for index-name is 1; the highest is, in any particular case, the maximum number of occurrences of the item. This maximum number is defined in the OCCURS clause. This same rule also applies to relative indexing.
6. Referencing a table element, or an item within a table element, does not change the index-name associated with this table.
7. The use of relative indexing will not change the values of indices in the object program.

General rules for both formats

1. The values of indices may be stored without conversion (SET statement) in data items defined with the USAGE IS INDEX clause. These data items are then called index data items (see "USAGE clause", page 192 and "SET statement", page 334).
2. An index may be modified only by a SET, SEARCH or PERFORM statement (see the descriptions of these statements).

2.4.4 Function-identifier

A function-identifier is a syntactically correct combination of character-strings and separators that uniquely references the data item resulting from the evaulation of a function.

Format

```
FUNCTION function-name-1 [({argument-1}...)] [reference-modifier]
```

Syntax rules

1. argument-1 must be an identifier, a literal, or an arithmetic expression. Specific rules governing the number, class, and category of argument-1 are given in the definition of each function (see chapter 12, "Intrinsic functions").
2. A reference-modifier may be specified only for functions of the category alphanumeric.
3. A function-identifier which references an alphanumeric function may be specified anywhere in the general formats that an identifier is permitted and where the rules associated with the general formats do not specifically prohibit reference to functions, except as follows:
   a) as a receiving operand of any statement,
   b) where the rules associated with the general formats require the data item being refernced to have particular characteristics (such as class and category, usage, size, sign, and permissible values) and the evaluation of the function according to its definition and the particular arguments specified would not have these characteristics.
4. A function-identifier which references an integer or numeric function may be used only in an arithmetic expression.

General rules

1. The class and other characteristics of the function being referenced are determined by the function definition.
2. At the time reference is made to a function, its arguments are evaluated individually in the order specified in the list of arguments, from left to right. An argument being evaluated may itself be a function-identifier or may be an expression containing function-identifiers. There is no restriction preventing the function referenced in evaluating an argument from being the same function as that for which the argument is specified.

2.4.5 Reference modification

Function

Reference modification defines a data item through specification of the position of the leftmost character and the length of the data item.

Format

```
{data-name-1                                          } (leftmost-char-position: [length])
{FUNCTION function-name-1 [({argument-1}... )]}
```

data-name-1 and FUNCTION function-name-1 are not part of the reference-modifier. They are included here for the sake of clarity.

Syntax rules

1. data-name-1 must reference a data item that is described with USAGE IS DISPLAY.
2. leftmost-character-position and length must be arithmetic expressions.
3. Unless otherwise specified, reference modification may be used wherever an alphanumeric identifier is permitted.
4. data-name-1 may be qualified or subscripted.
5. The function referenced by function-name-1 and its arguments (if any) must be an alphanumeric function.

General rules

1. Each character of data-name-1 or function-name-1 is assigned an ordinal number that is incremented stepwise by one from the leftmost position to the rightmost position. The number one is assigned to the leftmost position. If the data description entry for data-name-1 contains a SIGN IS SEPARATE clause, an ordinal number is likewise assigned to the sign position in this data item.
2. If data-name-1 is described as numeric, numeric edited, alphanumeric or alphanumeric edited, it is operated upon for the purposes of reference modification as if it were redefined as an alphanumeric data item of the same size.
3. If data-name-1 is subscripted and ALL is specified for a subscript, the reference-modifier refers to each of the implicitly referenced table elements.

4. Reference modification creates a unique data item that forms a subset of the data item referenced by data-name-1 or function-name-1. This unique data item is defined as follows:

   a) leftmost-character-position specifies the character position of data-name-1 at which the subfield is to begin.
   leftmost-character-position must give a positive nonzero integer value that is less than or equal to the number of character positions of data-name-1 or function-name-1 and its arguments (if any).

   b) length denotes the length of the unique data item. length must give a positive non-zero integer value.

   c) The sum of leftmost-character-position and length minus 1 must not exceed the number of characters of the data item referenced by data-name-1 or function-name-1. If "length" is not specified, the unique data item extends from the position denoted by leftmost-character-position to the last character (inclusive) of the data item referenced by data-name-1 or function-name-1.

5. The unique data item is considered to be an elementary item without a JUSTIFIED clause. If function-name-1 is specified, the data item has "alphanumeric" class and category. If data-name-1 is specified, it has the same category and class as the data item referenced by data-name-1, except that the categories "numeric", "numeric edited" and "alphanumeric edited" are considered to be alphanumeric category and class.

Example 2-9

A data item CARREG contains a 10-character car registration, the last 6 characters of which are to be transferred to a subitem SHORTREG:

```
Program extract:
...
01  CARREG     PIC X(10).
01  SHORTREG   PIC X(6).
...

    MOVE CARREG (5:6) TO SHORTREG.
...
```

The "5" within the parentheses specifies that the MOVE operation is to take effect starting at the fifth character; the colon is the required separator; the "6" specifies that six characters are to be transferred to the item SHORTREG.

2.4.6 Identifier

Identifier is a term used to reflect that a data name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts or indices necessary to ensure uniqueness.

Format 1

```
FUNCTION function-name-1 [({argument-1}...)] [reference-modifier]
```

Format 2

```
                 [{IN}                 ]      [{IN} {file-name-1   }]
data-name-1      [{   } data-name-2    ]  ...  [{   } {             }]
                 [{OF}                 ]      [{OF} {report-name-1}]

   [({subscript} ...)] [reference-modifier]
```

2.4.7 Condition-name

If referenced explicitly, a condition-name must be unique or be made unique by means of qualification and/or subscripting. This is not necessary if the uniqueness of the reference is guaranteed by the naming conventions for the scope themselves.

If qualification is used in order to render a condition-name unique, the associated conditional variable can be used as the first qualifier. Also, in the case of qualification, the hierarchy of names that are assigned to the conditional variable must be used in order to render a condition-name unique.

If the reference to a conditional variable necessitates subscripting, the same combination of subscripts is required when referencing one of its condition-names.

With regard to the qualification and subscripting of condition-names, the same format and restrictions are applicable as for the identifiers except that data-name-1 is replaced by condition-name-1.

In the general format in the following sections, "condition-name-n" always refers to a condition-name that, depending on the requirements, is qualified or subscripted.

2.5 Table handling

A table is a series of data items of equal length. These items are the table elements, or table items. They all have an identical structure and are stored contiguously. The entire table itself also forms a data item in COBOL terms.

Problems arising during the processing of large amounts of identically structured data can often be solved more satisfactorily by putting this data into tabular form. This allows an effective interpretation and meaningful representation of the information involved.

The homogeneous structure of the individual table elements makes their relationship to one another readily apparent.

The individual table element occupies an easily determined physical location relative to the base of the table, i.e. to the start of the table in working storage. Thus every element can be referenced relative to the beginning of the table and does not need to be assigned a unique data-name. A table element is accessed with the aid of a table element number, or occurrence number (see "Subscripting", page 92 and "Indexing", page 94).

In addition, it is possible to determine the associated occurrence number for any given value of a table element (see "SEARCH statement", page 325).

The number of table elements in a table may be variable at object time (see example 3-19, page 162).

2.5.1 Table definition

A table element is indicated in the data description entry by specifying the OCCURS clause. This clause defines how many elements the table contains. The name and description of the table item apply to each recurrence thereof. In the case of multi-dimensional tables, each dimension in the hierarchical structure must be given an OCCURS clause.

Example 2-10

```
01  TABLE1.
    02 TABLE-ELEMENT PIC XXX OCCURS 20 TIMES.
```

The data item TABLE1 comprises 20 data items of identical length. These items are given the name TABLE-ELEMENT:

```
TABLE1: 1. TABLE-ELEMENT (1)   PIC XXX.
        2. TABLE-ELEMENT (2)   PIC XXX.
        .
        .
        .
       20. TABLE-ELEMENT (20)  PIC XXX.
```

One-dimensional tables

The OCCURS clause is entered in the data description entry of the table element.

Example 2-11

```
01  TABLE2.
    02 TABLE-ELEMENT OCCURS 2 TIMES.
       03 ELEMENT-ITEM-1      PIC X(4).
       03 ELEMENT-ITEM-2      PIC X(4).
```

TABLE2 is the name of the table.

TABLE-ELEMENT is the element which occurs twice within the one-dimensional TABLE2.

ELEMENT-ITEM-1 and ELEMENT-ITEM-2 are elements which are subordinate to TABLE-ELEMENT.

Multi-dimensional tables

When a data item is subordinate to a table-element within a two-dimensional table and contains an OCCURS clause, then this data item is an element within a three-dimensional table.

Up to seven dimensions are allowed for a single table.

Example 2-12

```
01  TABLE3.
    02 BLK OCCURS 2 TIMES.
       03 RECORD  OCCURS 2 TIMES.
          04 ITEM  OCCURS 2 TIMES PIC X(10).
```

BLK is an element which occurs twice within a one-dimensional table.

RECORD is an element in a two-dimensional table. It occurs twice within each occurrence of BLK.

ITEM is an element in a three-dimensional table. It occurs twice within each occurrence of RECORD.

TABLE	BLK (1)	RECORD (1, 1)	ITEM (1, 1, 1)
			ITEM (1, 1, 2)
		RECORD (1, 2)	ITEM (1, 2, 1)
			ITEM (1, 2, 2)
	BLK (2)	RECORD (2, 1)	ITEM (2, 1, 1)
			ITEM (2, 1, 2)
		RECORD (2, 2)	ITEM (2, 2, 1)
			ITEM (2, 2, 2)

Fig. 2-3 Schematic representation of TABLE

Initial values of table elements

A VALUE clause must not appear in a record description entry with an OCCURS clause, or in any record description entry subordinate to that entry. However, for the definition of condition-names, the VALUE clause is allowed and required here as well.

Initial values may be assigned to a table in the WORKING-STORAGE SECTION by using the VALUE clause.

Example 2-13

```
 WORKING-STORAGE SECTION.

****** 1. VALUE ON GROUP-LEVEL ******

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

****** 2. REPEATED VALUE WITH OCCURS ******

 01  WEEK.
     02  WDAY  PIC X(10) OCCURS 7 TIMES VALUE FROM (1)
               "MONDAY" "TUESDAY" "WEDNESDAY" "THURSDAY"
               "FRIDAY" "SATURDAY" "SUNDAY".

****** 3. REPEATED VALUE SUBORDINATE TO OCCURS ******

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

References to table elements

All the elements within a table have the same data-name. To identify individual occurrences of table elements, occurrence numbers (indexes) enclosed in parentheses are appended to the data-name.

Example 2-14

```
01  TABLE4.
    02  ELEMENT   OCCURS 10 TIMES.
          .
          .
          .
    MOVE ELEMENT OF TABLE4 (8) TO ...
```

Here the eighth table element is accessed.
An occurrence number must be supplied for each dimension.

There are two techniques for referencing table elements:

– subscripting
– indexing

2.5.2 Subscripting

One method of specifying occurrence numbers is to append one or more subscripts to the data-name. A subscript is an integer whose value represents the occurrence number of a table element or one of the items subordinate to that table element. The subscript may be represented

– by an integer literal

– by a data-name defined as a numeric elementary data item without any character positions to the right of the assumed decimal point

– by an arithmetic expression that is neither a direct nor a relative subscript.

In either case, the subscript must be enclosed in parentheses and must be written immediately following any qualification for the name of the table element. The referenced table element must have appended to it as many subscripts as the associated table has dimensions. A subscript must therefore be supplied for each OCCURS clause, including the OCCURS clause which contains the data-name within the defined hierarchy.

In example 2-12 (three-dimensional table), the following are required:

– one subscript for references to BLK

– two subscripts for references to RECORD

– three subscripts for references to ITEM.

Subscripts are written proceeding from the outermost to the innermost table.

Thus, for example,

ITEM (1, 2, 2)

identifies the second element ITEM

within the second element RECORD

within the first element BLK.

A reference to a data item must not be subscripted unless the data item is a table element, or unless it is an item or condition-name within a table element.

There are three forms of subscripting:

– direct subscripting

– relative subscripting

– subscripting by means of an arithmetic expression

Direct subscripting

With direct subscripting, the subscript is specified either by an integer literal or by a data-name. The data-name must be defined as a numeric elementary item with no character positions to the right of the assumed decimal point. In the preceding example, direct subscripting was used.

Relative subscripting

If the name of the table element is followed by a subscript in the form:

```
(data-name + integer-1),
```

then the occurrence number required to complete the reference is calculated from the value of data-name at object time, plus integer-1.

If it takes the form:

```
(data-name - integer-2),
```

the occurrence number is obtained by subtracting integer-2 from data-name.

Relative subscripting is treated in the same manner as relative indexing. For further details see "Indexing" (page 94).

Subscripting by means of an arithmetic expression

A subscript can consist of an arithmetic expression that supplies an integer as its result.

If a subscript consists of an arithmetic expression that is neither a direct nor a relative subscript, then the required occurrence number is calculated from the value of the arithmetic expression at object time.

At both compile time and object time, arithmetic expressions are processed more slowly than direct or relative subscripts. For this reason, swapping of data-name and integer in relative subscripts should be avoided, as too should the enclosure of a direct or relative subscript in parentheses since such expressions are considered to be arithmetic expressions.

If a subscript ends with a data-name or an index, then an immediately following subscript must not begin with a left parenthesis since this would initiate subscripting of the data-name or index.

2.5.3 Indexing

Another technique for referencing table elements is indexing. Indexing is made possible by supplying the INDEXED BY phrase in the OCCURS clause.

The index does not require its own data description entry. At object time, the value of an index is a binary value representing a displacement from the beginning of the table. The value of this binary number is calculated from the number and length of the table element as follows:

binary value of index = (occurrence number − 1) * length of table element

The value of an index may only be set using the SET, SEARCH or PERFORM statement. The initial value is undefined and must be set explicitly.

There are two forms of indexing:

– direct indexing
– relative indexing

Direct indexing

Direct indexing obtains when an index is used in the manner of a direct subscript.

Example 2-15

```
01  TABLE1.
    02  TABLE-A PIC XX OCCURS 10 TIMES INDEXED BY INDEX-A.
    02  TABLE-B PIC X(3) OCCURS 5 TIMES INDEXED BY INDEX-B.
            .
            .
    SET INDEX-A TO 7.
            .
    MOVE "X7" TO TABLE-A (INDEX-A).
```

Two tables are defined here:

– TABLE-A with 10 elements, each 2 bytes long
– TABLE-B with 5 elements, each 3 bytes long

INDEX-A is declared for TABLE-A and INDEX-B is declared for TABLE-B by means of the INDEXED BY phrase. Indices may only be used with the corresponding elementary item, e.g. TABLE-A(INDEX-A) or TABLE-B(INDEX-B).
The SET statement sets the index to a value that points to the seventh element of TABLE-A. The displacement from the start of the table, i.e. the internal binary contents of INDEX-A, is (7−1) * 2 = 12. Thus, the MOVE statement transfers X7 to the seventh table element.

Relative indexing

When the name of a table element is followed by an index in the form

(index + integer-1),

then the required occurrence number is calculated from the value of index-name at object time, plus integer-1.

If the form

(index - integer-2),

is used, then the new occurrence number is obtained by subtracting integer-2 from the corresponding current occurrence number.

The use of relative indexing will not change the values of the index-names in the object program.

Permissible value ranges for indices

As specified in the standard, the value of an index should correspond to a valid occurrence number of the associated table. This compiler also permits corresponding occurrence numbers for 0, ZERO, or negative numbers, and values beyond the maximum permissible occurrence numbers, if the binary value of the index remains within the range (representable in 4 bytes) -2^{31} to $+2^{31}-1$. In these cases, the index must be set (e.g. with SET UP or SET DOWN) to a valid occurrence number before it is used, or corresponding relative indexing must be used to ensure that only valid table elements are addressed.

2.5.4 Indexing and subscripting compared

Availability of occurrence numbers for the user

Subscripting:

The occurrence number is immediately available.

Indexing:

The occurrence number is available only if preceded by a SET, SEARCH or PERFORM statement. It is calculated as follows:

value of index divided by length of table element plus 1.

References to table elements

Subscripting:

At object time, the address of the table element must be calculated anew each time from subscripts (except in the case of literal subscripts), i.e. subscripted data items cannot be referenced as quickly as data items outside the table.

Indexing:

When indexing is used, references to a table element are faster than subscripting using identifiers, since the displacement from the start of the table is already stored in the index.

Changing the index

Subscripting:

Changing a subscript in the form of a data-name (using MOVE, ADD etc.) is faster than changing an index-name using SET, since the SET statement requires that the occurrence number must first be converted to the displacement from the start of the table. This applies when the index is not being set up or down by a fixed integer value.

Indexing:

It is faster to change an index using PERFORM or SEARCH statements than to change a subscript.

Validity

Subscripting:

A subscript can also be used for other table elements.

Indexing:

An index may only be used with its associated table element (except in SET, PERFORM and SEARCH statements).

2.6 Statements and sentences

There are four types of statements:

– conditional statements
– compiler-directing statements
– imperative statements
– delimited scope statements

There are three types of sentences:

– conditional sentences
– compiler-directing sentences
– imperative sentences

2.6.1 Conditional statements and conditional sentences

- A **conditional statement** is used to determine the truth value of a condition and to specify the subsequent action in the object program on the basis of this value.

  Conditional statements include:

  – IF, EVALUATE, SEARCH and RETURN statements;

  – READ statements which contain the (NOT) AT END or (NOT) INVALID KEY phrase;

  – WRITE statements which contain the (NOT) INVALID KEY or (NOT) END-OF-PAGE phrase;

  – START, REWRITE or DELETE statements which contain the (NOT) INVALID KEY phrase;

  – ADD, COMPUTE, DIVIDE, MULTIPLY or SUBTRACT statements which contain the (NOT) ON SIZE ERROR phrase;

  – STRING and UNSTRING statements which contain the (NOT) ON OVERFLOW phrase;

  – CALL statements which contain the ON OVERFLOW or (NOT) ON EXCEPTION phrase.

  – ACCEPT or DISPLAY statements which contain the (NOT) ON EXCEPTION phrase.

- A **conditional sentence** is a conditional statement which may optionally be preceded by an imperative statement and which is terminated by a period followed immediately by a space.

2.6.2 Compiler-directing statements and compiler-directing sentences

- A **compiler-directing statement** consists of one of the compiler-directing verbs COPY, REPLACE or USE and its operands.

  A compiler-directing statement causes the compiler to perform certain actions during compilation.

- A **compiler-directing** sentence is a single compiler-directing statement which is terminated by a period followed immediately by a space.

2.6.3 Imperative statements and imperative sentences

- An imperative statement causes a specific action to be carried out in the object program.
- An imperative statement is a statement which begins with an imperative verb and specifies that an action is to be executed unconditionally or a conditional statement which is delimited by its explicit scope terminator.
- An imperative statement may consist of a sequence of imperative statements, each separated from the next by a COBOL separator. The imperative statements are:

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

(1) without the (NOT) ON SIZE ERROR phrase

(2) without the (NOT) INVALID KEY phrase

(3) without the (NOT) ON OVERFLOW phrase

(4) without the (NOT) AT END or (NOT) INVALID KEY phrase

(5) without the (NOT) INVALID KEY or (NOT) END-OF-PAGE phrase

(6) without the ON OVERFLOW or (NOT) ON EXCEPTION phrase

(7) without the (NOT) ON EXCEPTION phrase

- Whenever "imperative-statement" occurs in the general format of statements, it also refers to a sequence of imperative statements terminated either by a period or by a phrase which in turn includes an imperative statement. For example, the statement

  ```
  DIVIDE A INTO B.
  ```

  is an imperative statement, as is also

  ```
  DIVIDE A INTO B ON SIZE ERROR PERFORM DIV-FEHLER.
  ```

- An imperative sentence is an imperative statement which is terminated by a period followed immediately by a space.

2.6.4 Delimited scope statements

A delimited scope statement is any statement which includes an explicit scope terminator.

2.6.5 Scope of statements (scope terminators)

Scope terminators delimit the scope of certain Procedure Division statements.
Statements which include their explicit scope terminators are termed delimited scope statements.
The scope of statements which are contained within other statements (nested) may also be implicitly terminated.
When statements are nested within other statements, a separator period which terminates the sentence also implicitly terminates all nested statements.
Whenever any statement is contained within another statement, the next phrase of the containing statement following the contained statement terminates the scope of any unterminated contained statement.
When a delimited scope statement is nested within another delimited scope statement with the same verb, each explicit scope terminator terminates the statement begun by the most recently preceding, and as yet unterminated, occurrence of that verb.
When statements are nested within other statements which allow optional conditional phrases, any optional conditional phrase encountered is considered to be the next phrase of the nearest preceding unterminated statement with which that phrase is permitted to be associated according to the general format and the syntax rules for that statement, but with which no such phrase has already been associated.
An unterminated statement is one which has not been previously terminated either explicitly or implicitly.

In addition to the separator period (implicit scope terminator), the following explicit scope terminators can be used to support structured programming:

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

Explicit scope terminators are reserved COBOL words.

2.6.6 Categories of statements

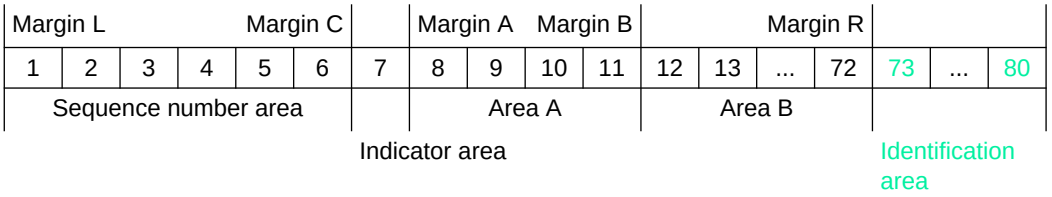
Arithmetic statements	ADD COMPUTE DIVIDE MULTIPLY SUBTRACT
Conditional statements	ACCEPT (EXCEPTION) ADD (SIZE ERROR) CALL (OVERFLOW or EXCEPTION) COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DISPLAY (EXCEPTION) DIVIDE (SIZE ERROR) EVALUATE GO TO (DEPENDING ON) IF MULTIPLY (SIZE ERROR) PERFORM (UNTIL) READ (AT END or INVALID KEY) RETURN (AT END) REWRITE (INVALID KEY) SEARCH START (INVALID KEY) STRING (OVERFLOW) SUBTRACT (SIZE ERROR) UNSTRING (OVERFLOW) WRITE (INVALID KEY or END-OF-PAGE

Data processing statements	ACCEPT (DATE, DAY, DAY-OF-WEEK or TIME) INITIALIZE INSPECT MOVE SET (TO TRUE) STRING UNSTRING
Input/output statements	ACCEPT (identifier) CLOSE DELETE DISPLAY OPEN READ REWRITE START STOP (literal) WRITE
End statement	STOP
Report Writer statements	GENERATE INITIATE TERMINATE
Program communication statements	CALL CANCEL ENTRY EXIT PROGRAM GOBACK
Procedure control statements	ALTER CALL EXIT EXIT PERFORM GO TO PERFORM
Sort statements	MERGE RELEASE RETURN SORT
Table handling statements	SEARCH SET
Compiler-directing statements	COPY REPLACE USE

2.7 Reference format

2.7.1 General description

The standardized reference format for writing COBOL source programs can be described in terms of a line consisting of 80 character positions. The compiler only accepts COBOL source programs written in the reference format and generates a listing of the source program in the same format. A line is divided as follows:



Margin L
is located to the left of the leftmost character position in a line.

Margin C
is located between the sixth and seventh character position in a line.

Margin A
is located between the seventh and eight character position in a line.

Margin B
is located between the eleventh and twelfth character positions in a line.

Margin R
is located to the right of the rightmost character position in a line.

Sequence number area
contains six character positions (columns 1 to 6) located between Margin L and Margin C.

Indicator area
the seventh character position in a line.

Area A
contains the character positions 8 through 11 and is located between Margin A and Margin B.

Area B
contains the character positions 12 through 72. It begins at the first character position to the right of Margin B and ends at the character position to the left of Margin R.

2.7.2 Rules for using the reference format

COBOL programs are written in a standardized format. The rules governing the use of spaces take precedence over all other rules in this manual that govern the insertion or omission of spaces.

- **Sequence number area (columns 1-6)**

  This field may be used to label lines of a COBOL source program.

  The content of the sequence number area is defined by the user and may consist of any character in the computer's character set. The sequence number area can contain a character-string or individual characters.

- **Indicator area (column 7)**

  This field is used to designate continuation, comment, and debugging lines.

  A hyphen (−) in this field signifies that this is a **continuation line**, i.e. the previous line is being continued (see "Continuation of lines" further below). The absence of a hyphen in the indicator field is taken to indicate that the last character in area B (see below) of the previous line is followed by one space character.

  An asterisk (*) supplied in this field indicates a **comment line** (see "Comment line" further below).

  A slash (/) in this field indicates a special kind of comment line, which causes a form feed to be carried out in the source program listing before this line is printed.

  A letter D in this field designates a **debugging line** (see "Debugging", page 353).

- **Area A (columns 8-11)**

  This area is reserved for the beginning of the headers of the four COBOL program divisions, of the section headers and paragraph headings, for level indicators, and for certain level numbers (see Table 2-8, page 105).

- **Area B (columns 12-72)**

  This area is the main field for entries of a COBOL source program. It is used to hold all those clauses and statements which do not have to begin in area A (see Table 2-8, page 105).

- **Identification area (columns 73-80)**

  This area may be used to assign names to lines of a COBOL source program. It may contain any characters from the computer's character set (EBCDIC) or may be blank. Its contents are not evaluated by the compiler. It is advisable for reasons of clarity to provide parts of the program or the entire program with a meaningful name in this area.

- **Continuation of lines**

  A sentence or entry requiring more than one line may be continued on subsequent lines in area B. The first line is called a **continued line**, the following lines are called **continuation lines**. If a sentence or entry spans more than two lines then all lines, except the first and last, are both continued and continuation lines.

  A word, PICTURE character-string, or literal may be continued in the next line. If this occurs, the following apply:

  – Continuation of nonnumeric literals
    If a nonnumeric literal is continued on the next line, a hyphen should be entered in the indicator area (column 7) of the continuation line. The continuation may follow anywhere in area B (from column 12), immediately preceded by a quotation mark. All blanks located at the end of the continued line or following the quotation mark of the continuation line are regarded as part of the literal.

  – Continuation of words and numeric literals
    If a word or a numeric literal is continued on the next line, a hyphen must be entered in the indicator area (column 7) of the continuation line in order to show that the first non-blank character in area B of the continuation line is the immediate successor of the last non-blank character of the continued line, i.e. without any intervening space.

- **Blank lines**

  A blank line is a line that contains only spaces in columns 7 through 72. A blank line may appear anywhere in the source program, except immediately preceding a continuation line.

- **Program divisions, sections, paragraphs and description entries**

Item	Convention for the placement of items
DIVISION header	Must be written starting at margin A (i.e. in column 8) and must appear on a line by itself.
SECTION header	Must be written starting at margin A; no other text except USE and COPY statements as well as segment numbers may appear on the same line.
Paragraph name	Must start at margin A.
Statements/clauses	Statements or clauses of a paragraph must be written within area B. The first sentence of a paragraph may begin on the line which contains the paragraph-name or on a new line.
Level indicators, file, sort-file and report description entry	The level indicators FD, SD and RD must be written starting at margin A; they must be followed, on the same line, starting at margin B, by the related file-name, sort-name or report-name and the associated explanatory information, if any.
Level numbers and file description entries	Level numbers 01 and 77 must be written starting at margin A; all other level numbers may begin anywhere in area A or B. The data description entries which are associated with a particular level number must begin in area B of the same line which contains that level number.
End program header	This must begin in area A.

Table 2-8: COBOL margin conventions

- **Declaratives**

  The keyword DECLARATIVES and the keywords END DECLARATIVES which (respectively) open and close the declaratives section of the Procedure Division must each appear on a separate line. Both must be written starting in area A, and both must be terminated by a period followed by a space.

- **Comment lines**

  Explanatory comments may be included anywhere in the COBOL source program in the form of comment lines by setting an asterisk or a slash in the indicator area (column 7). Any combination of characters from the character set of the data processing system (EBCDIC) may be used in areas A or B of these lines. The contents of the comment lines will be produced on the source program listing (at the top of a new page if a slash has been entered in the indicator area) and have no effect on the program.

- **Debugging lines**

  Debugging lines may appear anywhere in the COBOL source program, following the OBJECT-COMPUTER paragraph. They are indicated by a "D" in the indicator area (see "Debugging", page 353).

- **Pseudo-text**

  The pseudo-text, which consists of character-strings and delimiters, may begin in area A or area B. If indicator area of a line following the opening pseudo-text delimiter contains a hyphen, area A must be left empty; text words are continued in accordance with the normal rules for continuation of lines (see above).

- **End program header**

  The end program header must begin in area A.

2.8 Processing a COBOL program

The COBOL compiler, with the aid of a linkage editor, generates an executable program from a COBOL source program. This executable program is also called an **object program** or **object module**.

The **linkage editor** links the modules generated by the compiler with the necessary runtime subroutines and, if applicable, with further compiler-generated modules.

The **runtime subroutines**, which are supplied as modules, are used for performing special COBOL functions such as input-output operations.

The interactive debugging aid AID is available for symbolic and hardware-oriented testing of COBOL programs.

2.9 EBCDIC character set

Siemens Nixdorf reference version of the 8-bit code

Decimal	Hexadecimal	EBCDIC	Printer graphics
0	00	0000 0000	(LOW-VALUE)
...	...	...	
64	40	0100 0000	(space)
...	...	...	
74	4A	0100 1010	c (cents)
75	4B	0100 1011	. (period)
76	4C	0100 1100	< (less than)
77	4D	0100 1101	((left parenthesis)
78	4E	0100 1110	+ (plus)
79	4F	0100 1111	| (vertical)
80	50	0101 0000	& (ampersand)
...	...	...	
90	5A	0101 1010	! (exclamation mark)
91	5B	0101 1011	$ (dollar sign)
92	5C	0101 1100	* (asterisk)
93	5D	0101 1101	) (right parenthesis)
94	5E	0101 1110	; (semicolon)
95	5F	0101 1111	? (logical NOT)
96	60	0110 0000	- (minus)
97	61	0110 0001	/ (slash)
98	62	0110 0010	§ (paragraph symbol)
99	63	0110 0011	[(left square bracket)
100	64	0110 0100	] (right square bracket)
...	...	...	
103	67	0110 0111	ß (special German character)
...	...	...	
106	6A	0110 1010	^ (logical AND)
107	6B	0110 1011	, (comma)
108	6C	0110 1100	% (percent)
109	6D	0110 1101	_ (underline)
110	6E	0110 1110	> (greater than)
111	6F	0110 1111	? (question mark)
....	....	....	...
122	7A	0111 1010	: (colon)
123	7B	0111 1011	# (number sign)
124	7C	0111 1100	@ (commercial at)
125	7D	0111 1101	' (apostrophe)
126	7E	0111 1110	= (equals)
127	7F	0111 1111	" (quote)
...	...	...	

Decimal	Hexadecimal	EBCDIC	Printer graphics
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
...	...	...	
162	A2	1010 0010	s
163	A3	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

Decimal	Hexadecimal	EBCDIC	Printer graphics
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)

3 Basic elements of a COBOL source program

3.1 General description

A COBOL source program is a syntactically correct set of COBOL statements.

With the exception of the COPY and REPLACE statements and the end program header, all statements, entries, paragraphs and sections of a COBOL source program are grouped into four divisions which are specified in the following sequence:

1. Identification Division
2. Environment Division
3. Data Division
4. Procedure Division

The end of a COBOL source program is indicated either by the end program header or by the absence of further source program lines.

The beginning of a division in a source program is indicated by the related program division header. The end of a division is indicated by

– the division header of a succeeding division in the program or

– the end program header or

– the absence of further source lines.

All separately compiled source programs in a sequence of source programs, except for the last source program in the sequence, must be terminated by an end program ENTRY. Although a COBOL source program may have an unlimited number of lines, the compiler will only number these lines uniquely up to a value of 65536.

3.2 Structure of a COBOL program

The following overall format shows, in detail, the general structure of a COBOL program.

```
[IDENTIFICATION DIVISION.]
[ID DIVISION.            ]

[PROGRAM-ID. program-name.

[AUTHOR.          [comment-entry]...]
[INSTALLATION.    [comment-entry]...]
[DATE-WRITTEN.    [comment-entry]...]
[DATE-COMPILED.   [comment-entry]...]
[SECURITY.        [comment-entry]...]

[  ENVIRONMENT DIVISION.          ]
[ [ CONFIGURATION SECTION.     ]  ]
[ [SOURCE-COMPUTER. [entry.]]  ]  ]
[ [OBJECT-COMPUTER. [entry.]]  ]  ]
[ [SPECIAL-NAMES.   [entry.]]  ]  ]

[ [ INPUT-OUTPUT SECTION.         ]  ]
[ [ FILE-CONTROL.     {entry.}...  ]  ]
[ [I-O-CONTROL.      [entry.]...]  ]  ]

[   DATA DIVISION.                                                    ]
[ [ FILE SECTION.                                                    ]  ]
[ [ [ file-description-entry.   {record-description-entry. }...    ]  ]  ]
[ [ [ sort-file-description-entry.  {record-description-entry. }...]...]  ]
[ [ [ report-file-description-entry.                               ]  ]  ]
[                                                                      ]
[ [  WORKING-STORAGE SECTION.          ]                                 ]
[ [ [77-level-description-entry. ]     ]                                 ]
[ [ [                            ]...  ]                                 ]
[ [ [record-description-entry.   ]     ]                                 ]
[                                                                      ]
[ [  LINKAGE SECTION.                  ]                                 ]
[ [ [77-level-description-entry. ]     ]                                 ]
[ [ [                            ]...  ]                                 ]
[ [ [record-description-entry.   ]     ]                                 ]
[                                                                      ]
[ [  REPORT SECTION.                                                   ]  ]
[ [                                                                    ]  ]
[ [ report-file-description-entry.  {report-group-description-entry. }...]...]  ]
[                                                                      ]
[ [  SUB-SCHEMA SECTION.         ]                                     ]
[ [  database-description-entry.]                                      ]

[   PROCEDURE DIVISION [USING {data-name-1}...].                      ]
[ [  DECLARATIVES.                                                   ]  ]
[ [ {section-name SECTION [segment-number].                          ]  ]
[ [      USE statement.                                              ]  ]
[ [ [paragraph-name.                                                 ]  ]
[ [     [sentence]...]...}...                                        ]  ]
[ [  END DECLARATIVES.                                               ]  ]
[                                                                      ]
[ [{section-name SECTION [segment-number]. ]                           ]
[ [[paragraph-name.                        ]                           ]
[ [     [sentence]...]...}...]             ]                           ]

[   END PROGRAM program-name.]
```

3.3 Structure of a nested source program

The format and sequence of the divisions that constitute a nested COBOL source program are discussed below.

The organization of a nested source program is discussed in detail in chapter 7, "Inter-program communciation".

Format

```
identification-division

[environment-division]

[data-division]

[procedure-division]

[nested-source-program]...

[end-program-header]
```

Syntax rules

1. The end program header must be present if:
   a) the COBOL source program contains one or more nested COBOL source programs, or if
   b) the COBOL source program is contained within another COBOL source program.

General rules

1. The beginning of a division is indicated by the appropriate division header. The end of a division is identified by one of the following:
   a) the division header of a succeeding division in this program.
   b) an Identification Division header which indicates the start of another source program.
   c) the end program header.

3.4 Sequence of programs

It is possible, in one compiler run, to compile several complete source programs which are stored sequentially in a file or a library member. For this, each source program within the sequence of programs must be terminated with an end program header. The last source program of the sequence of programs does not need to be terminated with an end program header.

```
{ IDENTIFICATION DIVISION.

  PROGRAM-ID. program-name-1.

[ ENVIRONMENT DIVISION. environment-division-content]

[ DATA DIVISION. data-division-content]

[ PROCEDURE DIVISION. procedure-division-content]

  END PROGRAM program-name-1. }...
```

3.5 End program header

Function

The end program header indicates the end of the COBOL source program.

Format

```
END PROGRAM program-name.
```

Syntax rules

1. The program-name must comply with the rules for forming user-defined words.
2. The specified program-name must be identical to the program-name declared in a preceding PROGRAM-ID paragraph of the source program.

General rules

1. If the next source statement after the program terminated by the END PROGRAM entry is a COBOL statement, then this must be the Identification Division of a program which is to be compiled separately from the program terminated by the end program header.
2. Only one space is permitted between END and PROGRAM.
3. program-name must be specified in the same line as END PROGRAM.

3.6 Identification Division

3.6.1 General description

Every COBOL program begins with the Identification Division. This division identifies the program by specifying the program-name.

In addition the user may specify the following: the date the program was written, the date the program is compiled, and other descriptive information explaining the purpose of the program.

3.6.2 Structure

General format

```
 A       Margin indication
 |
 |
 |
```

Every Identification Division paragraph except the PROGRAM-ID paragraph contains comment entries. A comment entry can be any combination of characters from the EBCDIC character set. The continuation of comment lines by the use of a hyphen in the indicator area is not permitted; however, the comment entry may extend over two or more lines.

The paragraphs with the names AUTHOR, INSTALLATION, DATE-WRITTEN and SECURITY serve only for documentation purposes of the user and are neither evaluated nor modified by the compiler.

3.6.3 Paragraphs

PROGRAM-ID paragraph

Function

The PROGRAM-ID paragraph provides the name by which a program is identified.

Format

```
PROGRAM-ID. program-name [program-attribute].
```

Syntax rules

1. The program-name must be a user-defined word. It must begin with a letter and the 8th character must not be a hyphen.
   A program name should not begin with the letter "I" to avoid conflict with names of COBOL85 runtime modules.
2. The operating system uses only the first eight characters of program-name for identifying the module. Therefore, these characters should be unique for every name in a particular module/program library.
3. The program attributes (INITIAL clause, COMMON clause) are described in chapter 7, "Inter-program communication".

DATE-COMPILED paragraph

Function

The DATE-COMPILED paragraph causes the compilation date to be inserted in the source program listing.

Format

```
DATE-COMPILED. [comment-entry]...
```

Syntax rules

1. A comment-entry may consist of any combination of the machine-specific character set.
2. The whole comment-entry is replaced with the current date (including the century). Any comment-entry lines within the DATE-COMPILED paragraph are left intact.

3.7 Environment Division

3.7.1 General description

The Environment Division provides a standard method for describing those aspects of a data processing problem which depend on the physical characteristics of a given computer installation. This division can be used to define the equipment configuration of the data processing system on which the program is to be compiled and executed. It also provides an opportunity for specifying input/output control, specific machine characteristics, and control techniques.

The Environment Division is optional in a COBOL source program.

The Environment Division consists of two optional sections:

1. CONFIGURATION SECTION
2. INPUT-OUTPUT SECTION.

The CONFIGURATION SECTION deals with the characteristics of the computers used for compiling and executing the program (source computer and object computer, respectively).

This section is divided into three paragraphs:

1. the SOURCE-COMPUTER paragraph, which describes the equipment configuration of the data processing system on which the source program is to be compiled
2. the OBJECT-COMPUTER paragraph, which describes the equipment configuration of the data processing system on which the object program is to be executed
3. the SPECIAL-NAMES paragraph, which (among other things) relates the implementor-names used by the compiler to the mnemonic-names used in the source program.

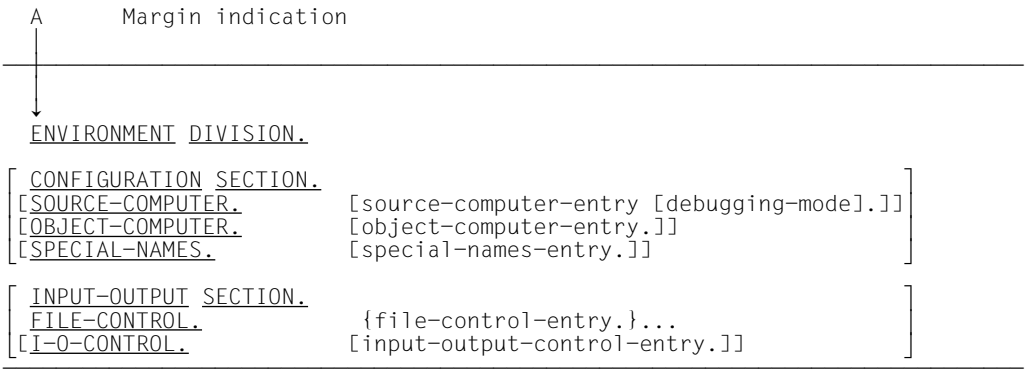
The INPUT-OUTPUT SECTION deals with the required information for controlling the transmission of data between external devices and the object program.

This section is divided into two paragraphs:

1. the FILE-CONTROL paragraph, which names the files and assigns them to external devices
2. the I-O-CONTROL paragraph, which describes special control techniques to be used in the object program.

The INPUT-OUTPUT SECTION is discussed in chapters 4 through 6, which deal with file processing.

General format



The Environment Division is optional.

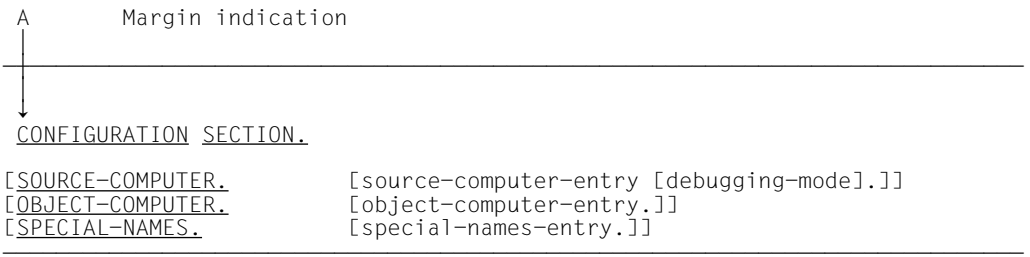
3.7.2 CONFIGURATION SECTION

Function

The CONFIGURATION SECTION makes up part of the Environment Division of a source program and provides a means to do the following:

– describe the computer configuration on which the program is to be compiled or executed
– declare a currency sign
– choose the decimal point
– specify symbolic names for characters
– relate implementor-names to user-specified mnemonic-names
– relate alphabet-names to character sets or collating sequences
– relate class-names to user-defined sets of characters

Format



Syntax rules

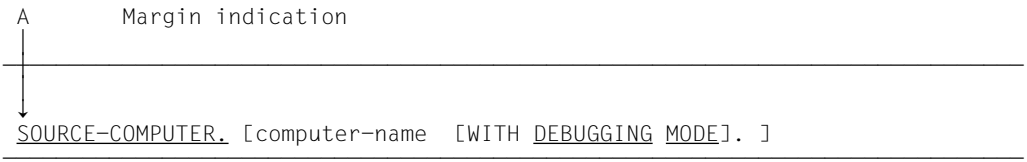
1. The CONFIGURATION SECTION and its associated paragraphs are optional.
2. If this paragraph is specified, the sequence indicated must be observed.

SOURCE-COMPUTER paragraph

Function

The SOURCE-COMPUTER paragraph identifies the data processing system on which the source program is to be compiled; it may also be used to specify debugging aids.

Format



Syntax rule

computer-name must be a user-defined COBOL word.

General rules

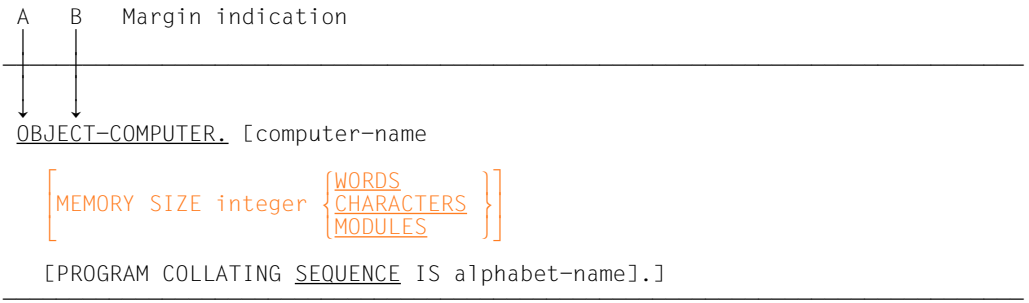
1. All clauses of this paragraph apply to the program in which they are explicitly or implicitly specified.
2. When the paragraph is not specified, or if it is specified but the computer-name is not, the computer upon which the source program is being compiled is the source computer.
3. If the WITH DEBUGGING MODE clause is specified, all debugging lines are compiled as specified in the rules described in section 3.10 (page 353ff).
4. If the WITH DEBUGGING MODE clause is not specified in a program, any debugging lines are compiled as if they were comment lines.

OBJECT-COMPUTER paragraph

Function

The OBJECT-COMPUTER paragraph describes the data processing system on which the program is to be executed.

Format



Syntax rules

1. computer-name and the MEMORY-SIZE clause are used for documentation purposes only and are treated as comments.
2. computer-name must be a user-defined COBOL word.
3. If the PROGRAM COLLATING SEQUENCE clause is specified, the collating sequence associated with alphabet-name (see "SPECIAL-NAMES paragraph", page 125) is used to determine the truth value of any nonnumeric comparisons:
   a) explicitly specified in relation conditions
   b) explicitly specified in condition-name condition
   c) implicitly specified by the presence of a CONTROL clause in a report description entry (see "CONTROL clause", page 579).
4. If the PROGRAM COLLATING SEQUENCE clause is not specified, the native collating sequence is used (EBCDIC).
5. The PROGRAM COLLATING SEQUENCE clause is also applied to any nonnumeric merge or sort keys unless the COLLATING SEQUENCE phrase of the respective MERGE or SORT statement is specified (see "MERGE statement", page 655 and "SORT statement", page 663).

For examples of the use of the PROGRAM COLLATING SEQUENCE and ALPHABET clauses see "SPECIAL-NAMES paragraph" (page 125).

SPECIAL-NAMES paragraph

Function

The SPECIAL-NAMES paragraph provides a means of

1. relating system-names to mnemonic-names specified by the user
2. relating alphabet-names to character sets and/or collating sequences
3. defining symbolic names for characters
4. relating class-names to character sets
5. supplying a character which is to be used as the currency sign in picture-strings
6. exchanging the functions of the comma and the decimal point in picture-strings and in numeric literals.

Format

```
A   B   Margin indication
|   |
|   |
|   |
```

General rule

The individual clauses of the SPECIAL-NAMES paragraph must, if they are used, be specified in the order given in the format.

The individual clauses of the SPECIAL-NAMES paragraph are described below.

Implementor-name

Syntax rules

1. implementor-name is a system-name and must be a name from the left column of the following table.

   Implementor-names and their meanings:

Implementor-name	Meaning
CONSOLE	System or main console or subconsole
TERMINAL	The user' s data display unit
SYSIPT	System logical input file
PRINTER PRINTER01-PRINTER99	System logical printer file
SYSOPT	System logical output file
C01 to C08	Skip to channel 1 through 8
C10 to C11	Skip to channel 10 or 11
JV-job-variable-name	Job variable describing the link name of a job variable (see below)
TSW-0 to TSW-31	Task switches
USW-0 to USW-31	User switches
COMPILER-INFO	Compiler information
CPU-TIME, PROCESS-INFO, TERMINAL-INFO DATE-ISO4	Operating system information

Table 3-1: Implementor-names and their meanings

2. job-variable-name indicates a BS2000 job variable. It is a COBOL word of up to 7 characters and is used to form the link name *job-variable-name and for accessing the job variable (see example 3-1).

General rules

1. If implementor-name is a user or task switch, at least one condition-name must be associated with it. The status of the switches is described under "Condition-names", and can be interrogated by testing the condition-name (see "Switch-status conditions", page 222).

   The status of a switch may be altered by using a format 3 SET statement (see "SET statement", page 338).

2. C01 through C08, C10 and C11 will not be supported in the next version of the COBOL85 compiler.
   If C01 through C08, C10 or C11 is specified as implementor-name, the associated mnemonic-name may be used only in a WRITE statement with ADVANCING phrase.

Example 3-1

Use of job variables:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. JVTEST.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    JV-JV1 IS JOB-VAR-1.
PROCEDURE DIVISION.
M SECTION.
M-PAR.
    DISPLAY "xyz" UPON JOB-VAR-1.
    STOP RUN.
```

Prior to the program call:

```
/SET-JV-LINK LINK-NAME=*JV1,JV-NAME=JV1TEST
```

ARGUMENT-NUMBER / ARGUMENT-VALUE / ENVIRONMENT-NAME / ENVIRONMENT-VALUE

are extensions from the X/OPEN Portability Guide. They are used only in conjunction with ACCEPT and DISPLAY statements and are described under those statements.

ALPHABET clause

Syntax rules

1. If the literal phrase is specified in the ALPHABET clause, any given character for literal-1, literal-2 etc. which is referenced by alphabet-name in the PROGRAM COLLATING SEQUENCE clause (see "OBJECT-COMPUTER paragraph", page 124) or in the COLLATING SEQUENCE phrase of the SORT or MERGE statement (page 663 and page 655 respectively) may be used once only (see examples 3-10 and 3-11).
2. The following rules apply to literal-1, ..., literal-11:
   a) If the literals are numeric, they must be unsigned integers with a value from 1 to 256.
   b) If the literals are nonnumeric and associated with the THROUGH, THRU or ALSO phrase, each literal must be one character long.
   c) The literals must not specify a symbolic-character figurative constant. Literal-6 must not be a figurative constant.
3. The words THROUGH and THRU are equivalent.
4. The NATIVE and EBCDIC phrases mean the same thing in BS2000.

General rules

1. The ALPHABET clause provides a means for relating a name to a particular character set and/or collating sequence. When referenced in the PROGRAM COLLATING SEQUENCE clause (see "OBJECT-COMPUTER paragraph", page 124) or the COLLATING SEQUENCE phrase of a SORT or MERGE statement (page 663 and page 655 respectively) alphabet-name specifies a collating sequence. When alphabet-name-1 is referenced in the SYMBOLIC-CHARACTERS clause or in a CODE-SET clause of a file description entry (for sequentially organized files), the ALPHABET clause specifies a character set.
   a) If the STANDARD-1 phrase is specified, the character set or collating sequence is that defined in the American National Standard Code for Information Interchange (ASCII), X3.4-1968.
   b) If the STANDARD-2 phrase is specified, the character set identified is the International Reference Version of the ISO 7-bit code, as defined in International Standard 646, "7-Bit Coded Character Set for Information Processing Interchange". Each character of the standard character set is associated with a corresponding character from the native character set.
   c) If the NATIVE or EBCDIC phrase is specified, the native character set or native collating sequence is used (EBCDIC).

d) If the literal phrase of the ALPHABET clause is specified, the alphabet-name must not be referenced in a CODE-SET clause (see "CODE-SET clause", page 382).

- The value of the literal specifies the ordinal number of a character (beginning with 1) within the native character set, if the literal is numeric. This value must not exceed the number of characters in the native character set (256).
- The value of the literal specifies the actual character within the native character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal is inserted into the collating sequence in the order specified (see example 3-2).
- The order in which the literals appear in the ALPHABET clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified (see example 3-3).
- Any characters within the native collating sequence which are not explicitly specified in the literal phrase assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.
- If the THROUGH/THRU phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH/THRU phrase may contain characters of the native character set in either ascending or descending sequence (see example 3-4).
- If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1 and literal-3 are assigned to the same ordinal position in the collating sequence being specified or in the character set (see example 3-5).
  If alphabet-name-1 is referenced in a SYMBOLIC CHARACTERS clause, only literal-1 is used to represent the character in the native character set.

2. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE (see examples 3-6 and 3-7).
3. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE (see examples 3-8 and 3-9).

Example 3-2

```
ALPHABET ALPHATAB IS "AJKCDF".
```

First character is A
Second character is J
.
.
Sixth character is F

Example 3-3

```
ALPHABET ALPHATAB IS "A" "C" "D" "Z".
```

First character in the collating sequence is "A"
Second character in the collating sequence is "C"
Third character in the collating sequence is "D"
Fourth character in the collating sequence is "Z"

Example 3-4

```
ALPHABET ALPHATAB IS "A" THRU "I".
```

First character is A
Second character is B
Third character is C
.
.
Eighth character is H
Ninth character is I

Example 3-5

```
ALPHABET ALPHATAB IS "A" ALSO "B" ALSO "C" ALSO "D".
```

The characters A, B, C, and D will be associated with the lowest ordinal positions in the collating sequence.

Example 3-6

```
ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 194.
```

The highest ordinal position in the collating sequence is occupied by the character which appears in the 194th position of the native character set, i.e. the character A.

A is associated with the figurative constant HIGH-VALUE.

Example 3-7

```
ALPHABET ALPHATAB IS 193 THRU 1, 255 THRU 197, "A" ALSO "B" ALSO "C".
```

Positions 1 through 193 of the collating sequence are associated with the characters which appear at positions 193 to 1 of the native character set.

Positions 194 through 253 of the collating sequence are associated with the characters which appear at positions 255 to 197 of the native character set.

Position 254 is assigned the characters A, B, C; with this all characters in the native character set are associated with a position in the collating sequence. The highest-order position (254) is occupied by the characters A, B, C. Being the character specified last, C is associated with the figurative constant HIGH-VALUE.

Example 3-8

```
ALPHABET ALPHATAB IS "0" "1" "2".
```

The lowest ordinal character in the collating sequence is 0. Hence 0 is associated with the figurative constant LOW-VALUE.

Example 3-9

```
ALPHABET ALPHATAB IS "A" ALSO "B" ALSO "C".
```

The lowest ordinal position in the collating sequence is occupied by the characters A, B, C. The character A, which was specified first, is associated with the figurative constant LOW-VALUE.

Example 3-10

PROGRAM COLLATING SEQUENCE and ALPHABET clauses:

```
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 "Collating sequence ok" UPON T
    END-IF
    STOP RUN.
```

With the definition of the alphabet-name ALPHATAB in the SPECIAL-NAMES paragraph, the character X was assigned to the first position in the collating sequence, Y to the second and Z to the third.

All remaining characters of the native character set are assigned a position in the collating sequence implicitly, since their positions in the collating sequence are higher than those of the specified characters X, Y, Z and their order in the collating sequence was taken from the native character set without alteration.

Positions 1 through 231 in the native character set correspond to positions 4 through 234 in the collating sequence.

Positions 235 through 256 in the native character set correspond to positions 235 through 256 in the collating sequence.

Thus, A occupies position 197, B position 198, and C position 199.

Hence, the relation ITEM-1 > ITEM-2 is **true**.

Example 3-11

PROGRAM COLLATING SEQUENCE and ALPHABET clauses:

```
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.
COMPARISON.
    IF ITEM-1 = ITEM-2
    THEN
      DISPLAY "First relation ok" UPON T
    ELSE
      DISPLAY "First relation not ok" UPON T
    END-IF
    IF ITEM-2 = HIGH-VALUE
    THEN
      DISPLAY "Second relation ok" UPON T
    ELSE
      DISPLAY "Second relation not ok" UPON T
    END-IF.
FINISH-PAR.
    STOP RUN.
```

Characters less than 7 remain as in native collating sequence. Characters greater than 9 are then appended, thereby becoming less than 7.
The characters 7, 8, 9 are set at the highest ordinal position, with 9, being the last character specified, corresponding to "HIGH-VALUE".

Result:

First relation OK
Second relation OK

SYMBOLIC CHARACTERS clause

Syntax rules

1. No symbolic name for a character may be used more than once in the SYMBOLIC CHARACTERS clause.
2. The relationship between each separate symbolic name and its corresponding integer results from the sequence within the SYMBOLIC CHARACTERS clause: symbolic-character-1 is paired with integer-1, symbolic-character-2 with integer-2, and so on.
3. An integer must be specified for each symbolic name which is specified.
4. The position specified within the collating sequence by integer-1 must exist in the native character set. If IN is specified, the position must exist in the character set named by alphabet-name-2.
5. The internal representation of symbolic-character is identical to that of the corresponding character in the native character set or in the character set specified with alphabet-name-2.
6. symbolic-character is a figurative constant.

Example 3-12

```
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 PRINT-RECD.
   02 CNTRLBYTE         PIC X.
   02 PRINT-LINE        PIC X(132).
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    MOVE HEX-0A TO CNTRLBYTE.
    DISPLAY CNTRLBYTE UPON T.
    STOP RUN.
```

The symbolic name HEX-0A is assigned to the eleventh character of the EBCDIC character set (this character corresponds to the hexadecimal value 0A).
The MOVE statement uses this symbolic name in order to move the hexadecimal value 0A into the control byte.

CLASS clause

Syntax rules

1. The CLASS clause enables the user to associate a name with the character set defined in this clause. This class-name may be referenced in a class condition only. Characters specified by the values of literal-4, literal-5, ... form the exclusive character set named by class-name.

   The value of each literal specifies:

   a) The ordinal number of a character in the native character set, if the literal is numeric.

   b) The actual character in the native character set, if the literal is nonnumeric. If the value of this nonnumeric literal contains more than one character, each of these characters is included in the character set named by class-name.

2. If THROUGH is specified, the contiguous characters in the native character set beginning with literal-4 and ending with literal-5 are included in the special character set identified by class-name. The THROUGH phrase may be used to specify this characterstring in either ascending or descending order.

Example 3-13

```
SPECIAL-NAMES.
    CLASS HEXADECIMAL-CHARACTERS
    194 THRU 199, 241 THRU 250.
```

194 through 199 corresponds to the letters A through F
241 through 250 corresponds to the digits 0 through 9

CURRENCY SIGN clause

Syntax rules

1. When used in the CURRENCY SIGN clause, literal-6 is limited to a single character, which must not be one of the following:
   - digits 0 through 9
   - uppercase letters A, B, C, D, P, R, S, V, X, Z, or the space
   - lowercase letters a - z
   - special characters
     - * (asterisk)
     - + (plus)
     - − (minus)
     - , (comma)
     - . (period)
     - : (colon)
     - ; (semicolon)
     - ((left parenthesis)
     -) (right parenthesis)
     - " (quotation mark)
     - / (slash)
     - = (equal sign)
2. If the CURRENCY SIGN clause is not present, only the currency sign $ may be used as the currency symbol in a PICTURE string.

DECIMAL-POINT IS COMMA clause

Syntax rule

The DECIMAL-POINT IS COMMA clause means that the functions of comma and period are exchanged in the character-string of the PICTURE clause and in numeric literals.

General rule

The DECIMAL-POINT IS COMMA clause is used to exchange the functions of the decimal point and the comma in numeric literals and in picture-strings. When this clause is used, the decimal point required in a numeric literal or in a picture-string must be represented by a comma. The decimal point, in turn, must be used for the function normally performed by the comma.

3.8 Data Division

3.8.1 General description

Two types of data are processed by a COBOL program:

1. data stored on some external medium
2. data generated internally during program execution.

The first type of data is combined into records in files; the second type of data must be declared by the user as records or subordinate data items.

To ensure maximum independence of data from its specific representation on external volumes and in data processing systems, the properties or contents of data are described with respect to a standard data format rather than a system-oriented format. This format is adapted to the general application of data processing and uses decimal numbers for representing numbers and the rest of the COBOL character set for representing nonnumeric characters.

The data description method used allows a distinction to be made between the physical and system-dependent properties on the one hand, and conceptual characteristics on the other.

Physical and certain system-specific properties of data on an external medium are defined in a COBOL source program in order that efficient use may be made of special techniques.

1. The term "physical properties of data" refers to
   a) the grouping of logical records within the physical boundaries of the external volume
   b) the recording mode in which the data is stored on the external volume.
2. The term "system-dependent properties of data" refers to the description under which a file is identified on some external medium.

Most of these properties are described in chapters 4, 5 and 6 under the different types of file organization.

The conceptual characteristics of data on an external volume pertain to the logical units of data, the logical records, and are not associated with any physical or system-specific properties.

These characteristics are described:

for files:	in chapters 4, 5 and 6;
for records:	in the Data Division record description entries.

A data description in a COBOL program is separate from the declaration of execution procedures. This permits the programmer a great number of options for modifying a data description entry without any change to the procedures which are related to that entry. Therefore, to a certain extent, the procedures of a COBOL program may be seen as data-independent.

Structure

The Data Division is one of the divisions of a source program. It is divided into five sections as follows:

1. FILE SECTION,
   see chapters 4 to 6, "File organization"
2. WORKING-STORAGE SECTION
3. LINKAGE SECTION,
   see chapter 7, "Inter-program communication"
4. REPORT SECTION,
   see chapter 8, "Report Writer"
5. SUB-SCHEMA SECTION
   (For further information see the "UDS Reference Manual" [6]).

All data which is, or is meant to be, stored on external media must first be described in the **FILE SECTION** before it can be processed by a COBOL program.

Information intended for internal use must be described in the **WORKING-STORAGE SECTION**:

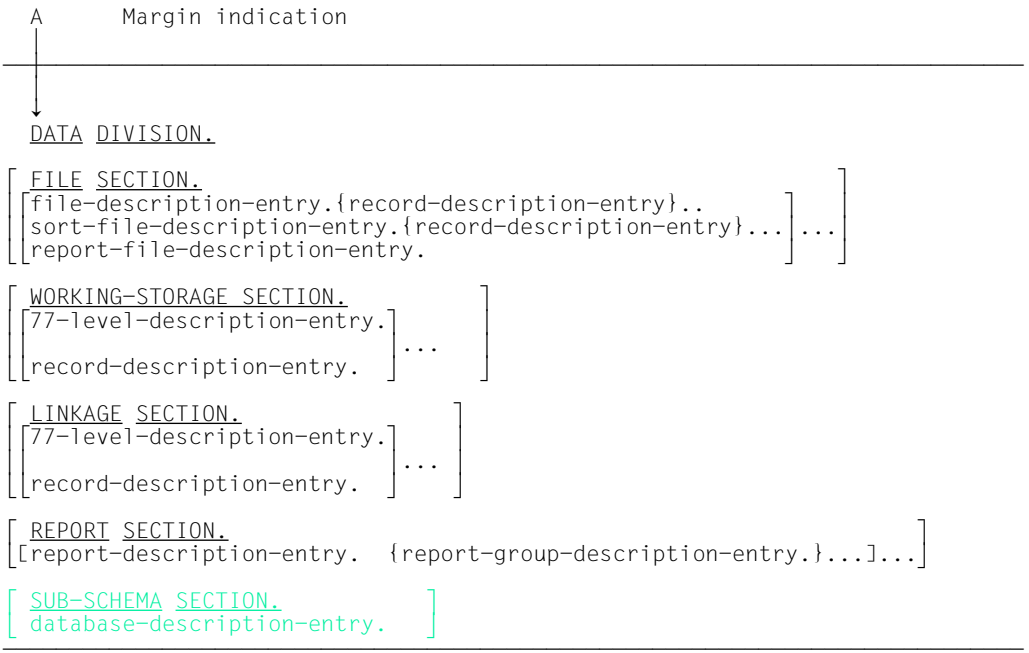
Information transmitted from one program to another must be described in the **LINKAGE SECTION**.

The contents and appearance of all listings created by the Report Writer must be described in the **REPORT SECTION**.

All information pertaining to the description of database structures must be entered in the **SUB-SCHEMA SECTION**. (For further details see the "UDS Reference Manual" [6]).

The following pages present the general format for the sections of the Data Division and define the order in which they should be entered in the source program.

General format

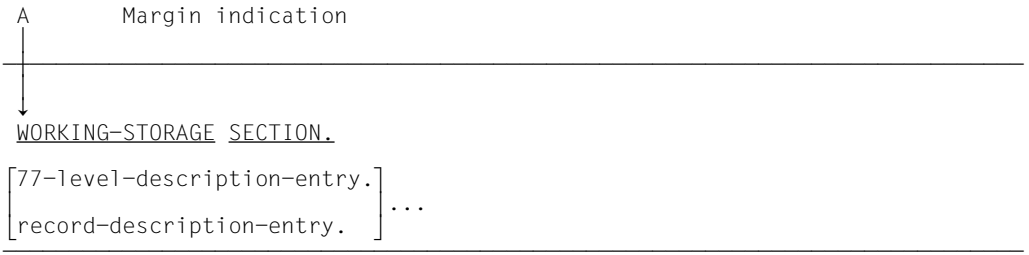


3.8.2 WORKING-STORAGE SECTION

Function

The WORKING-STORAGE SECTION describes records and structurally noncontiguous data items (refer to "General format") which are not part of external files.

Format



Data description entry

General description

Data description entry is the general term for the description of every single data item in the Data Division; the entry is composed of the level number, followed by a data-name (if necessary) and several data clauses.

The **record description entry** is used to define all data description entries which are associated with a particular record; that is, the record description entry describes all properties of that record.

Multiply-defined 01- and 77-level record description entries are not treated as errors provided they are not used in the Procedure Division.

The concept of level is contained in the structure of a logical record. This concept arises from the need of assigning names to the parts of a record in order to access them. Once a record has been thus subdivided, the subdivision can be carried further to permit even more detailed data references.

A "report group" is to the REPORT SECTION what a "record" is to other sections of the Data Division. The **report group description entry** describes all data description entries associated with a particular report group. Within a report group description entry, a distinction is made between the first and the subsequent data description entries (see chapter 8, "Report writer").

Those components of a record which are not further subdivided are called **elementary items**; a record thus either consists of a sequence of elementary items or is itself an elementary item.

An elementary item may be at the most 65535 bytes long.

In order to reference a number of elementary items at one time, these items are arranged into **"groups"** or **"group items"**. These groups may in turn be arranged into sets of two or more groups. Consequently, an elementary item may belong to more than one group.

The word **"data item"** is used in those cases where there is no need to distinguish between elementary and group items.

Organization of the entries of the Data Division

Table 3-2 shows the permitted level numbers and their associated Data Division entries.

Level-number	Use
01	Record description entries
02 - 49	Data description entries describing subdivisions of a record
77	Description entries for independent or noncontiguous data items which are not subdivisions of other items and are not themselves subdivided
66	Elementary items or group items described by the RENAMES clause for the purpose of regrouping data items (see "RENAMES clause", page 183)
88	Condition-name entries to specify condition-names associated with particular values of a conditional variable (see "VALUE clause", page 202)

Table 3-2: Summary of level numbers

Level numbers are used in structuring a logical record so that subdivisions of the record may be referenced. Once a record has been subdivided, this structuring may be carried further to permit even more detailed data references.

The level numbers 01 and 77 must be written starting in area A, followed by their associated data names and appropriate descriptive information. All other level numbers may begin either at margin A or margin B, followed, from margin B, by associated data names and appropriate descriptive information.

Consecutive data description entries may have the same format as the first such entry or may be intended according to their level numbers. While indentation is helpful for documentation purposes, it does not affect the compiler.

For further rules, refer to "Clauses for data description" (page 148).

Data description entry formats

Function

A data description entry describes the attributes of a single data item.

Format 1

```
A       Margin indication
|
|
|
v

                [data-name]
level-number    [          ]
                [FILLER    ]

                [REDEFINES clause]
                [BLANK WHEN ZERO clause]
                [DYNAMIC clause]
                [EXTERNAL clause]
                [GLOBAL clause]
                [JUSTIFIED clause]
                [OCCURS clause]
                [PICTURE clause]
                [SIGN clause]
                [SYNCHRONIZED clause]
                [USAGE clause]
                [VALUE clause]
```

Syntax rules

1. level-number may be any number from 01 through 49 or 77.

   Descriptions of level 77 specify data items that are not hierarchically related and are not subdivided into smaller parts.

2. The clauses may be written in any order, except for FILLER and the data-name phrase, and also the REDEFINES clause. When used, these must be written as shown in the format. In the following discussion, the clauses appear in alphabetical order. The EXTERNAL and GLOBAL clauses are described in chapter 7, "Inter-program communication".

3. The PICTURE clause must be specified for every elementary item except index data items and internal floating-point data items.

4. The OCCURS clause must not be specified in a data description entry that has a level-number of 01, 66, 77 or 88.

Example 3-14

for the structure of a record with description of group items

```
01  RECORD.   ————————————————————— Logical record
    02  REF-NO PIC ...  ———————————— Elementary item
    02  CUSTMR-NO PIC ...  ————————— Elementary item
    02  ADDRESS.  ————————————————— Group item        ⎫                ⎫
        03  FIRST-NAME PIC ... ——— Elementary item  ⎬ Group item  ⎪
        03  LAST-NAME PIC ... ———— Elementary item  ⎪                ⎪
        03  STATE PIC ... ————————— Elementary item  ⎭                ⎬ Group item
        03  CITY.  ——————————————— Group item        ⎫                ⎪
            04  ZIP-CODE PIC ... - Elementary item  ⎬ Group item  ⎪
            04  CITY PIC ... ———— Elementary item  ⎭                ⎪
        03  STREET PIC ... ————————— Elementary item                ⎭
    02  ART-NO PIC ...  ———————————— Elementary item
    02  PRICE.  ——————————————————— Group item        ⎫
        03  DOMES PIC ... ————————— Elementary item  ⎬ Group item
        03  FORGN PIC ... ————————— Elementary item  ⎭
```

The group item contains no information on data class or size of item. Definitions (e.g. REDEFINES, OCCURS) can, however, follow the group item name. The entry ends with a period.

Format 2

```
66 data-name-1 RENAMES data-name-2 [{THROUGH} data-name-3].
                                   {THRU   }
```

For syntax rules and general rules, see the "RENAMES clause" (page 183).

Format 3

```
88 condition-name {VALUE IS  } {literal-1 [{THROUGH} literal-2] }...
                  {VALUES ARE}            {THRU   }             }
```

For syntax rules and general rules, see format 2 of the "VALUE clause" (page 204).

Level number

Function

The level number indicates the position of a data item within the hierarchical structure of a logical record (see chapter 2, "Glossary"). It also identifies entries for data items within the WORKING-STORAGE and LINKAGE SECTIONs, as well as for condition-names and data-items in the RENAMES clause.

Format

```
level-number
```

Syntax rules

1. The level number is a special numeric literal consisting of one to two digits. A level number which is less than 10 may be written either as a single digit or with a leading zero.
2. Level numbers 01 and 77 must be entered starting at margin A. All other level numbers may begin in area A or area B.
3. Data description entries subordinate to an FD or SD entry must have level numbers with the values 01 to 49, 66, or 88.
4. Data description entries subordinate to an RD entry may have the level numbers 01 and 02 only.
5. Data description entries in the WORKING-STORAGE or LINKAGE SECTION must have level numbers with the values 01 to 49, 77, 66, or 88.
6. The first element in every data description entry must be a level number.

General rules

1. Level number 01 identifies the first entry of each record description or report group description.
2. Special level numbers are assigned to certain kinds of entries for which there is no real concept of hierarchy. These numbers include:
   Level number 66 is used to identify renaming entries. It may be used only as described in the RENAMES clause.
   Level number 77 is used to identify structurally noncontiguous data items in the WORKING-STORAGE and LINKAGE SECTIONs. It may be used only as described under "77-level description entry".

Level number 88 refers to entries which define condition-names associated with a conditional variable. It may be used only as described in format 2 of the VALUE clause.

3. Multiple level-01 entries which are subordinate to a given level indicator (except RD) represent implicit redefinitions of the same area.

Example 3-15

```
01  ADDRESS.
    02  NAME.
        03  FIRST-NAME                PIC X(18).
        03  LAST-NAME                 PIC X(20).
    02  STREET ADDRESS.
        03  ZIP-CODE.
            04  DIGIT-1               PIC 9.
            04  DIGIT-2               PIC 9.
            04  DIGIT-3               PIC 9.
            04  DIGIT-4               PIC 9.
            04  DIGIT-5               PIC 9.
        03  CITY                      PIC X(19).
        03  STREET                    PIC X(16).
        03  HOUSE-NUMBER              PIC XXX.
```

The statement

```
MOVE ADDRESS TO...
```

will move the entire group.

The statement

```
MOVE NAME TO...
```

will move the first and last names etc.

3.8.3 Clauses for data description

BLANK WHEN ZERO clause

Function

The BLANK WHEN ZERO clause specifies that an item is to be set to blanks when its value is zero.

Format

```
BLANK WHEN ZERO
```

Syntax rules

1. The BLANK WHEN ZERO clause may be specified only at the elementary level for numeric-edited or numeric items.
2. The numeric or numeric edited data description entry to which the BLANK WHEN ZERO clause applies must be described, either implicitly or explicitly, as USAGE IS DISPLAY.

General rules

1. When the BLANK WHEN ZERO clause is used, the item will contain only blanks if the value of the item is zero.
2. When the BLANK WHEN ZERO clause is used for numeric data items, the category of the item is considered to be numeric-edited.
3. If the BLANK WHEN ZERO clause and the PICTURE clause with asterisk (*) (for zero suppression) are used simultaneously in a data description entry, the zero suppression editing function overrides the function of the BLANK WHEN ZERO clause (see "PICTURE clause", page 164).

Example 3-16

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BWHENZ.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PURCHASE-EXAMPLE.
   02 PURCHASE PICTURE $Z.99 BLANK WHEN ZERO.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    MOVE ZERO TO PURCHASE.
    DISPLAY PURCHASE UPON T.
    STOP RUN.
```

Value of PURCHASE after the MOVE statement:

␣␣␣␣␣ (5 spaces)

DYNAMIC clause

Function

The DYNAMIC clause enables the dynamic provision of memory in a scope defined by the user.

Format

```
01 data-name IS DYNAMIC.
```

(level-number and data-name are not part of the DYNAMIC clause; they are specified here simply to improve clarity)

Syntax rules

1. The DYNAMIC clause may only be specified in level-01 record description entries in the WORKING-STORAGE SECTION.
2. If the DYNAMIC clause is specified for a data item, no other clause may be specified for this data item.

General rule

The data item to which the DYNAMIC cause is applied is set up in main memory at object time and begins on a 4-Kbyte boundary.

Data-name or FILLER clause

Function

A data-name specifies the data being described. The reserved word FILLER specifies an elementary or group item which is never referenced explicitly and therefore need not be given a name.

Format

```
level-number [data-name]
             [FILLER   ]
```

(The level number is not part of the data-name or FILLER clause; it is shown here merely for purposes of clarity.)

Syntax rules

1. data-name must be formed according to the rules for user-defined words.
2. In the FILE, WORKING-STORAGE and LINKAGE SECTIONs, the first word in a data description entry following the level number must be a data-name or the reserved word FILLER.
3. The reserved word FILLER is used to give a name to an elementary item or group item which is never referenced in the program, and therefore does not require a data-name. A FILLER data item cannot be referenced directly.
4. If the data-name or FILLER entry is omitted, FILLER is assumed.

General rule

All referenced 01- and 77-level entries in the WORKING-STORAGE and LINKAGE SECTIONs must, if they are to be referenced, be given unique data-names, since neither type of entry can be qualified. A subordinate data-name need not be unique if it can be qualified in a manner which makes it unique.

Example 3-17

```
01 REC.
   02 FIRST-NUMBER        PICTURE 9(8).
   02 SECOND-NUMBER       PICTURE 9(12).
   02 FILLER              PICTURE X(60).
```

Here, a record is identified by the data-name RECORD, and its first two fields are identified by the data-names FIRST-NUMBER and SECOND-NUMBER. Since the third field is not referenced in the program, its level number is followed by the reserved word FILLER.

JUSTIFIED clause

Function

The JUSTIFIED clause permits nonnumeric data to be aligned within a nonnumeric receiving item in an alternative manner to the standard.

Format

Syntax rules

1. JUST is the abbreviation of JUSTIFIED.
2. The JUSTIFIED clause can be specified for elementary items only.
3. The JUSTIFIED clause cannot be specified for numeric or edited data items.
4. The JUSTIFIED clause must not be specified for data items with level number 66 or 88.
5. The JUSTIFIED clause must not be specified for an index data item.
6. The JUSTIFIED clause must not be specified for a receiving item of a STRING statement (see "STRING statement", page 341).

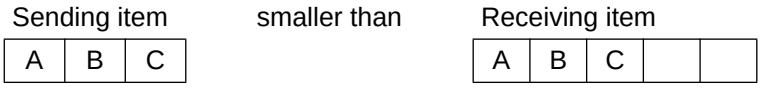
General rules

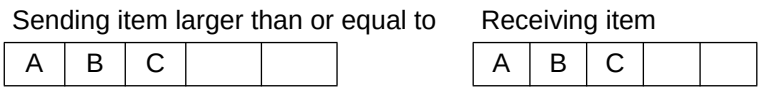
1. If the JUSTIFIED clause is specified for the receiving item, and the sending item is longer than the receiving item, the data is aligned at the rightmost character position, and the leftmost characters of the sending item are truncated.

   If the JUSTIFIED clause is specified for the receiving item, and the receiving item is longer than the sending item, the data is aligned at the rightmost character position, and the leftmost character positions of the receiving item are filled with blanks.
2. When the JUSTIFIED clause is omitted, the standard rules for data alignment within an elementary item are applicable (see under "Alignment", page 72).

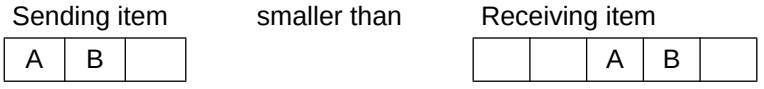
Example 3-18

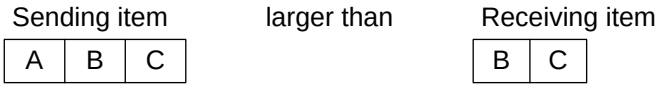
Normal alignment (without JUSTIFIED):





Alignment when JUSTIFIED clause is specified:





OCCURS clause

Function

The OCCURS clause is used to define tables. It specifies how many elements a table is to have, i.e. how often a data item is to recur. All elements have the same format. The size of the table may be variable. Furthermore, indices can be supplied.

Format 1 specifies the exact number of occurrences of a data item.

Format 2 specifies a variable number of occurrences of a given data item, ranging between a **maximum** and a **minimum** number of occurrences.
The minimum number may be omitted.

Format 1

```
OCCURS integer-2 TIMES

[{ASCENDING }                                        ]
[{          } KEY IS data-name-1 [data-name-2]...]...
[{DESCENDING}                                        ]

[INDEXED BY {index-1}...]
```

Syntax rules for format 1

1. integer-2 represents the exact number of occurrences.
2. integer-2 must be greater than 0.
3. data-name-1 may either be the subject of the OCCURS clause (in which case data-name-2,... must not be specified), or it must be subordinate to the group item referenced by the OCCURS clause.
4. data-name-2,... must be subordinate to the group item referenced by the OCCURS clause..
5. If data-name-1 does not match the subject of the OCCURS clause, the following rules apply:
   a) data-name-1, data-name-2,... must be subordinate to the group item referenced by the OCCURS clause.
   b) data-name-1, data-name-2,... must not be described with an OCCURS clause. Furthermore, they must not be subordinate to an entry which contains an OCCURS clause.
   c) data-name-1, data-name-2,... may be qualified with OF or IN (see "Qualification", page 77).

6. data-name-1, data-name-2,... are subject to the following additional rules:
   a) Up to 12 key fields may be specified for a given table element.
   b) The sum of the lengths of all key fields associated with a table element must not exceed 256.
   c) The key fields may have the data formats DISPLAY, BINARY, COMPUTATIONAL, COMPUTATIONAL-5 or COMPUTATIONAL-3 or PACKED-DECIMAL.
7. index-1... must be unique words in the program; otherwise, the indices specified by the INDEXED BY phrase are not defined in the program. Up to 12 index-names may be specified.
8. The OCCURS clause must not be specified in a data description entry that:
   a) has a level number of 01, 66, 77 or 88, or
   b) describes an item whose size is variable (the size of an item is variable if the data description entry of any item subordinate to it contains an OCCURS clause with the DEPENDING phrase).
9. The ASCENDING/DESCENDING KEY phrase defines whether the elements in the table are to be arranged in ascending or descending order, according to the values contained in data-name-1, data-name-2, etc. The data-names must be listed in descending order of significance.

   The user is responsible for seeing that the table elements are sorted properly (this order is presupposed in the SEARCH ALL statement).
10. The INDEXED BY phrase indicates that indexing may be used to reference the data-name which is the subject of the OCCURS clause, or any entry subordinate to that data-name. The storage allocation and format for indices are automatically defined by the compiler.
11. Each index contains a binary value that represents a displacement from the beginning of the table, corresponding to an occurrence number. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by the index-name (see "Indexing", page 94).
12. Except for the OCCURS clause itself, all data description clauses associated with the item whose description contains that OCCURS clause apply to each occurrence of the item described.
13. When a computational elementary item (i.e. an item whose USAGE is BINARY, COMPUTATIONAL, COMPUTATIONAL-5, COMPUTATIONAL-1, COMPUTATIONAL-2) is an element in a table and is defined with the SYNCHRONIZED clause, the compiler adds any necessary slack bytes for each occurrence of the item (see "SYNCHRONIZED clause", page 189).

General rules for format 1

1. The subject of an OCCURS clause must be indexed whenever it is referenced in a statement other than a SEARCH statement.

   If the subject is the name of a group item, then all data-names belonging to the group must be indexed whenever they are used as operands.

   An indexed data-name references one particular table element. When used in a SEARCH statement, the data-name refers to the entire table.

2. The ASCENDING/DESCENDING phrase in conjunction with the INDEXED BY phrase is used in the execution of a SEARCH ALL statement (see "SEARCH statement", page 325).

3. Before an index-name may be used as an index, it must be initialized (using a SET, SEARCH ALL or PERFORM statement with VARYING phrase).

Format 2

```
OCCURS [integer-1 TO] integer-2 TIMES DEPENDING ON data-name-1

   [{ASCENDING }                                          ]
   [{          } KEY IS data-name-2 [data-name-3]...]...
   [{DESCENDING}                                          ]

   [INDEXED BY {index-1}...]
```

Syntax rules for format 2

1. integer-1 must be a positive integer or 0.
2. integer-2 must be greater than 0.
3. When used together, integer-1 must be less than integer-2.
4. data-name-1 must be defined as a positive integer numeric data item.
5. data-name-1 may be qualified (see "Qualification", page 77).
6. data-name-1 must not be indexed, i.e. it must not be a table element or an item within a table element.
7. If data-name-1 appears in the same record as the table whose occurrences it controls, it must appear before the variable portion of that record. In other words, the data item defined by data-name-1 must precede the record portion described by the OCCURS clause with the DEPENDING ON phrase.
8. The current value of data-name-1 at object time must not exceed that of integer-2, which specifies the maximum number of occurrences.
9. When used in the DEPENDING ON phrase of the OCCURS clause for a data area of variable length, data-name-1 must be supplied with a value before this area is used in a MOVE operation as a sending or receiving item. The same data-name-1 may be used by the sending and receiving items (see example 3-19).
10. data-name-2 may either be the subject of the OCCURS clause (in which case data-name-3,... must not be specified) or it must be subordinate to the group item referenced by the OCCURS clause.
11. data-name-3 must be subordinate to the group item referenced by the OCCURS clause.
12. If the OCCURS clause is specified in a data description entry included in a record description entry containing the EXTERNAL clause, data-name-1, if specified, must reference a data item possessing the external attribute which is described in the same Data Division.

13. If data-name-2 does not match the subject of the OCCURS clause the following rules apply:
    a) data-name-2, data-name-3,... must be subordinate to the group item referenced by the OCCURS clause.
    b) data-name-2, data-name-3,... must not be described with an OCCURS clause. Furthermore, they must not be subordinate to an entry which contains an OCCURS clause other than the one discussed here.
    c) data-name-2, data-name-3,... may be qualified with OF or IN (see "Qualification", page 77).
14. data-name-2, data-name-3,... are subject to the following rules:
    a) Up to 12 key fields may be specified for a given table element.
    b) The sum of the lengths of all key fields associated with a table element must not exceed 256.
    c) The key fields may have the following data format:
    DISPLAY, BINARY, COMPUTATIONAL, COMPUTATIONAL-5, COMPUTATIONAL-3 or PACKED-DECIMAL.
15. index-1... must be unique words in the program; otherwise, the indices specified by the INDEXED BY phrase are not defined in the program. Up to 12 index-names may be specified.
16. The OCCURS clause must not be specified in a data description entry that:
    a) has a level number of 01, 66, 77 or 88, or
    b) describes an item whose size is variable (the size of an item is variable if the data description entry of any item subordinate to it contains an OCCURS clause with the DEPENDING phrase).
17. A data description entry containing an OCCURS clause with the DEPENDING ON phrase may only be followed, within that record description, by data description entries which are subordinate to it. The COBOL85 compiler also allows data description entries which are independent of data hierarchy (see syntax rule 22).
18. The DEPENDING ON phrase specifies that the data item described by the OCCURS clause has a variable number of occurrences. The number of occurrences is controlled at object time by the value of data-name-1.
19. integer-1 and integer-2 specify the minimum and maximum number of occurrences, respectively. The value of the data item referenced by data-name-1 must range between integer-1 and integer-2.
20. If the value of data-name-1 is reduced at object time, the contents of the data items with occurrence numbers greater than the new value of data-name-1 are undefined.

21. When reference is made to a group item to which an entry with an OCCURS DEPENDING ON clause is subordinate, the part of the table area used in the operation is determined as follows:

    a) If the data item referenced by data-name-1 is outside the group, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used.

    b) If the data item referenced by data-name-1 is included in the same group and the group data item is referenced as a sending item, only that part of the table area that is specified by the value of the data item referenced by data-name-1 at the start of the operation will be used in the operation. If the group is a receiving item, the maximum length of the group will be used.

22. If, within a record description entry, a data area follows data items with the DEPENDING ON phrase, but is not subordinate to those items, then its position depends on the current values of data-name-1 in the preceding DEPENDING ON phrases.

    If the value of data-name-1 is changed (i.e. change of table length), the position of these data areas are shifted accordingly. However, their original contents are **not** shifted (see example 3-20).

23. The ASCENDING/DESCENDING KEY phrase defines whether the elements in the table are to be arranged in ascending or descending order, according to the values contained in data-name-2, data-name-3, etc. The data-names must be listed in descending order of significance.

    The user is responsible for seeing that the table elements are sorted properly (this order is presupposed in the SEARCH ALL statement).

24. The INDEXED BY phrase indicates that indexing may be used to reference the data-name which is the subject of the OCCURS clause, or any entry subordinate to that data-name. The storage allocation and format for indices are automatically defined by the compiler.

25. Each index contains a binary value that represents a displacement from the beginning of the table, corresponding to an occurrence number. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by the index-name (see "Indexing", page 94).

26. Except for the OCCURS clause itself, all data description clauses associated with the item whose description contains that OCCURS clause apply to each occurrence of the item described.

27. When a computational elementary item (i.e. an item whose USAGE is BINARY, COMPUTATIONAL, COMPUTATIONAL-5, COMPUTATIONAL-1, COMPUTATIONAL-2) is an element in a table and is defined with the SYNCHRONIZED clause, the compiler adds any necessary slack bytes for each occurrence of the item (see "SYNCHRONIZED clause", page 189).

28. Any entry that contains an OCCURS clause with the DEPENDING ON phrase, or has a subordinate entry with an OCCURS... DEPENDING ON clause, must not contain a REDEFINES clause.
29. Records are variable-length when format 2 is specified in a record description entry and the associated file description entry contains the RECORD clause with VARYING phrase.

    If DEPENDING ON is not specified in the RECORD clause, the contents of the data item referenced by data-name-1 in the OCCURS clause must be set to the number of occurrences to be written before the execution of any RELEASE, REWRITE or WRITE statement.

General rules for format 2

1. The subject of an OCCURS clause must be indexed whenever it is referenced in a statement other than a SEARCH statement.

   If the subject is the name of a group item, then all data-names belonging to the group must be indexed whenever they are used as operands.

   An indexed data-name references one particular table element. When used in a SEARCH statement, the data-name refers to the entire table.
2. The ASCENDING/DESCENDING phrase in conjunction with the INDEXED BY phrase is used in the execution of a SEARCH ALL statement (see "SEARCH statement", page 325).
3. Before an index-name may be used as an index, it must be initialized (using a SET, SEARCH ALL or PERFORM statement with VARYING phrase).

Example 3-19

Supplying a value to data-name-1 in OCCURS DEPENDING ON, using a MOVE operation. Given the following data definition:

```
WORKING-STORAGE SECTION.
01  ITEM-A.
    02  COUNTER-A PIC 9.
    02  DATA-A.
        03  CHARACTER-A PIC X OCCURS 1 TO 9
            DEPENDING ON COUNTER-A.
01  ITEM-B.
    02  COUNTER-B PIC 9.
    02  DATA-B.
        03  CHARACTER-B PIC X OCCURS 1 TO 9
            DEPENDING ON COUNTER-B.
```

The following MOVE operations are to be performed:

Case a) Sending item longer than receiving item

```
MOVE 6 TO COUNTER-A,
MOVE 3 TO COUNTER-B,
MOVE ITEM-A TO ITEM-B.
```

Contents following MOVE operation:

```
A-FELD: 6ABCDEF
B-FELD: 6ABCDEF
```

MOVE DATA-A TO DATA-B would result in:

```
A-FELD: 6ABCDEF
B-FELD: 3ABC
```

Case b) Sending item shorter than receiving item (contents of both items as they were before case a)

```
MOVE 3 TO COUNTER-A,
MOVE 6 TO COUNTER-B,
MOVE ITEM-A TO ITEM-B.
```

Contents following MOVE operation:

```
A-FELD: 3ABC
B-FELD: 3ABC
```

MOVE DATA-A TO DATA-B would result in:

```
A-FELD: 3ABC
B-FELD: 6ABC␣␣␣
```

The MOVE operations proceed according to the rules for alphanumeric moves (see "MOVE statement", page 294).

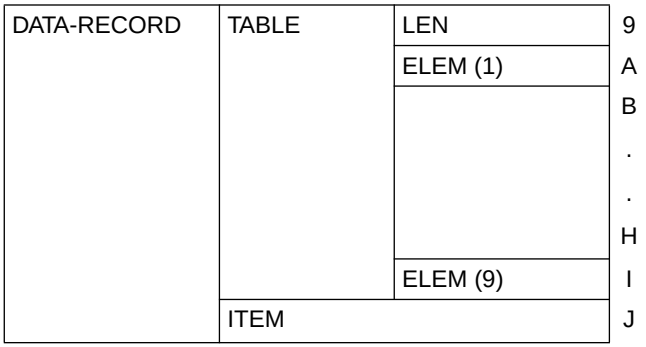
Example 3-20

```
OCCURS DEPENDING ON data-name-1
```

Given the following data definition:

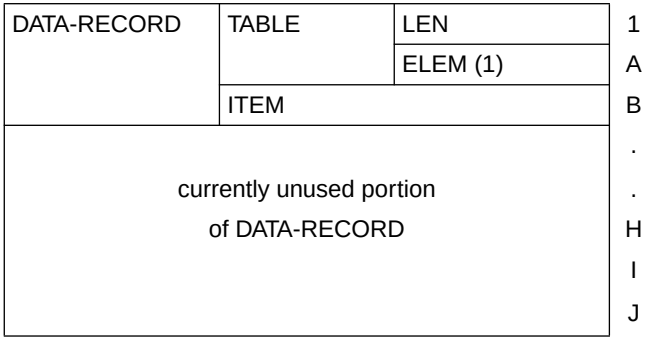
```
WORKING-STORAGE SECTION.
01  DATA-RECORD.
    02  TABLE1.
        03  LEN          PIC 9.
        03  TAB-ELEM     PIC X OCCURS 1 TO 9
                         DEPENDING ON LEN.

    02  ITEM       PIC X.
```

If the current value of LEN is 9, the following starting position of the items results:



After MOVE 1 TO LEN

the length of the table and hence the position of ITEM is changed:



The data item LEN now has the value 1; the data item ITEM has the value B.

PICTURE clause

Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary data item.

Format

```
{PICTURE}
{       } IS character-string
{PIC    }
```

Syntax rules

1. PIC is the abbreviation of PICTURE.
2. A character-string consists of certain allowable combinations of the characters in the COBOL character set. These combinations determine the category of data to which an elementary item belongs.
3. There are 18 characters or symbols that may be used in a character-string: A, comma (,), X, 9, P, Z, *, B, 0, +, minus (−), currency symbol ($), slash (/), S, V, period (.), credit (CR), and debit (DB). The functions of each character and symbol are described below under "Summary of characters and symbols in the PICTURE character-string".
4. The characters S, V, ., CR, and DB may appear only once in a PICTURE clause.
5. An integer enclosed in parentheses can follow the symbols A, comma (,), X, 9, P, Z, *, $, B, slash (/), 0, minus (−), and plus (+), to indicate the number of consecutive occurrences of the symbol. The number in parentheses must be at least 1 and must not exceed 131071.
6. At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, −, or CS [1)] must be present in a character-string.
7. The maximum number of characters allowed in a character-string is 30. This does not limit the number of characters in the represented area, which may be much more than 30.
8. The allowable combinations of symbols used in a character-string are shown in Table 3-3.
   An X at an intersection means that, in a character-string, the "second symbol" specified in the header line of the associated column may be located at any position to the **right** of the "first symbol" located at the start of the row. The leftmost column and uppermost

row for each symbol represent its use to the left of the decimal point position (l). The rightmost column and lowermost row for each symbol represent its use to the right of the decimal point position (r).

First symbol			Second symbol: B	0	/	,	.	+ − l	+ − r	CR DB	WZ[1)]	Z * l	Z * r	+ − l	+ − r	WZ[1)] l	WZ[1)] r	9	A X	S	V	P l	P r
			Non-floating insertion symbols									Floating insertion symbols							Other symbols				
Non-float. insertion symbols	B		X	X	X	X	X		X	X		X	X	X	X	X	X	X	X		X	X	
	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	X	X		X	X		X	X	X	X	X	X	X			X	X	
	.		X	X	X	X			X	X			X		X		X	X					
	+ or −	l	X	X	X	X	X				X	X	X			X	X	X			X	X	X
	+ or −	r																					
	CR / DB																						
	WZ[1)]		X	X	X	X	X		X	X		X	X	X	X			X			X	X	X
Float. insertion symbols	Z or *	l	X	X	X	X	X		X	X		X	X					X			X	X	
	Z or *	r	X	X	X	X			X	X			X										
	+ or −	l	X	X	X	X	X							X	X			X			X	X	
	+ or −	r	X	X	X	X									X								
	WZ[1)]		l	X	X	X	X	X		X	X						X	X	X			X	X
	WZ[1)]		r	X	X	X	X			X	X							X					
Other symbols	9		X	X	X	X	X		X	X								X	X		X	X	
	A or X		X	X	X													X	X				
	S																	X			X	X	X
	V		X	X	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	X					X

Table 3-3 Precedence of symbols used in the PICTURE clause

1) CS is the abbreviation for the currency symbol.

9. The number of characters specified in the character-string is used to determine the size of the item. However, the actual internal storage requirements are determined by the combination of the PICTURE and USAGE clauses (see "USAGE clause", page 192).
10. Five data categories may be described with the PICTURE clause:

    alphabetic
    alphanumeric
    numeric
    alphanumeric edited
    numeric edited
11. There are two general methods of performing editing in the PICTURE clause: by insertion, or by suppression and replacement.

    Four types of insertion editing are available:

    simple insertion
    special insertion
    fixed insertion
    floating insertion

    Two types of suppression and replacement editing are available:

    zero suppression and replacement with spaces
    zero suppression and replacement with asterisks (*).

General rules

1. The PICTURE clause is permitted only for elementary items.
2. The PICTURE clause must be specified for all elementary items, except for index data items and internal floating-point data items.

Summary of characters and symbols in a PICTURE character-string

The characters and symbols that are permitted in a character-string have the following meaning:

A Each A in the character-string represents a character position that may contain only a letter or a space.

B Each B in the character-string represents a character position into which a space character will be inserted. Each space is counted in the size of the item.

P P is the scaling position character. It represents a numeric digit position; however, storage space is never reserved for it, and it is always treated as if it contained a zero. P (or a group of Ps) indicates the location of an assumed decimal point (to the left of the Ps if Ps are the leftmost characters of the character-string, and to the right of the Ps if the Ps are the rightmost characters of the character-string). The maximum number of digit positions in the scaling position character for numeric and numeric-edited items is 18.

The character V (see below) may be specified or omitted. When specified, it must be inserted in the position of the assumed decimal point, to the left or to the right of the P or Ps specified. The scaling character P is not counted in the size of the data item. But the scaling characters are counted in determining the maximum number of digit positions (18) in numeric-edited or numeric data items. The scaling character P and the insertion character . (period) must not be specified at the same time in a character-string.

S The character S indicates the presence but not the location or mode of representation of an operational sign. If used, it must be the leftmost character of the character-string. The S is not counted in the size of the item unless the entry is subject to a SIGN clause which specifies the SEPARATE CHARACTER phrase.

V The character V indicates the position of an assumed decimal point. Since a numeric item cannot contain a printed decimal point, an assumed decimal point simply provides the compiler with information about the scaling alignment of items involved in computations. Storage is never reserved for the character V; therefore, V is not counted in the size of the item. If the assumed decimal point is the rightmost character in the character-string, the character V need not be supplied.

X Each X in the character-string represents a character position which may contain any allowable character from the EBCDIC set.

Z Each Z in the character-string represents a leading numeric character position. If such a character position contains a zero, then the zero is replaced by a space. Each Z is counted in the size of the item.

9 Each 9 in the character-string represents a character position that contains a numeral and is counted in the size of the data item.

0 Each zero in the character-string represents a character position into which the numeral zero will be inserted. Each zero is counted in the size of the item.

/ Each slash (/) in the character-string represents a character position into which a slash will be inserted. Each slash is counted in the size of the item.

, Each comma (,) in the character-string represents a character position into which a comma is inserted. Each comma is counted in the size of the data item.

. A period (.) in the character-string is an editing symbol and represents the decimal point used for alignment of the data item. Additionally, it represents a character position into which a period is inserted. The period is counted in the size of the data item.

Note

In a given program, the functions of the period and the comma are exchanged if the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph of the Environment Division. The rules for the period then apply to the comma, and vice versa, whenever they appear in a PICTURE character-string.

\+
\−
CR
DB These symbols are used as editing sign control symbols. When used, each represents the character position into which the editing sign control will be placed. These symbols are mutually exclusive in any one character-string, and each character used in the symbol is counted in determining the size of the data item.
Editing sign control symbols produce different results for positive and negative data items, depending on the value (see "Rules for fixed insertion editing", page 174).

* This symbol is a check protection symbol. Each asterisk (*) in the character-string represents a leading numeric character position into which an asterisk will be placed if that position contains a zero. Each asterisk (*) is counted in the size of the item.
If the asterisk is used together with the BLANK WHEN ZERO clause in a data description entry, the print editing routine cancels the effect of the BLANK WHEN ZERO clause since it suppresses any zeros.

$ The currency symbol ($) in the character-string represents a character position into which a currency sign is to be placed. The currency symbol in a character-string is represented either by the symbol $ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Alphabetic data items

Syntax rules

1. Only the symbol A is allowed in the character-string of an alphabetic item.
2. Any combination of the 52 letters of the alphabet and the space may be specified as the contents of an alphabetic data item.

Example 3-21

Picture	Value
`PICTURE AAA`	`NEW`

Alphanumeric data items

Syntax rules

1. The character-string for an alphanumeric data item is restricted to certain combinations of the following symbols: A, X, 9.

   An alphanumeric item is treated as if its character-string contained all Xs, with each X representing one character position.
2. A character-string that contains all As or 9s does not define an alphanumeric item.

Example 3-22

The alphanumeric value AB1234 could be represented by any of the following character-strings:

```
PICTURE XXXXXX
PICTURE AAXXXX
PICTURE AA9999
PICTURE A(2)X(4)
```

Numeric data items

There are two types of numeric data items: fixed-point items and floating-point items.

Fixed-point items

There are three types of fixed-point items: external decimal, binary, and internal decimal (see "USAGE clause", page 192).

Syntax rules

1. The character-string for a fixed-point data item may contain any permissible combination of the following symbols: 9, V, P, S.
2. It can contain from one up to and including 18 digit positions.
3. If the symbol S has not been specified, a data item may contain a combination of the digits 0 through 9.
4. If the symbol S has been specified, the data item may contain, in addition to the above digits, a "+", `"-"` or other representation of an arithmetic sign (see "SIGN clause", page 185).

Example 3-23

Valid combinations for fixed-point items:

```
PICTURE 9999
PICTURE S99
PICTURE S99V9
PICTURE PPP999
PICTURE S999PPP
```

Example 3-24

Let 8735 be the contents of a data item.

For PICTURE P(4)9(4), the arithmetic value of this item is .00008735.

For PICTURE 9(4)P(2), the arithmetic value of this item is 873500.

The arithmetic value is used for all operations except DISPLAY.

Floating-point data items

There are two types of floating-point data items: internal and external floating-point items (see "USAGE clause", page 192).

Syntax rules

1. The character-string for an external floating-point item has the following format:

   where the following rules are to be observed for the elements of the character-string:

   A positive or negative sign must be written immediately in front of the mantissa and the exponent in the character-string.

   + indicates that a plus sign is to represent positive values and a minus sign is to represent negative values.

   – indicates that a blank is to represent positive values and a minus sign is to represent negative values.

   mantissa

   The mantissa is the decimal part of the number after the decimal point; it consists of 1 to 18 ’9’s (each 9 representing a numeric character) and a leading, embedded, or trailing decimal point (.) or V. The decimal point indicates an actual (printed) decimal point, and the V indicates an assumed decimal point; these two characters are mutually exclusive.

   E

   immediately follows the mantissa and indicates that an exponent follows.

   exponent

   The exponent immediately follows the second sign and consists of two consecutive 9s.

2. The PICTURE clause **must not** be specified for an internal floating-point item.

Example 3-25 External floating-point item:

```
PICTURE          -9V99E-99
PICTURE          +9999.99E+99
PICTURE          -9(16)VE+99
PICTURE          +9(16).E-99
```

Alphanumeric edited data items

Syntax rules

1. The character-string for an alphanumeric edited data item is restricted to certain combinations of the following symbols: A, X, 9, 0, B, / (slash).
2. An alphanumeric edited character-string must contain at least one A or X, and at least one B or 0 or / (slash).
3. Characters from the computer's character set are allowed provided that the contents are represented in standard data format.
4. Only one type of editing is performed on alphanumeric edited data items: simple insertion editing using the characters zero (0), slash (/) and space (B) (see rules for simple insertion editing, page 173).

Example 3-26

Picture	Value
`PICTURE BAAAB`	`␣NEW␣`

Numeric edited data items

Syntax rules

1. The character-string for a numeric edited data item is restricted to certain combinations of the following symbols:
   B, / (slash), P, V, Z, 0, 9, , (comma), . (period), *, +, −, CR, DB, $.
2. The character-string must contain at least one of the symbols:
   0, B, / (slash), Z, *, +, , (comma), . (period), −, CR, DB or $.
3. The maximum number of digit positions in the character-string is 18.
4. The maximum length of a numeric edited data item is 127 characters.
5. Allowable combinations of these characters are governed by the editing rules and the symbol precedence rules (see following syntax rules and Table 3-3, page 166).

Simple insertion editing

Syntax rules

1. In simple insertion editing, the following insertion characters are used: , (comma), B (space), 0 (zero), and / (slash).
2. The insertion characters are counted in the size of the item, and represent the positions within the item into which they will be inserted.

Example 3-27

Category of data	PICTURE string receiving item	Data being moved	Edited result
Numeric edited	999,999 99B99B99 99B99B00 99/99/99	54321 654321 654321 654321	054,321 65␣43␣21 43␣21␣00 65/43/21
Alphanumeric edited	XXBXXX 000X(5) XX/XX	123AA A5CD3 CD05	12␣3AA 000A5CD3 CD/05

Special insertion editing

Syntax rules

1. Special insertion editing is performed by using the period (.) as an insertion character.

   In addition to being used as the insertion character, the period is also used as the decimal point for alignment purposes.
2. The assumed decimal point (represented by the character V) and the actual (printed) decimal point cannot be used in the same character-string.
3. Special insertion editing may be used on numeric edited data items only.
4. As a result of special insertion editing, the insertion character (decimal point) appears in the item in the same position as shown in the character-string; thus, the insertion character is the actual decimal point. The actual decimal point is counted in the size of the item.

Example 3-28

PICTURE string of receiving item	Data being moved *)	Edited result
999.99	123&4	123.40
999.99	12&34	012.34
999.99	1&234	001.23
999.99	&1234	000.12

*) & designates the position of the assumed decimal point, which does not appear in the MOVE operation.

Fixed insertion editing

Syntax rules

1. The editing symbols used for fixed insertion editing are: + (plus), − (minus), CR (credit), DB (debit), and $ (currency symbol).
2. Only one currency symbol and only one of the editing sign control symbols (+, −, CR, DB) may be used in a given character-string.
3. The currency symbol may be preceded only by a plus or a minus symbol; otherwise, it must be the leftmost character.
4. The symbols CR or DB, when specified, must be either the leftmost or the rightmost character.
5. The plus or minus symbol, when specified, must be either the leftmost or the rightmost character.
6. Fixed insertion editing results in the editing character occupying the same character position in the edited item as in the character-string.
7. The symbols CR or DB, when used, represent two character positions which are counted in the size of the data item. All fixed insertion editing characters are counted in the size of the data item.
8. Editing sign control symbols produce the results listed in Table 3-4, depending on the value of the data item.

Editing symbol in PICTURE character-string	Data item positive or zero	Data item negative
+	+	−
−	space	−
CR	2 spaces	CR
DB	2 spaces	DB

Table 3-4: Editing sign control symbols and their results

Example 3-29

PICTURE string of receiving item	Data being moved *)	Edited result
+999.99	+123&45	+123.45
+999.99	−123&45	−123.45
−999.99	+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

*) & designates the position of the assumed decimal point, which does not appear in the MOVE operation.

Floating insertion editing

Syntax rules

1. In floating insertion editing, the currency symbol ($) and the editing sign control symbols (+ and −) are used as insertion characters. These characters are mutually exclusive as floating insertion characters in the same character-string.

   Floating insertion editing is indicated in a character-string by the use of a sequence of at least two of the allowable insertion characters to represent the leftmost numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters (, B 0 /) embedded in the sequence of floating insertion characters or to the immediate right of this sequence are part of this floating string.

2. Only two types of floating insertion editing may be specified in a character-string.:

   – Some or all of the leading numeric character positions to the left of the decimal point may be represented by insertion characters.

   – All of the numeric character positions of the character-string may be represented by insertion characters.

3. The result of floating insertion editing depends on the representation in the character-string:
   - If the insertion characters are specified only to the left of the decimal point, a single insertion character is placed into the character position which immediately precedes the decimal point, or the first non-zero digit to the left of the character-string, and which is located inside the data represented by the insertion symbol string. The character positions preceding the insertion character are replaced with spaces.
   - If each of the numeric character positions in the character-string is represented by the insertion character, the result depends on the value of the data item concerned. If the value is zero, the entire data will contain spaces. If the value of the item is not zero, the result is the same as that occurring when the insertion characters are specified only to the left of the decimal point.
4. Every floating insertion character is counted in the size of the data item.
5. To avoid truncation of character positions, the programmer must form the character-string for the receiving item according to the following rule:
   - The minimum size of the character-string must equal the number of nonfloating insertion characters which are used for editing in the receiving item, plus one floating insertion character.

Example 3-30

Receiving area PICTURE	Data being moved*)	Edited result
$$$$.99	123&12	$123.12
$$$$.99	3&12	$3.12
$$$$.99	&12	$.12
$,$$$.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	1234&56	$1,234.56
+,+++,999.99	−123456&78	−123,456.78
+,+++,+++.++	000&00	(Leerzeichen)

*) & designates the position of the assumed decimal point which does not appear in the MOVE operation.

Zero suppression and replacement editing

Syntax rules

1. Suppression of leading zeroes in numeric character positions is indicated by the use of the alphabetic character Z or the character * (asterisk) as suppression symbols in a PICTURE character-string. These characters are mutually exclusive in the same character-string. If Z is used, the replacement character is a space. If * is used, the replacement character is a space. If * is used, the replacement character is * (asterisk).
2. Zero suppression and replacement editing in a character-string is achieved by using a string of one or more of the permissible symbols (* or Z) to represent leading numeric character positions which are to be filled with the replacement characters when the associated character positions in the data contain zeros. Any of the simple insertion characters (, B 0 /) embedded in the string of symbols or to the immediate right of the string are part of the string; each of these simple insertion characters works like an * or a Z, until a non-zero character is encountered. The simple insertion characters (, B 0 /) and fixed insertion characters ($ + −) to the left of the suppression string are not subject to the rules for zero suppression and replacement.
3. Two types of zero suppression editing may be used in a character-string.
   – Some or all of the leading numeric character positions to the left of the decimal point may be represented by suppression symbols.
   – All of the numeric character positions in the character-string may be represented by suppression symbols.
4. The result of zero suppression and replacement editing depends on the representation in the character-string:
   – If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a suppression symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point.
   – If all numeric character positions in the character-string are represented by suppression symbols and the value of the data item is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is Z, the entire data item is replaced by spaces. If the value is zero and the suppression symbol is *, all characters in the data item, except for the decimal point, are replaced by asterisks; in this case, zero suppression editing overrides the BLANK WHEN ZERO clause, if the latter is specified.
5. Each suppression character is included in the size of the data item.

Example 3-31

Receiving area PICTURE	Data being moved*)	Editing result
ZZZZ.ZZ	0000&00	(spaces)
****.**	0000&00	****.**
ZZZZ.99	0000&00	.00
****.99	0000&00	****.00
ZZZZ.ZZ	+135&00	135.00
$**,***.**BDB	-2135&00	$*2,135.00␣DB
$BB****,**.99BBCR	-2135&00	$␣␣**21,35.00␣␣CR

*) & designates the position of the assumed decimal point which does not appear in the MOVE operation.

REDEFINES clause

Function

The REDEFINES clause allows the programmer to define different data description entries for the same area of computer storage.

Format

```
level-number [data-name-1 ] REDEFINES data-name-2
             [FILLER      ]
```

(The level number, data-name-1 and FILLER are not part of the REDEFINES clause, and are shown here only for clarity.)

Syntax rules

1. The REDEFINES clause, when used, must immediately follow data-name-1.
2. The level numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88.
3. The length of the data item of data-name-1 must be less than or equal to the length of the data item of data-name-2, if the associated level-number is not equal to 01. There is no such restriction in effect at level 01.
4. Data-name-2 may be qualified but not indexed or subscripted.
5. The data description entry for data-name-2 must not contain an OCCURS clause; however, data-name-2 may be subordinate to a data item which contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause must not be indexed or subscripted. A data item that is subordinate to data-name-2 may contain an OCCURS clause without the DEPENDING ON phrase (see "OCCURS clause").
6. Data-name-1 or any data item subordinate to data-name-1 may contain an OCCURS clause without the DEPENDING ON phrase. If data-name-1 contains an OCCURS clause, the size of data-name-1 is calculated by multiplying the length of one table element by the number of occurrences of the table element.
7. The REDEFINES clause must not appear in 01-level entries in the FILE SECTION (implicit redefinition is provided there automatically at 01-level).
8. Multiple redefinitions of the storage area are permitted but must all refer to the data-name supplied in the original definition.

9. Except for condition-name entries, the entries giving a new description of a storage area must not contain a VALUE clause.
10. No entries having level numbers numerically lower than that of data-name-1 and data-name-2 may occur between the descriptions of data-name-2 and data-name-1.
11. The REDEFINES clause may be specified for an item subordinate to a redefined item, or for a data item which is subordinate to an item containing a REDEFINES clause.

General rules

1. Data-name-1 is the name of the data area associated with the redefinition. Data-name-2 is the name of the original definition of the data area to be redefined.
2. Redefinition starts at data-name-2 and ends when a level number less than or equal to that of data-name-2 is encountered.
3. When an area is redefined, all descriptions of that area remain in effect. For example, if A and B are two separate data items sharing the same storage area, the procedure statements MOVE ALPHA TO A or MOVE BETA TO B could be executed at any point in the program. In the first case, ALPHA would be moved to A and would take the form specified by the description of A. In the second case, BETA would be moved to the same physical area and would take the form specified by the description of B. If both MOVE statements were executed successively in the order specified, the value BETA would overlay the value ALPHA; however, redefinition of an area does not erase any data and does not supersede a previous description.
4. Moving a data item from A to B when B is a redefinition of A amounts to moving an item to itself, and the result of such a move is unpredictable. The same is true of the opposite type of move; that is, moving A to B when A redefines B.
5. The use of data items defined by the PICTURE and USAGE clauses within an area can be redefined. Altering the use of an area by the REDEFINES clause does not, however, change any existing data.
6. When the SYNCHRONIZED clause is specified in a data entry that is redefining a previous data entry, the user should ensure that the area being redefined begins on the proper boundary: halfword, fullword, or doubleword.

Example 3-32

```
02  ALPHA.
    03  A-1 PICTURE X(3).
    03  A-2 PICTURE X(2).
02  BETA REDEFINES ALPHA PICTURE 9(5).
02  GAMMA.
```

BETA is data-name-1; ALPHA is data-name-2. BETA redefines the area assigned to ALPHA (that is, the area occupied by A-1 and A-2). Redefinition starts at BETA and ends at the next level number 02 (the number preceding GAMMA).

Example 3-33

(Multiple redefinitions)

```
02 ALPHA PICTURE 9(3).
02 BETA REDEFINES ALPHA PICTURE X(3).
02 GAMMA REDEFINES ALPHA PICTURE A(3).
```

Example 3-34

```
01  SAMPLE-AREA-1.
    02  FIRST-DEFINITION  PICTURE 99 VALUE 12.
    02  SECOND-DEFINITION REDEFINES FIRST-DEFINITION
        USAGE COMPUTATIONAL  PICTURE S9(4).
```

In this example, FIRST-DEFINITION is a 2-byte unsigned external decimal number with the value 12. This means that the contents of the two bytes in hexadecimal is X'F1F2'. SECOND-DEFINITION is also a number, and occupies the same two bytes; but it does not have the value 12. The data in these two bytes (X'F1F2') is unchanged by the redefinition; and, since SECOND-DEFINITION is a signed, binary number, this data has the value −3598.

Example 3-35

```
01  SAMPLE-AREA-2.
    02  FIRST-DEFINITION.
        03  ALPHA                   PICTURE X(3).
        03  BETA                    PICTURE X(5).
        03  GAMMA REDEFINES BETA    PICTURE 9(5).
        03  FILLER                  PICTURE X(10).
    02  SECOND-DEFINITION REDEFINES FIRST-DEFINITION PICTURE X(18).
```

In this example, one of the items subordinate to FIRST-DEFINITION is redefined: GAMMA REDEFINES BETA. This is permitted, and is not blocked by the fact that FIRST-DEFINITION is itself later redefined by SECOND-DEFINITION.

Example 3-36

```
01  SAMPLE-AREA-3.
    02  FIRST-DEFINITION PICTURE S9(7).
    02  SECOND-DEFINITION REDEFINES FIRST-DEFINITION.
        03  A-1 PICTURE A.
        03  N-1 REDEFINES A-1 PICTURE 9.
        03  FILLER PICTURE X(6).
```

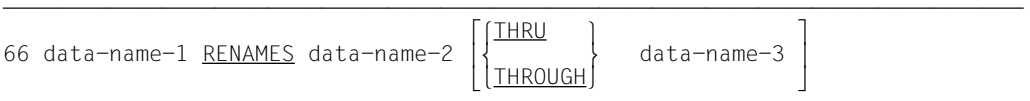
In this example, one of the data items subordinate to SECOND-DEFINITION is redefined; N-1 REDEFINES A-1. This is permitted, and is not blocked by the fact that SECOND-DEFINITION itself is a redefinition.

RENAMES clause

Function

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary data items. This clause assigns a new name to an item or items established by a record description. Unlike REDEFINES, the RENAMES clause does not redefine existing data descriptions but merely allows data to be accessed and/or grouped under alternative names while maintaining the previously defined data descriptions.

Format



(The level number 66 and data-name-1 are not part of the RENAMES clause, and are shown only to improve clarity.)

Syntax rules

1. All entries of the RENAMES clause which refer to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.
2. data-name-2 must precede data-name-3 in the record description. After each redefinition, the beginning point of the area described by data-name-3 must logically follow the beginning point of the area defined by data-name-2.
3. data-name-2 and data-name-3 must be the names of elementary items or groups of elementary data items in the associated logical record, and cannot be the same data-name.
4. The beginning of the area defined by data-name-3 must not lie to the left of the beginning of the area defined by data-name-2. The end of the area defined by data-name-3 **must** lie to the right of the end of the area defined by data-name-2. Hence, data-name-3 cannot be subordinate to data-name-2.
5. None of the data items within the area of data-name-2 and data-name-3, when specified, may have a variable size as described in the OCCURS clause (see "OCCURS clause" with DEPENDING ON phrase).
6. data-name-1 cannot be used as a qualifier and can be qualified only by the names of the associated 01-level, SD, or FD entries.
7. data-name-2 and data-name-3 may be qualified.

8. Neither data-name-2 nor data-name-3 may contain an OCCURS clause in its data description entry, nor may it be subordinate to a data item which contains an OCCURS clause in its data description entry.
9. The RENAMES clause may neither refer to another 66-level entry nor to a 77-level, 88-level, or 01-level entry.
10. data-name-1 specifies an alternative definition for one or more data items.

    data-name-2 or data-name-3 specifies the data item(s) to be renamed.
11. When data-name-3 is specified, data-name-1 is a group item that includes all elementary items:
    – starting with data-name-2 (if this is an elementary data item); or starting with the first elementary item within data-name-2 (if this is a group item).
    – concluding with data-name-3 (if this is an elementary data item); or concluding with the last elementary item within data-name-3 (if this is a group item).
12. If data-name-3 is not specified, then data-name-2 may be either a group item or an elementary item. If data-name-2 is a group item, data-name-1 is treated as a group item; if data-name-2 is an elementary item, data-name-1 is treated as an elementary data item.

General rule

More than one RENAMES clause may be written for the same logical record.

Example 3-37

The following example shows how a RENAMES clause may be used in an actual program:

```
01  INPUT-RECORD.
    02  ARTICLE-1.
        03  ARTICLE-NO              PIC 99.
        03  PRICE                   PIC 9999.
    02  ARTICLE-2.
        03  ARTICLE-NO              PIC 99.
        03  PRICE                   PIC 9999.
    02  ARTICLE-3.
        03  ARTICLE-NO              PIC 99.
        03  PRICE                   PIC 9999.
    66  ART-ONE RENAMES ARTICLE-1.
    66  ART-TWO RENAMES ARTICLE-1 THRU ARTICLE-2.
    66  ART-THREE RENAMES ARTICLE-1 THRU ARTICLE-3.
```

In this case, each reference to ART-ONE would access group item ARTICLE-1; each reference to ART-TWO, the group items ARTICLE-1 and ARTICLE-2; each reference to ART-THREE, the group items ARTICLE-1, ARTICLE-2 and ARTICLE-3.

SIGN clause

Function

The SIGN clause specifies the position and the mode of representation of the operational sign for numeric data items.

Format

```
[SIGN IS] {LEADING  } [SEPARATE CHARACTER]
          {TRAILING }
```

Syntax rules

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S, or a group item containing at least one such numeric data description entry.
2. The numeric data description entries to which the SIGN clause applies must be described, explicitly or implicitly, as USAGE IS DISPLAY.
3. If a SIGN clause is specified for either a group item or an elementary numeric item subordinate to a group item for which a SIGN clause is also specified, the SIGN clause of the subordinate group or numeric data item takes precedence for that item.
4. If the CODE-SET clause is specified, any signed numeric data description entries must be described with the SIGN IS SEPARATE clause.
5. The SIGN clause specifies the position and the mode of representation of the operational sign. If entered for a group item, it applies to each numeric data description entry subordinate to that group. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character S; the S indicates the presence, but not the mode of representation, of the operational sign.
6. A numeric data description entry whose PICTURE contains the character S, but to which no SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character S. (For representation of the operational sign see "USAGE clause", page 192.)

General rules

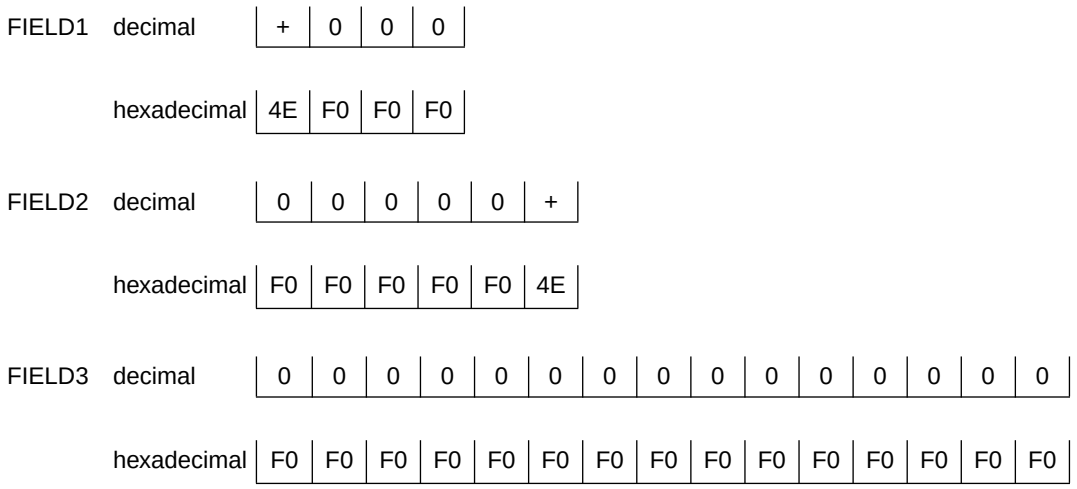
1. If the SEPARATE CHARACTER phrase is not present, then:
   a) The letter S in a PICTURE character-string is not counted in determining the size of the item.
   b) The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item. The TRAILING phrase is taken as this compiler's default value.
   c) For the compiler, the operational sign is the half-byte C for positive and the half-byte D for negative.
2. If the SEPARATE CHARACTER phrase is present, then:
   a) The letter S in a PICTURE character-string is counted in determining the size of the item.
   b) The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.
   c) The operational signs for positive and negative are the standard data format characters + and −, respectively.
3. Every numeric data description entry whose PICTURE character-string contains the character S is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

Example 3-38

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SIGNEXPL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FIELD1          PIC S999   SIGN IS LEADING SEPARATE.
01  GROUP1          USAGE IS DISPLAY.
    02  FIELD2      PIC S9(5)  SIGN IS TRAILING SEPARATE.
    02  FIELD3      PIC X(15).
    02  FIELD4      PIC S99    SIGN IS LEADING.
01  FIELD5          PIC S9(9)  SIGN IS TRAILING.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    MOVE ZEROES TO FIELD1,FIELD2,FIELD3,FIELD4,FIELD5.
    MOVE 3 TO FIELD4.
    MOVE -2 TO FIELD5.
    MOVE FIELD4 TO FIELD2.
    MOVE FIELD2 TO FIELD3.
    MOVE FIELD5 TO FIELD4.
    DISPLAY "Field1 = " FIELD1 UPON T.
    DISPLAY "Field2 = " FIELD2 UPON T.
    DISPLAY "Field3 = " FIELD3 UPON T.
    DISPLAY "Field4 = " FIELD4 UPON T.
    DISPLAY "Field5 = " FIELD5 UPON T.
    STOP RUN.
```

The contents of all fields after each MOVE statement are shown below.

After the first MOVE statement: :



FIELD4	decimal	0^{+}	0							
	hexadecimal	C0	F0							
FIELD5	decimal	0	0	0	0	0	0	0	0	0+
	hexadecimal	F0	F0	F0	F0	F0	F0	F0	F0	C0

After the second MOVE statement:

FIELD4	decimal	+	3
	hexadecimal	C0	F3

After the third MOVE statement:

FIELD5	decimal	0	0	0	0	0	0	0	0	2^{-}
	hexadecimal	F0	F0	F0	F0	F0	F0	F0	F0	D2

After the fourth MOVE statement:

FIELD2	decimal	0	0	0	0	3	+
	hexadecimal	F0	F0	F0	F0	F3	4E

After the fifth MOVE statement:

FIELD3	decimal	0	0	0	0	3										
	hexadecimal	F0	F0	F0	F0	F3	40	40	40	40	40	40	40	40	40	40

After the sixth MOVE statement:

FIELD4	decimal	0^{-}	2
	hexadecimal	D0	F2

SYNCHRONIZED clause

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on a natural boundary of the computer memory to ensure efficiency during the performance of arithmetic operations on the item.

As an extension to standard COBOL, the compiler described in this publication allows the SYNCHRONIZED clause to be specified at group level; this has the effect of aligning the elementary items subordinate to the group.

Format

```
{SYNCHRONIZED} [LEFT ]
{SYNC        } [RIGHT]
```

Syntax rules

1. SYNC is the abbreviation of SYNCHRONIZED.
2. LEFT and RIGHT are treated as comments.
3. When items are boundary-aligned because of the presence of a SYNCHRONIZED clause, it is sometimes necessary for the compiler to insert slack bytes. Slack bytes are unused character positions inserted into a record immediately ahead of an item requiring boundary alignment. Such slack bytes are included in the length of the group item containing the aligned item.
4. The actual boundary at which a synchronized item is placed depends on the phrase specified in the USAGE clause for the item.

   If the SYNCHRONIZED clause is specified, the following actions are carried out:

   For a data item with USAGE COMPUTATIONAL, COMPUTATIONAL-5 or BINARY:

   a) For the area S9 through S9(4) in the PICTURE clause, the data item is aligned on a halfword boundary (multiple of 2).

   b) For the area S9(5) through S9(18) in the PICTURE clause, the data item is aligned on a fullword boundary (multiple of 4).

   For a data item with USAGE IS COMPUTATIONAL-1, the item is aligned on a fullword boundary.

   For a data item with USAGE IS COMPUTATIONAL-2, the item is aligned on a doubleword boundary (multiple of 8).

For an index data item, the item is aligned on a fullword boundary.

For a data item with USAGE DISPLAY, or COMPUTATIONAL-3 or PACKED-DECIMAL, the SYNCHRONIZED clause is treated as a comment, as no alignment is necessary in these cases.

5. If the SYNCHRONIZED clause is omitted for binary data items or internal floating-point items, no slack bytes are generated. However, if arithmetic operations are performed on these items, the compiler will generate the statements necessary to move these items to auxiliary items which are properly aligned for the arithmetic operation.
6. If the SYNCHRONIZED clause is specified for a group item, then all elementary items with USAGE COMPUTATIONAL, COMPUTATIONAL-5, BINARY, COMPUTATIONAL-1, COMPUTATIONAL-2 or INDEX will be aligned as if the SYNCHRONIZED clause had been specified in the data description entries of these items.

General rules

1. Standard COBOL permits specification of the SYNCHRONIZED clause for elementary data items only. However, the compiler described in this manual allows the SYNCHRONIZED clause to be specified for group items also, with the effect described above.
2. When the SYNCHRONIZED clause is specified within a table (described by the OCCURS clause), each table element will be aligned. (This process is described under "Alignment by insertion of slack bytes".)
3. When specifying the SYNCHRONIZED clause in conjunction with a REDEFINES clause, the programmer must ensure that the element being redefined is aligned (see example 3-39).
4. The SYNCHRONIZED clause does not alter the length of an elementary data item. Each unused internal memory location (slack bytes) is included in the size of the group to which the elementary item is subordinate, and must be allowed for an internal memory allocation if the group item was the object of a REDEFINES clause (see example 3-40).
5. All record descriptions (01-level entries) in all sections of the Data Division begin at doubleword boundaries.
6. When blocking records that contain elementary items with the SYNCHRONIZED clause specified, the user must add the necessary slack bytes to ensure proper alignment after the first record within the block. (This process is described under "Alignment by insertion of slack bytes").

7. For the purpose of aligning elementary items with USAGE COMPUTATIONAL, COMPUTATIONAL-5, BINARY, COMPUTATIONAL-1, COMPUTATIONAL-2, specified in the LINKAGE section, all 01-level elementary items are assumed to be aligned on doubleword boundaries. Consequently, the user must ensure that these operands are appropriately aligned in the USING phrase when he writes a CALL statement.

Example 3-39

In the following example, A has to be aligned on a fullword boundary:

```
    02  A  PICTURE X(4).
    02  B  REDEFINES A PICTURE S9(9) USAGE BINARY SYNC.
```

Example 3-40

```
01  RECORD.
    02  A.
        03  G PICTURE X(5).
        03  H PICTURE S9(9) SYNC USAGE BINARY.
    02  B   REDEFINES A.
        03  I PICTURE X(12).
```

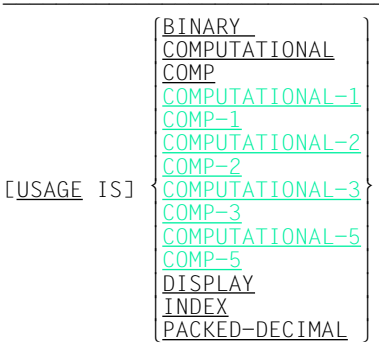
Here, elementary item G occupies 5 bytes, and elementary item H occupies 4 bytes. The SYNCHRONIZED and USAGE clauses indicate that the elementary data item H is aligned on fullword boundary; elementary item H is therefore preceded by 3 slack bytes. As data item A as a whole occupies 12 bytes, the subject of the REDEFINES clause (data item B) must also occupy 12 bytes.

USAGE clause

Function

The USAGE clause specifies the format in which an elementary item is represented in the computer's internal storage.

Format



Syntax rules

1. COMPis the abbreviation for COMPUTATIONAL.
   COMP-1 is the abbreviation for COMPUTATIONAL-1.
   COMP-2 is the abbreviation for COMPUTATIONAL-2.
   COMP-3 is the abbreviation for COMPUTATIONAL-3.
   COMP-5 is the abbreviation for COMPUTATIONAL-5.
2. If the USAGE clause is not specified for an elementary item, or for a group item, the usage is implicitly DISPLAY.
3. For a description of the various categories of data see chapter 2.

General rules

1. The USAGE clause may be written at any data description level. If it is specified at group level, it applies to each elementary data item of the group.
2. The USAGE of an elementary item must not conflict with the USAGE of the group item to which the elementary item belongs.
3. An elementary item described with USAGE BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, COMPUTATIONAL-5 or PACKED DECIMAL represents a value for use in arithmetic operations and must

therefore be numeric. If any of these phrases is specified for a group item, it refers only to the elementary items of that group; the group item itself must not be used in arithmetic operations.

4. The USAGE clause does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to.
5. The internal representation of the numeric data items is shown in Table 3-5.

DISPLAY phrase

Syntax rules

1. The type of elementary item for which the DISPLAY phrase is written is defined by the character-string in the PICTURE clause.
2. External decimal data items are described below under general rule 1, external floating-point data items under general rule 2. In addition, all data items are also described under "PICTURE clause" (page 164).
3. The DISPLAY phrase specifies that the data item is to be stored in standard data format; that is, in character form, with one character per 8-bit byte. Each character position of the data item is represented by one byte, as specified in the appropriate character-string of the PICTURE clause.

General rules

1. External decimal data items are internally represented as follows:
   Each digit of a number is represented by a single byte. The four high-order bits of each byte are the zone portion. The zone portion of the low-order or high-order byte (depending on the SIGN clause) represents the sign of the number, assuming that a sign exists. The four low-order bits contain the value of the digit.
   The maximum length of an external decimal item is 18 digits.
2. External floating-point items consist of a mantissa, which represents the decimal part of the number, and an exponent with the base 10.
   The value of an external floating-point item is calculated by multiplying the mantissa by 10 to the power of exponent.
   The magnitude of a number represented by a floating-point item must be greater than $5.4 * 10^{-79}$ and must not exceed $7.2 * 10^{75}$.

   An external floating-point item, when used as a numeric operand, is checked at object time and is converted into an internal floating-point item. It is used in this form in arithmetic operations (see notes on COMPUTATIONAL-1 and COMPUTATIONAL-2).

Example 3-41

Data formats for USAGE IS DISPLAY

Data category	Value	PICTURE description	Internal representation*)									
alphabetic	ABCD	AAAA.	C1	C2	C3	C4						
alphanumeric	A1B2	XXXX.	C1	F1	C2	F2						
alphanumeric edited	123AB	XXBXXX.	F1	F2	40	F3	C1	C2				
numeric edited	54321	99,999	F5	F4	6B	F3	F2	F1				
numeric												
external decimal	+1234	9999	F1	F2	F3	F4						
	+6879	S9999	F6	F8	F7	C9						
	−6879	S9999	F6	F8	F7	D9						
external floating-point	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

*) Each box represents one byte.

BINARY phrase or COMPUTATIONAL phrase or COMPUTATIONAL-5 phrase

Syntax rules

1. These phrases specify binary data items.
2. The PICTURE clause of a binary data item must contain no other characters but 9s, the operational sign S, the assumed decimal point V, and one or more Ps (see "PICTURE clause", page 164).
3. The data items are stored in a halfword, fullword, or doubleword, and are aligned only if the SYNCHRONIZED clause was specified.
4. If a data item described with USAGE IS BINARY is used as a receiving data item, a check is made to determine whether the value to be transferred to this data item exceeds the maximum possible value indicated by the PICTURE character-string. If this is the case, the value is made to conform by truncation.
   If a receiving data item is described with USAGE IS COMPUTATIONAL or COMPUTATIONAL-5, this check and any subsequent truncation which may be required is not performed.

General rules

1. The storage requirements for binary items vary depending on the number of decimal digits specified in the PICTURE clause, as follows:

Decimal digits in the PICTURE clause	Bytes required in computer storage	Alignment
1-4	2	halfword
5-9	4	fullword
10-18	8	fullword

2. The leftmost bit of a binary data item is the operational sign. The remaining bits represent the value.

For examples of the BINARY, COMPUTATIONAL or COMPUTATIONAL-5 phrases see Table 3-5 (page 200), "Internal representation of numeric data items".

COMPUTATIONAL-1 phrase

Syntax rules

1. This phrase specifies internal floating-point items, which are equivalent to external floating-point items in terms of capacity and application (see "Data categories", page 68).
2. For a COMPUTATIONAL-1 data item, the PICTURE clause is prohibited.
3. The COMPUTATIONAL-1 phrase indicates that a data item is stored in single-precision floating-point format.
4. A COMPUTATIONAL-1 data item has a length of 4 bytes and is aligned on a fullword boundary if the SYNCHRONIZED clause is specified.

General rules

1. A COMPUTATIONAL-1 data item is represented in storage as follows:

   Here, S is the sign of the mantissa.
   Characteristic = exponent + 32

2. An internal single-precision floating-point item permits representation with a precision of six decimal digits.
3. The following applies to the value that may be represented in a COMPUTATIONAL-1 data item:

   value = 0 or the absolute value of it may range from $5.4 * 10^{-79}$ to $7.2 * 10^{75}$.

For examples of the COMPUTATIONAL-1 phrase see Table 3-5, "Internal representation of numeric data items" (page 200).

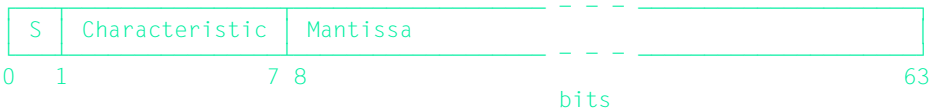
COMPUTATIONAL-2 phrase

Syntax rules

1. This phrase specifies internal floating-point items, which are equivalent to external floating-point items in terms of capacity and application (see "Data categories", page 68).
2. For a COMPUTATIONAL-2 item, the PICTURE clause is prohibited.
3. The COMPUTATIONAL-2 phrase indicates that a data item is to be stored in double-precision floating-point format.
4. A COMPUTATIONAL-2 item has a length of 8 bytes and is aligned on a doubleword boundary if the SYNCHRONIZED clause is specified.

General rules

1. A COMPUTATIONAL-2 data item is represented in storage as follows:



   Here, S is the sign of the mantissa.
   Characteristic = exponent + 64

2. A double-precision internal floating-point item permits representation with a precision of 15 decimal digits.
3. The following applies to the value that may be represented in a COMPUTATIONAL-2 data item:

   value = 0 or the absolute value of it may range from $5.4 * 10^{-79}$ to $7.2 * 10^{75}$.

For examples of the COMPUTATIONAL-2 phrase see Table 3-5, "Internal representation of numeric data items" (page 200).

COMPUTATIONAL-3 phrase or PACKED-DECIMAL phrase

Syntax rules

1. The COMPUTATIONAL-3 and PACKED-DECIMAL phrases are identical in meaning.
2. The phrases indicate that the data item is stored in internal decimal format (i.e. in packed form).
3. The PICTURE clause of a COMPUTATIONAL-3 or PACKED-DECIMAL item may contain no characters other than 9s, the operation sign S, the assumed decimal point V, and one or more Ps (see "PICTURE clause", page 164).

General rule

Internal decimal data items are represented by 2 digits per byte; the sign is contained in the four low-order bits of the low-value byte.

For internal decimal data items whose PICTURE clause contains no S, the representation of the absolute value corresponds to the number.

For examples of the COMPUTATIONAL-3 or PACKED-DECIMAL phrases see Table 3-5, "Internal representation of numeric data items" (page 200).

Format	**PICTURE clause**	**USAGE and SIGN phrase**	**Value in** external represen-tation	**Value in internal representation** [4]	**Bytes required**	**Conversion for arithmetic operation** Alignment	**Alignment if SYNC is specified**
External decimal (zoned)	9999	DISPLAY	1234	F1F2F3F4	1 byte/digit	Yes, in order to conform to format of other operands or COMP-3 or PACKED-DECIMAL	
	S9999		+1234	F1F2F3C4[1)2)]			
	S9999		−1234	F1F2F3D4[1)2)]			
	S9999	DISPLAY SIGN TRAILING	1234+	F1F2F3C4			
			1234−	F1F2F3D4			
	S9999	DISPLAY SIGN TRAILING SEPARATE	1234+	F1F2F3F44E	+ 1 byte for sign		
			1234−	F1F2F3F460			
	S9999	DISPLAY SIGN LEADING	+1234	C1F2F3F4			
			−1234	D1F2F3F4			
	S9999	DISPLAY SIGN LEADING SEPARATE	+1234	4EF1F2F3F4	+ 1 byte for sign		
			−1234	60F1F2F3F4			
Internal decimal (packed)	9999	COMP-3 or PACKED-DECIMAL	+1234	01234F[2)]	2 digits per byte, except for low-order byte which contains a digit and the sign	No, except when other operand is binary and conversion to binary would be more advan-tageous.	None
	9999		−1234	01234F[2)]			
	S9999		+1234	01234C[2)]			
	S9999		−1234	01234D[2)]			

Table 3-5 Internal representation of numeric data items

Format	PICTURE clause	USAGE and SIGN phrase	Value in external representation	Value in internal representation[4]	Bytes required	Conversion for arithmetic operation alignment	Alignment if SYNC is specified
Binary	S9999 S9999	BINARY or COMP or COMP-5	+1234 −1234	04D2 FB2E	2 bytes for 1-4 digits 4 bytes for 5-9 digits 8 bytes for 10-18 digits 2	No, except when used in mixed-form computations to maintain common formats, or if COMP-3 or PACKED-DECIMAL would be more advantageous.	 Halfword[3]
Ext. floating point	+99.99E-99	DISPLAY	+12.34E+2	4EF1F26BF3F4C540F0F2	1 byte per character	Yes. To internal floating-point	None
Int. floating point	None allowed	COMP-1	+12.34E+2	434D2000	4	No	Word
	None allowed	COMP-2	−12.34E-2	C01F972474538EF3	8	No	Doubleword

Table 3-5 Internal representation of numeric data items

1) One byte per digit, except for the low-order byte which contains the sign in the first halfbyte and the last digit in the second halfbyte.

2) Mode of sign representation:
F = non-printable plus sign (treated as an absolute value)
C = internal equivalent of plus sign
D = internal equivalent of minus sign.

3) See rules for binary data items.

4) Each box represents one byte.

INDEX phrase

Syntax rule

An elementary item described with USAGE IS INDEX is called an index data item. This is a data item (not necessarily associated with any table) which may be used to save values of index-names for future reference. An index data item is assigned the value of an index by the SET statement. The value of an index data item is not an occurrence number.

General rules

1. The USAGE clause with INDEX phrase may be written at any level. If a group item is described with USAGE IS INDEX, the elementary items in the group are all index data items; the group itself is not an index data item.
2. An index data item can be referenced directly only in a SEARCH statement, in a SET statement, in a relation condition, in the USING phrase of the Procedure Division header, or in the USING phrase of a CALL statement.
3. An index data item cannot be a conditional variable.
4. An index data item may be part of a group which is referenced in a MOVE statement or an input/output statement. When such statements are executed, however, the contents of the index data item are not converted.
5. SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO or VALUE clauses cannot be used to describe group items or elementary items described with USAGE IS INDEX.
   However, the compiler allows the SYNCHRONIZED clause to be used with the USAGE IS INDEX clause.

Example 3-42

```
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.
```

Here the index data item A-INDEX is set to the current value of the index-name A-NAME, i.e. the occurrence number (3) minus 1, multiplied by the length of the entry (9) = 18.

VALUE clause

Function

The VALUE clause defines the initial value of a data item in the WORKING-STORAGE SECTION, the value of a printable data item of the REPORT SECTION, or the value or range of values associated with a condition-name.

Format 1 of the VALUE clause is specified to define the initial value of a data item in the WORKING-STORAGE SECTION or the value of a printable data item in the REPORT SECTION.

Format 2 of the VALUE clause is specified to define the value or range of values associated with a condition-name. The optional addition "WHEN SET TO FALSE IS LITERAL-4" defines the value to which the related data item is set when the "SET condition-name TO FALSE" statement is executed.

Format 3 serves to initialize table elements.

In the FILE SECTION and in the LINKAGE SECTION the value clause may be used only in connection with level-number 88 (format 2 of the VALUE clause).

Format 1

```
[VALUE IS literal]
```

Syntax rules for format 1

1. The literal specified can be replaced by a figurative constant.
2. A numeric literal must have a size which is within the number of positions specified by the PICTURE clause, and must not have a value which would require truncation of non-zero digits.
3. A nonnumeric literal must not exceed the size specified in the PICTURE clause.
4. A signed numeric literal must be associated with a PICTURE clause which provides a signed numeric character-string.
5. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal; here, the group area will be initialized without consideration for the individual elementary or group items contained within the group. The VALUE clause cannot be stated at the subordinate levels within the group.

6. The VALUE clause must not be specified for a group item containing subordinate items with descriptions that include JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

General rules for format 1

1. The VALUE clause is prohibited for external floating-point data items.
2. If a VALUE clause is specified in a data description entry of a data item which is associated with a variable-length data item, the initialization of the data item behaves as if the value of the data item referenced by the DEPENDING ON phrase in the OCCURS clause specified for the variable-length data item had the maximum possible value. A data item is associated with a variable-length data item in any of the following cases:

   a) It is a group data item which contains a variable-length data item.

   b) It is a variable-length data item.

   c) It is a data item that is subordinate to a variable-length data item.
3. The VALUE clause must not conflict with other clauses in the data description of an item or in the data description within the hierarchy of an item. The following rules are applicable:

   If the category of the item being described is numeric, the literal in the VALUE clause must be a numeric literal. If the data item is defined in the WORKING-STORAGE SECTION, the value is aligned in the data item according to the standard alignment rules.

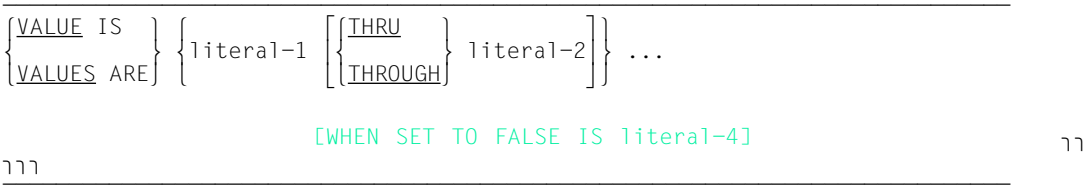
   If the category of the item being described is alphabetic, alphanumeric, alphanumeric edited or numeric edited, the literal in the VALUE clause must be nonnumeric. The literal is aligned in the data item as if the data item had been described as alphanumeric. Initialization of a data item is not affected by any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.
4. A VALUE clause specified in a data description entry that contains an OCCURS clause or in an entry that is subordinate to a data description entry that contains an OCCURS clause causes every occurrence of the associated data item to be assigned the specified initial value.
5. In the WORKING-STORAGE SECTION, the VALUE clause may be used to specify the initial value of any data item; in this case, the clause causes the item to be initialized to the specified value at the start of program execution. If the VALUE clause is not specified in the description of a data item, the initial value of that item is undefined.
6. Format 1 of the VALUE clause must not be specified in the FILE SECTION and LINKAGE SECTION.

Example 3-43 (for format 1)

```
77 FIELD PICTURE IS AA VALUE IS "AA".
```

Here the value of FIELD is initialized to AA.

Format 2



Syntax rules for format 2

1. A format 2 VALUE clause may be used only in connection with condition-names (level-number 88).
2. Level number 88 applies to declarations of condition-names which are associated with a conditional variable; these declarations are called condition-name declarations. A conditional variable is a data item which is followed by one or more condition-name declarations. A condition-name assigns a name to a value or a range of values which a conditional variable may assume at run time. A condition-name can then be "true" or "false" during program execution. A condition-name is not a data item and requires no storage space (see the description of the use of condition-names under the heading "Condition-name conditions", page 219).
3. The specified literals may be replaced by figurative constants.
4. All numeric literals must have a length which is within the number of positions specified by the PICTURE clause for the related elementary item (conditional variable), and must not have a value which would require truncation of non-zero digits.
5. Nonnumeric literals must not exceed the size specified in the PICTURE clause for the related elementary item (conditional variable).
6. A signed numeric literal must be associated with a PICTURE clause which provides a signed numeric character-string.
7. When the THRU/THROUGH phrase is used, the literal preceding THRU/THROUGH must be less than the literal which follows it.
8. The THRU/THROUGH phrase assigns a range of values to the specified condition-name.
9. literal-4 may not be equal to any literal-2.

10. The following must apply to each literal-2, literal-3 pair: literal-4 must be less than literal-2 or greater than literal-3.

General rules for format 2

1. The VALUE clause is prohibited for external floating-point data items.
2. The VALUE clause must not be specified for items whose size, whether explicitly or implicitly, is variable.
3. The VALUE clause must not conflict with other clauses in the data description of an item or in the data description within the hierarchy of an item. The following rules are applicable:
4. If the category of the item being described is numeric, all literals in the VALUE clause must be numeric literals. If the condition-name is defined in the WORKING-STORAGE SECTION, the value is aligned in the data item according to the standard alignment rules.
5. If the category of the item being described is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The value is aligned in the data item as if the data item had been described as alphanumeric.
6. Format 2 of the VALUE clause is only allowed in the FILE, WORKING-STORAGE and LINKAGE SECTIONs. It must not be specified in the REPORT SECTION.
7. The specification of FALSE in the VALUE clause is relevant only when the condition-name is used in a SET-TO-FALSE statement.

Example 3-44 (for format 2)

```
02  CITIES PICTURE 9.
    88  BERLIN VALUE 1.
    88  HAMBURG VALUE 2.
    88  MUNICH VALUE 3.
    88  COLOGNE VALUE 4.
```

Here, CITIES is the conditional variable, and BERLIN, HAMBURG, MUNICH, and COLOGNE are the condition-names. If a statement IF MUNICH GO TO TEST-C were written in the Procedure Division, then the value of the conditional variable CITIES would be compared to the value 3; this statement would be equivalent to the statement IF CITIES IS EQUAL TO 3 GO TO TEST-C.

Example 3-45 (for format 2)

```
02  AGE PICTURE 99.
    88  TWENTIES VALUE 20 THRU 29.
    88  THIRTIES VALUE 30 THRU 39.
```

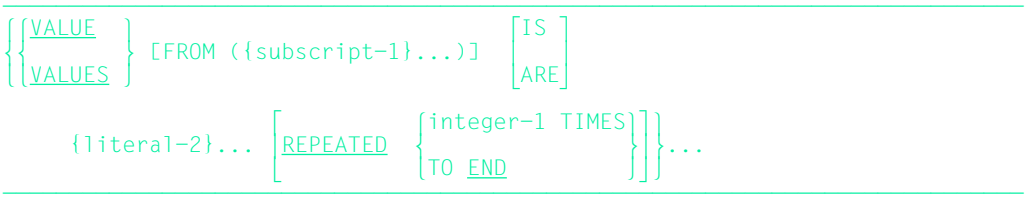
If the statement IF TWENTIES... were to be written in the Procedure Division, the value of the conditional variable AGE would be compared to the values 20, 21, ... and 29. This statement would be equivalent to the statement IF AGE NOT LESS THAN 20 AND NOT GREATER THAN 29...

Example 3-46 (for format 2)

```
02  NAME-OF-DAY PICTURE X(3).
    88  BEGINNING-WEEK VALUE "MON" "TUE" "WED".
    88  END-OF-WEEK    VALUE "THRU" "FRI".
    88  WEEKEND        VALUE "SAT" "SUN".
```

If the statement IF BEGINNING-OF-WEEK... were to be written in the Procedure Division, the conditional variable NAME-OF-DAY would be compared with "MON", "TUE" and "WED". This statement would be equivalent to IF NAME-OF-DAY IS EQUAL TO "MON" OR "TUE" OR "WED".

Format 3



Syntax rules for format 3

1. Format 3 of the VALUE clause must be used only in connection with Working-Storage table elements.
2. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the associated PICTURE clause, and must not have a value which would require truncation of non-zero digits.
3. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the associated PICTURE clause.
4. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause must not be stated at the subordinate levels within this group.
5. The VALUE clause must not be specified for a group item containing items subordinate to it with descriptions including JUSTIFIED or USAGE (other than USAGE IS DISPLAY).

6. When format 3 is specified, the data description entry must contain an OCCURS clause or be subordinate to a data description entry that contains an OCCURS clause.
7. Subscript-1 must be a numeric literal that is an integer. If all subscripts have the value 1, no subscripts need be specified; otherwise, all subscripts required to reference an individual element in a table must be specified.
8. The number of table elements to be initialized is determined as follows:

   a) If integer-1 is not specified, it is the number of repetitions of literal-2.

   b) If integer-1 is specified, it is the number of repetitions of literal-2 times integer-1.

   The number of table elements to be initialized must not exceed the maximum number of occurrences in the table from the point of reference to the end of the table.
9. If multiple format 3 VALUE clauses are specified in an entry:

   a) The TO END phrase may be specified only once.

   b) A given table element may be referenced only once.

General rules

1. All formats of the VALUE clause can be used in one table.
2. Within the same data description entry, if more than one VALUE clause references the same table element, the value defined by the last specified VALUE clause in the data description entry is assigned to the table element.
3. A format 3 VALUE clause initializes a table element to the value of literal-2. The table element initialized is identified by subscript-1. Consecutive table elements are initialized, in turn, to the successive occurrences of the value of literal-2. Consecutive table elements are referenced by augmenting by 1 the subscript that represents the least inclusive dimension of the table. When any reference to a subscript, prior to augmenting it, is equal to the maximum number of occurrences specified by its corresponding OCCURS clause, that subscript is set to 1 and the subscript for the next most inclusive dimension of the table is augmented by 1.
4. If the REPEATED phrase is specified, all occurrences of literal-2 are reused, in the order specified.
   If the TO END phrase is specified, this reuse occurs until the end of the table is reached.
   If the integer-1 TIMES phrase is specified, the occurrences of literal-2 are reused, in the order specified, integer-1 times.
   If the REPEATED phrase is not specified, the occurrences of literal-2 are used, in the order specified, only once.
5. If a VALUE clause is specified in a data description entry of a data item which is associated with a variable-occurrence data item, the initialization of the data item behaves as if the value of the data item referenced by the DEPENDING ON phrase in

the OCCURS clause specified for the variable-occurrence data item is set to the maximum number of occurrences as specified by that OCCURS clause. A data item is associated with a variable-occurrence data item in any of the following cases:

a) It is a group data item which contains a variable-occurrence data item.

b) It is a variable-occurrence data item.

c) It is a data item that is subordinate to a variable-occurrence data item.

6. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:

   If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a data item in the WORKING-STORAGE SECTION, the literal is aligned in the data item according to the standard alignment rules.

   If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric.

   A data item is initialized regardless of whether a BLANK WHEN ZERO or JUSTIFIED clause was specified.

Example 3-47

```
 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.
```

This results in the following field assignments:

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(5,4) = "3"	B(6,3) = "3"	.	
		B(6,4) = "1"	.	

All other table elements are not assigned.

FIELD2: 78 times "x"

FIELD3:	A(1) = "ab␣␣"	A(2) = "c␣␣␣"
	A(3) = "ab␣␣"	A(4) = "c␣␣␣"
	...	...
	A(19) = "ab␣␣"	A(20) = "c␣␣␣"

3.9 Procedure Division

3.9.1 General description

The Procedure Division contains the specific instructions for solving a given data processing problem. COBOL instructions are written in the form of statements.

A **statement** is a syntactically valid combination of words and symbols, beginning with a COBOL verb.

Example of a statement:

```
MOVE A TO B
```

Several statements may be combined into a sentence, groups of sentences into paragraphs, and one or more paragraphs into a section.

Normally, a COBOL statement refers to user-defined data or procedures by means of data-names or procedure-names. References to user-defined words must be unique (see under "Qualification", page 77).

A logical subset of the program, consisting of one or more successive paragraphs or one or more successive sections of the Procedure Division, is called a "procedure". A procedure-name is a word which is used for referring to a paragraph or a section; it consists of a paragraph-name (which may be qualified by a section-name) or a section-name.

There are two types of procedures in the Procedure Division:

- Declaratives, which cannot be executed within the normal sequence of statements in the Procedure Division.
- Nondeclarative procedures that contain statements for normal execution when there are no special exceptional conditions.

The execution of the program begins with the first Procedure Division statement following the declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules for a given statement indicate some other order.

If program segmentation is used, the programmer must divide the entire Procedure Division into named sections. Program segmentation is discussed in chapter 9, "Segmentation".

Structure

General format

Format 1

```
  A      Margin indication
  |
--|-------------------------------------------------------------------------
  |
  V
  PROCEDURE DIVISION [USING {data-name-1}... ].

[ DECLARATIVES.                                  ]
|{section-name SECTION.                          |
|     USE statement.                             |
|[paragraph-name.                                |
|    [sentence]...]...}...                       |
[ END DECLARATIVES.                              ]

 {section-name SECTION [segment-number].

 [paragraph-name.
     [sentence]...]...}...
-------------------------------------------------------------------------------
```

Format 2

```
 A      Margin indication
 |
-|-------------------------------------------------------------------------
 |
 V
 PROCEDURE DIVISION [USING {data-name-1}... ].

 {paragraph-name.
     [sentence]...}...
-------------------------------------------------------------------------------
```

General rules

1. The Procedure Division must begin with the header PROCEDURE DIVISION, followed by a period and a space, unless Program Communication is being used. In this case, the Procedure Division header in a called program may optionally include the USING phrase preceding the period (see chapter 7, "Inter-program communication").
2. The Procedure Division header is followed, optionally, by the declarative portion containing declarative procedures, which is followed, in its turn, by nondeclarative procedures. Each of these procedures consists of statements, sentences, paragraphs, and/or sections in a syntactically valid format.

   The end of the Procedure Division (and the physical end of the source program) is the point in a COBOL source program which is not followed by any further procedures and statements.

3. For a description of the declarative subdivision, see "Declarative subdivision of the Procedure Division" (page 213).
4. If sections are used within the Procedure Division, format 1 must be applied. Otherwise, format 2 may be used.
5. A **section** consists of a section header followed either by zero, one, or more successive paragraphs. (The section header consists of a section-name, followed by the word SECTION and a period; if program segmentation is desired, a space and a segment-number followed by a period may be inserted after the word SECTION.) A section can end in one of three ways: immediately before the next section-name; at the end of the Procedure Division; or in the declaratives subdivision of the Procedure Division immediately before the next section or at the keywords END DECLARATIVES.

   Multiple definitions of section-names, or of paragraph-names within a section, will not be treated as errors by the compiler as long as they are not referenced.

   Paragraph-names and section-names must not be more than 30 characters in length.
6. A **paragraph** consists of a paragraph-name followed by a period and a space, followed by zero, one or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name, or at the end of the Procedure Division, or, in the declaratives portion of the Procedure Division, immediately before the next paragraph-name or section-name, or at the keywords END DECLARATIVES.
7. If one paragraph is in a section, then all of the paragraphs must be in sections.
8. A **sentence** consists of one or more statements, optionally separated by semicolons, spaces, or commas, and is terminated by a period, followed by spaces.
9. A **statement** consists of a syntactically valid combination of words and symbols, and must begin with a COBOL word.

3.9.2 DECLARATIVES

Function

The DECLARATIVES subdivision is an optional portion of the Procedure Division. It contains a group of procedures, called declarative procedures, which are not executed within the normal sequence of statements in the Procedure Division but only when a particular condition occurs.

Declarative procedures are used for performing the following functions:

– input/output label handling

– handling of input/output errors

– special Report Writer functions.

Format (General format in the Procedure Division)

```
  A       Margin indication
  |
  |
--|-------------------------------------------------------------------------
  |
  |
  |
  PROCEDURE DIVISION [USING {data-name-1}... ].
[ DECLARATIVES.                                  ]
|{section-name SECTION.                          |
|     USE statement.                             |
|[paragraph-name.                                |
|    [sentence]...]...}...                       |
[ END DECLARATIVES.                              ]
 {section-name SECTION [segment-number].
 [paragraph-name.
     [sentence]...]...}...
-------------------------------------------------------------------------------
```

Syntax rules

1. Declarative procedures must be placed at the beginning of the Procedure Division, preceded by the keyword DECLARATIVES and followed by a period and a space. Declarative procedures are terminated by the keyword END DECLARATIVES, followed by a period and a space.

2. As indicated in the general format of the Procedure Division, the DECLARATIVES subdivision must be divided into sections. These sections are called declarative sections. Each declarative section contains a group of related procedures, and is preceded by a section header, immediately followed by a USE statement with subsequent period and space.

3. The USE statement defines the type of declarative procedures according to the three functions listed above. The formats of the USE statement are described in detail starting on page 411 ("Sequential file organization": Formats 1 and 2), page 473 ("Relative file organization": Format) and page 626 ("Report Writer": USE BEFORE REPORTING statement).
4. The USE statement itself is never executed; rather, it defines the conditions for executing the declarative procedures specified in the associated section.

3.9.3 Arithmetic expressions

Function

Arithmetic expressions allow the user to combine arithmetic operations.

Format

An **arithmetic expression** may be one of the following:

a) An identifier of a numeric elementary item.

b) A numeric literal.

c) Two arithmetic expressions separated by an arithmetic operator or an arithmetic expression enclosed in parentheses.

Any arithmetic expression may be preceded by a unary plus (+) or a unary minus (−).

Arithmetic operators

The following **operators** may be used in arithmetic expressions:

Binary arithmetic operator	Meaning
+	Addition
−	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary arithmetic operator	Meaning
+	The effect of multiplication by the numeric literal +1
−	The effect of multiplication by the numeric literal −1

As defined by the standard, a binary arithmetic operator must always be preceded and followed by a space.
However, the compiler allows all these operators, with the exception of the addition and subtraction operators, to be used without the enclosing spaces. The subtraction operator (-) must always be preceded and followed by a space.
The addition operator (+) must be followed by a space if it occurs before an unsigned numeric literal. Both operators may be immediately preceded and followed by a parenthesis.

A unary + must be followed by a space, if it is before an unsigned literal.
A unary − always must be followed by a space.

Rules for the formation and evaluation of expressions

1. An arithmetic expression may begin only with a left parenthesis, a unary +, a unary −, an identifier, or a literal; and it may end only with a right parenthesis or a variable (identifier or literal).
2. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression.
3. Table 3-6 shows the permissible combinations of operators, variables and parentheses in arithmetic expressions.

First symbol	Second symbol				
	identifier, literal	arithmetic operator	unary operator	(	)
identifier, literal	–	P	–	–	P
arithmetic operator	P	–	P	P	–
unary operator	P	–	–	P	–
(	P	–	P	P	–
)	–	P	–	–	P

Table 3-6 Valid symbol combinations in arithmetic expressions

P indicates that the two symbols may appear consecutively (permissible pair).
− indicates that the two symbols must not appear consecutively (invalid pair).

4. Parentheses may be used in arithmetic expressions in order to indicate the order in which the elements are to be evaluated.
5. Expressions within parentheses are evaluated first. When nested parentheses are used, evaluation proceeds from the innermost to the outermost set of parentheses.
6. When no parentheses are used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:
   a) Unary plus or minus(evaluated first)
   b) Exponentiation
   c) Multiplication and division
   d) Addition and subtraction(evaluated last)
7. If consecutive operations of the same hierarchical level occur, they are evaluated from left to right.

General rules

1. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution.
2. Arithmetic expressions are used in arithmetic and conditional statements.

Example 3-48

Expression: A + (B − C) * D

Evaluation:
1. B − C (denote result by x)
2. x * D (denote result by y)
3. A + y (final result)

Example 3-49

Expression: A + ((B / C) + (D ** E) * F) − G

Evaluation:
1. B / C (denote result by z)
2. D ** E (denote result by x)
3. x * F (denote result by y)
4. z + y (denote result by a)
5. A + a (denote result by b)
6. b − G (final result)

3.9.4 Conditions

General description

A condition enables the program to select between two alternative paths of execution, depending upon the truth value of a test. There are two categories of conditions: simple conditions and complex conditions.

Simple conditions:

a) Condition-name condition

b) Class condition

c) Switch-status condition

d) Condition-name condition

e) Sign condition

Each of these conditions may be enclosed in parentheses.

Complex conditions:

Complex conditions are formed by combining simple conditions and/or complex conditions with the logical operators AND and OR or by negating these conditions with the logical operator NOT.

Condition-name condition

Function

The condition-name condition causes a conditional variable to be tested to determine whether or not its value is equal to one of the values associated with a specified condition-name. (See VALUE clause for additional information, page 202.)

Format

```
condition-name
```

Syntax rules

1. condition-name specifies the condition-name to be used in the test.
2. If a condition-name is associated with a single value, then the related test is true, only if the value corresponding to the condition-name equals the value of its associated conditional variable.
3. If the condition-name is associated with one or more ranges of values, then the conditional variable is tested to determine whether or not its value falls in the range, including the end values.
4. The condition-name condition is a shorthand form of the relation condition (see example).

   See also "SET statement", format 4 (page 338).

Example 3-50

```
 02  PAY-CLASS PICTURE 9.
     88  HOURLY   VALUE 1.
     88  WEEKLY   VALUE 2.
     88  MONTHLY  VALUE 3.
           ...
     IF  HOURLY GO TO HOUR-PROCEDURE.
```

Here, PAY-CLASS is a conditional variable, and HOURLY, WEEKLY, and MONTHLY are condition-names. If the current value of PAY-CLASS is 1, the result of the test in the IF statement is true. Otherwise, the result is false.

As noted above, the condition-name is a shorthand form of the relation condition. The following statement, which contains a relation condition, is equivalent to the above IF statement:

```
IF PAY-CLASS = 1 GO TO HOUR-PROCEDURE.
```

Class condition

Function

The class condition determines whether an operand is numeric, alphabetic, alphabetic-lower, alphabetic-upper, or whether it contains only characters from a character set specified with class-name in the CLASS clause of the SPECIAL-NAMES paragraph of the Environment Division.

Format

```
                          ⎧NUMERIC          ⎫
                          ⎪ALPHABETIC       ⎪
identifier IS [NOT]       ⎨ALPHABETIC-LOWER ⎬
                          ⎪ALPHABETIC-UPPER ⎪
                          ⎩class-name       ⎭
```

Syntax rules

1. identifier must be a data item described implicitly or explicitly with USAGE IS DISPLAY or COMPUTATIONAL-3 or PACKED-DECIMAL.
2. identifier specifies the data item to be tested.
3. NUMERIC, ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER and class-name (possibly negated by NOT) indicate which characteristic is to be tested.
4. identifier is treated as numeric when its contents consist of a combination of the digits 0 through 9 (with or without sign).
5. identifier is treated as alphabetic when its contents consist of any combination of the characters A through Z and/or a through z and the space character.
6. identifier is treated as alphabetic-lower when its contents consist of a combination of the lowercase letters a through z and the space character.
7. identifier is treated as alphabetic-upper when its contents consist of a combination of the uppercase letters A through Z and the space character.
8. identifier corresponds to class-name when its contents consist solely of a combination of those characters which were defined by means of class-name in the SPECIAL-NAMES paragraph.

9. identifier cannot be tested as numeric if it is defined in the data description entry as an alphabetic data item.

   If no operational sign (PIC 9 or PIC XX) has been defined in the character-string of an identifier described with USAGE IS DISPLAY, identifier is determined to be numeric only if its contents consist of the digits 0-9. If the character-string does not contain an operational sign (PIC 9), but does contain the data format COMP-3 or PACKED-DECIMAL, identifier is determined to be numeric only if its contents are numeric and there is a non-printable plus sign (represented by F).

   If the character-string contains an operational sign definition (PIC S9), identifier is determined to be numeric only if its contents are numeric and a valid operational sign is present (represented by C, D, or F). This is the case when identifier is described either with USAGE IS DISPLAY or with USAGE COMP-3 or PACKED-DECIMAL.

10. An identifier cannot be tested as alphabetic, alphabetic-lower, or alphabetic-upper, or for conformance to class-name, if it is defined as numeric in the data description entry.

11. Table 3-7 lists all permissible formats of the class condition.

Type of identifier	Tests allowed	
Alphabetic	ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, class-name	NOT ALPHABETIC, NOT ALPHABETIC-LOWER, NOT ALPHABETIC-UPPER, NOT class-name
Alphanumeric or alphanumeric-edited or numeric-edited or group item	ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, NUMERIC, class-name	NOT ALPHABETIC, NOT ALPHABETIC-LOWER, NOT ALPHABETIC-UPPER, NOT NUMERIC, NOT class-name
Numeric	NUMERIC	NOT NUMERIC

Table 3-7: Valid formats of the class condition

Switch-status condition

Function

The switch-status condition tests the setting of an implementor-defined user or task switch. The implementor-name and the ON or OFF value associated with the condition must appear in the SPECIAL-NAMES paragraph of the Environment Division.

Format

```
condition-name
```

Syntax rules

1. The result of the test is true if the switch is set to the position corresponding to condition-name.
2. The status of a switch can be changed by means of a format 3 SET statement (see "SET statement", format 3, page 338).

Relation condition

Function

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, or an arithmetic expression.

Format

```
{identifier-1              }                          {identifier-2              }
{literal-1                 }                          {literal-2                 }
{                          } relational-operator      {                          }
{arithmetic-expression-1   }                          {arithmetic-expression-2   }
{index-1                   }                          {index-2                   }
```

Syntax rules

1. The first operand of a relational condition is called the subject of the condition, and the second operand is called the object of the condition. The operands must be written according to the following rules:

   a) The subject and object must not both be literals.

   b) The subject and object must have the same data format, except when two numeric operands are compared.

2. Relational-operator must be one of the operators listed in Table 3-8. It must be preceded and followed by a space.

Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	[Not] greater than
IS [NOT] LESS THAN IS [NOT] <	[Not] less than
IS [NOT] EQUAL TO IS [NOT] =	[Not] equal to
IS GREATER THAN OR EQUAL TO IS > =	Greater than or equal to
IS LESS THAN OR EQUAL TO IS <=	Less than or equal to

Table 3-8: Relational operators

The special symbols <, > and = are not underlined in could be mistaken for other symbols.

3. The relational operator specifies the type of comparison to be made in a relation test.
4. The following rules describe comparisons between numeric operands, comparisons between nonnumeric operands, and comparisons between indexnames and/or index data items. In compare operations, a group item is treated as a nonnumeric data item.

5. **Comparison of numeric operands**

   When two numeric operands are compared, their algebraic values are compared; their lengths (that is, the number of digits they contain) are not significant.

   Unsigned numeric operands are considered to be positive for purposes of comparison.

   Zero is considered to be a unique value, regardless of sign.

   Comparison of two numeric operands is permitted, regardless of the data formats in their respective USAGE clauses.

   Example 3-51

   −50 is less than +5

   +75 is greater than +5

   −100 is less than −10

   −0 is equal to +0

6. **Comparison of nonnumeric operands**

   When two nonnumeric operands are compared, or when a numeric operand is compared with a nonnumeric operand, the comparison is made with respect to the binary collating sequence of the PROGRAM COLLATING (see "OBJECT-COMPUTER paragraph", page 124).

   If one of the operands is numeric, it must be an integer data item or integer literal. Furthermore:

   a) If the nonnumeric operand is an **elementary item** or a nonnumeric literal, the numeric operand will be treated as if it had been transferred to an alphanumeric elementary item equal in size to the numeric item, and as if the contents of this alphanumeric elementary item had been compared with the nonnumeric operand (see "MOVE statement", page 294 and "PICTURE clause", page 167).

   b) If the nonnumeric operand is a **group item**, the numeric operand will be treated as if it had been transferred to a group item equal in size to the numeric data item, and as if the contents of this group item had then been compared with the nonnumeric operand (see "MOVE statement", page 294, and "PICTURE clause", page 167).

c) A numeric operand which is not an integer cannot be compared with a nonnumeric operand.

Another important factor in a nonnumeric comparison is the length of the operands. The size of an operand is equal to the total number of characters within it. There are two cases to consider: the comparison of operands of equal size, and the comparison of operands of unequal size.

a) Comparison of operands of equal size

If the operands are of equal size, the comparison proceeds as follows: The program compares characters in corresponding character positions, starting at the high-order (that is, leftmost) end, and continuing until it encounters two unequal characters or until it reaches the low-order end of the operands.

If the object program encounters a pair of unequal characters, it determines which character has a higher position in the collating sequence. The operand containing the higher character is considered to be the greater operand.

If all pairs of corresponding characters are equal, the operands are considered to be equal.

Example 3-52

In the following examples the binary collating sequence of the EBCDIC character set is assumed, i.e. either PROGRAM COLLATING SEQUENCE has been omitted or PROGRAM COLLATING SEQUENCE IS NATIVE has been specified.

Relationship	Reason
"123" IS GREATER THAN "ABC"	1 (the 1st character in the 1st operand) is greater than A (the 1st character in the 2nd operand).
"SMYTH" IS GREATER THAN "SMITH"	Y (the 3rd character in the 1st operand) is greater than I (the 3rd character in the 2nd operand).
"ABC" IS EQUAL to "ABC"	All characters compare equally.

b) Comparison of operands of unequal size

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

Example 3-53

Relationship	Reason
"CAR" IS GREATER THEN "AUTO"	C (the 1st character of the 1st operand) is greater than A (the 1st character of the 2nd operand).
"SMITH" is less than "SMITHY"	SMITH is space-filled as follows: SMITH␣. The space (sixth character in first operand) is less than Y (sixth character in second operand).

c) Comparisons involving index-names and/or index data items.

The allowable relation tests involving indices and/or index data items, as well as the data items compared in each case, are listed below (for a summary of all relation tests permitted, see "Conditions", page 218).

1. Two indices:
   The occurrence numbers corresponding to the two indices are compared.
2. One index and one integer (the integer may be a numeric data item or a numeric literal):
   The integer is treated as an occurrence number, and is compared with the occurrence number corresponding to the index.
3. One index data item and one index or other index data item:
   The current values of the items (i.e. the displacements from the beginning of the table) are compared.
4. The result of a comparison of an index data item with any data item or literal not specified above will be undefined.

First operand	Second operand			
	Index	Index data item	Data-name (integer only)	Numeric literal (integer only)
Index	Compare occurrence number	Compare without conversion	Compare occurrence number with numeric integer of data-name	Compare occurrence number with literal
Index data item	Compare without conversion	Compare without conversion	Illegal	Illegal
Data-name (integer only)	Compare occurrence number with numeric integer of data-name	Illegal	Compare numbers	Compare numbers
Numeric literal (integer only)	Compare occurrence number with literal	Illegal	Compare numbers	Illegal

Table 3-9: Validity of comparisons using indices and index data items

7. Allowable comparisons

All allowable comparisons are shown in Table 3-10.

First operand	Second operand													
	GR	AL	AN	ANE	NE	FC[2] NNL	ZR NL	ED	BI	ID	EF	IF	IN	IDI
Group element (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN		
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN							
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphanumeric edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN						
Numeric edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN						
Figurative constant (FC) [2] and nonnumeric literal (NNL)	NN	NN	NN	NN	NN			NN						
Figurative constant ZERO (ZR) and numeric literal (NL)	NN		NN	NN	NN			NU	NU	NU	NU	NU	IN[3]	
External decimal (ED)	NN		NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	IN[3]	
Binary (BI)	NN						NU	NU	NU	NU	NU	NU	IN[3]	
Internal decimal (ID)	NN						NU	NU	NU	NU	NU	NU	IN[3]	
Ext. floating point (EF)	NN						NU	NU	NU	NU	NU	NU		
Int. floating point (IF)	NN						NU	NU	NU	NU	NU	NU		
Index name (IN)							IN[3]	IN[3]	IN[3]	IN[3]			TI	ID
Index data item (IDI)													ID	ID

Table 3-10: Permissible comparisons between operands[1]

1) Function values used in table
NN = Comparison as described for nonnumeric operands
NU = Comparison as described for numeric operands
TI = Comparison as described for two index-names
(see "Table handling", page 88ff)
IN = Comparison as described for index-name and numeric integer
(see "Table handling", page 88ff)
ID = Comparison as described for index data item and index-name
or other index data items (see "Table handling", page 88ff).

2) FC includes all figurative constants except ZERO.

3) Valid only if the numeric item is an integer.

Sign condition

Function

The sign condition determines whether or not the algebraic value of a numeric operand (that is, an item described as numeric) is less than, greater than, or equal to zero.

Format

```
{identifier            }          [POSITIVE]
{                      } IS [NOT] {NEGATIVE}
{arithmetic-expression }          {ZERO    }
```

Syntax rules

1. identifier or arithmetic-expression identifies the operand to be tested.
2. POSITIVE, NEGATIVE or ZERO specifies the test to be made.
3. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

Complex conditions

Function

A complex condition consists of a combination of two or more simple conditions.

Format

```
condition {AND} [NOT] condition  [{AND} [NOT] condition] ...
          {OR }                  [{OR }                ]
```

Syntax rules

1. condition specifies a simple condition.
2. Parentheses may be used within a complex condition to improve readability or to modify the normal hierarchical sequence of execution.
3. The simple conditions within a complex condition are separated from each other by logical operators, according to the specified rules. The logical operators must be preceded by a space and followed by a space.
4. A complex condition may comprise up to 60 simple conditions.
5. Table 3-11 lists the logical operators and their meanings.

Operator	Meaning	Example
OR	Logical inclusive Or (either or both)	The expression A OR B is true if A is true, or B is true, or both A and B are true.
AND	Logical conjunction (both)	The expression A AND B is true only if both A and B are true.
NOT	Logical negation	The expression "NOT" A is true only if A is false.

Table 3-11: Logical operators

6. The ways in which conditions and logical operators may be combined are shown in Table 3-12.

First symbol	Second symbol					
	simple-condition	OR	AND	NOT	(	)
simple-condition	–	P	P	–	–	P
OR	P	–	–	P	P	–
AND	P	–	–	P	P	–
NOT	P	–	–	–	P	–
(	P	–	–	P	P	–
)	–	P	P	–	–	P

Table 3-12: Valid symbol pairs of conditions and logical operators[1)]

1) P indicates that the two symbols may be used as a pair.

7. **Rules of precedence for evaluation of expressions**

   The evaluation of complex conditions starts with the innermost pair of parentheses and proceeds through to the outermost pair of parentheses.

   If the order of evaluation is not determined by parentheses, the expression is evaluated according to the following precedence (hierarchical levels):

   – Arithmetic expressions
   – Relational operators
   – NOT conditions
   – AND and its associated conditions are evaluated from left to right.
   – OR and its associated conditions are evaluated last, also proceeding from left to right.
   – If consecutive expressions have the same hierarchical level, they are evaluated from left to right.

Example 3-54

Consider this expression:

```
A IS NOT GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE
```

This expression is evaluated as if the following parentheses had been supplied:

```
(A IS NOT GREATER THAN B) OR (((A+B) IS EQUAL TO C) AND (D IS POSITIVE)).
```

Example 3-55

Table 3-13 shows some of the relationships between logical operators and simple conditions.

Operands	Value of A [1)]	True	False	True	False
	Value of B [1)]	True	True	False	False
Combinations	NOT A	False	True	False	True
	A AND B	True	False	False	False
	A OR B	True	True	True	False
	NOT (A AND B)	False	True	True	True
	NOT A AND B	False	True	False	False
	NOT (A OR B)	False	False	False	True
	NOT A OR B	True	True	False	True

Table 3-13: Use of logical operators

1) A and B represent simple conditions.

Implied subjects and relational operators

Function

When a complex condition is written without parentheses, any relation condition except the first may be abbreviated as follows:

- the subject of the relation condition may be omitted.
- the subject and relational operator of the relation condition may be omitted.

However, the compiler permits the use of parentheses in relation subjects and relation objects which are arithmetic expressions, and in order to affect the sequence in which the logical operators AND and OR are evaluated.

Format of implied subject

```
...subject relational-operator object { AND } [NOT] relational-operator object...
                                      { OR  }
```

Format of implied subject and relational operator

```
...subject relational-operator object { AND } [NOT] object...
                                      { OR  }
```

Syntax rules

1. Within a sequence of relation conditions, both forms of abbreviation may be used. The effect of using such abbreviations is the same as if the omitted subject were replaced by the most recently stated subject, or the omitted relational-operator were replaced by the most recently stated relational-operator.
2. NOT in an abbreviated complex condition is interpreted as follows:
   a) If the word NOT is followed by one of the relational operators GREATER, >, LESS, <, EQUAL, =, then NOT is considered as part of the relevant relational operator.
   b) If the word NOT is followed by one of the other relational operators, then NOT is considered as a logical operator to negate the relevant relation condition.
3. A NOT appearing in front of a left parenthesis remains in effect up to the associated right parenthesis (see example 3-60).

Example 3-56

Implied subjects

```
IF X = Y OR > W OR < Z
```

is equivalent to

```
IF X = Y OR > W OR X < Z
```

In this example, the implied subject is the most recently stated subject, i.e. X.

Example 3-57

Implied subjects and relational operators

```
IF X = Y OR Z OR W
```

is equivalent to

```
IF X = Y OR X = Z OR X = W
```

In this example, the implied subject is the most recently stated subject, i.e. X; and the implied relational operator is the most recently stated operator, i.e. =.

Example 3-58

Implied subject, and implied subject with relational operator

```
X = Y AND > Z OR A
```

is equivalent to

```
X = Y AND X > Z OR X > A
```

Here, since X is the only stated subject, it is substituted in both simple conditions. The most recently stated operator, >, is substituted in the third simple condition.

Example 3-59

```
A > B AND NOT > C OR D
```

is equivalent to

```
A > B AND NOT A > C OR NOT A > D
```

or

```
((A > B) AND (A NOT > C)) OR (A NOT > D)
```

Example 3-60

```
A NOT = "A" AND NOT ("B" OR NOT "C")
```

is equivalent to

```
A NOT = "A" AND NOT (A NOT = "B" OR NOT A NOT = "C")
```

or

```
A NOT = "A" AND A = "B" AND A NOT = "C"
```

or

```
A = "B".
```

3.9.5 Arithmetic statements

Syntax rules

1. All identifiers used in arithmetic statements must be defined as numeric data in the Data Division.
2. All literals used in arithmetic statements must be numeric. They may be floating-point literals.
3. The maximum size of any operand (identifier or literal) is 18 decimal digits.
4. The maximum size of all results after decimal point alignment is 18 decimal digits.
5. When several operands occurring in an arithmetic statement are "overlapped" in a hypothetical data item, aligned on their decimal points, then the maximum size of the data item required (i.e. the composite of operands) is 18 decimal digits (see "ADD statement", page 251 and "SUBTRACT statement", page 345).
6. A maximum of 100 operands may be supplied in one arithmetic statement or arithmetic expression. The number of right and left parentheses () must not exceed 250.
7. The format of any data item involved in computations (for example, an addend, a subtrahend, or a multiplier) cannot contain editing symbols. Operational signs and implied decimal points are not considered to be editing symbols.
8. Identifiers which are used only to receive the result of an arithmetic statement (for example, the identifier used with the GIVING phrase) may be numeric-edited items (see "GIVING phrase", page 241).
9. Condition-names cannot appear as operands.

General rules

1. The operands need not have the same data description; any necessary conversion and decimal point alignment is supplied throughout the calculation (see "MOVE statement", Rules for numeric moves, page 299).
2. If the sending or receiving items of an arithmetic statement, or of an INSPECT, MOVE, SET, STRING or UNSTRING statement, share the same storage area (that is, if the operands overlap), the result of the execution of such a statement is undefined.
3. The results are also undefined if the identifiers contain any data other than numeric data at object time.

Note

If the input operands for an arithmetic statement do not contain valid numeric data, a data error will occur at object time.

4. The following rules apply to evaluation of exponentiation in an arithmetic expression:

   a) If the value of an expression to be raised to a power is zero, the exponent must have a value greater than zero. Otherwise, the size error condition exists.

   b) If the evaluation yields both a positive and a negative real number, the value returned as the result is the positive number.

   c) If no real number exists as the result of the evaluation, the size error condition exists.

5. In evaluating arithmetic statements, the compiler generates a number of arithmetic operations. Depending on the relationship between the various operands, the compiler will generate one or more intermediate result items. These intermediate result items are retained until required for solving the final result of that statement.

   Table 3-14 shows the number of the integer and decimal digits which are stored in the result item according to the operation executed. From this table, it is possible to determine the optimum operand size for the desired precision to be achieved in the statement. On the basis of the decimal places contained in each operand, and by reference to the formulae supplied in the table, the programmer may determine the exact number of positions which the compiler will make available to the result item. However, the result placed in the result item is aligned on the decimal point. Hence decimal point alignment is likewise important in determining the precision of the result.

   Other considerations affecting the results of arithmetic operations are (see also Table 3-14):

   a) If the ROUNDED phrase is specified, then the value of Fd (decimal places in the result) will be Fd+1.

   b) In all additions or subtractions where an operand has a USAGE declared as COMPUTATIONAL or BINARY, or as COMPUTATIONAL-3 or PACKED-DECIMAL, "i" (the calculated number of integer places) is increased by 1.

   If one of the operands has a USAGE declared as COMPUTATIONAL or COMPUTATIONAL-5, then special rules apply that cannot be shown in Table 3-14.

Statement type	Operation	Decimal places in intermediate result (d)	Integer places in intermediate result (i)	If i+d > 30 digits	
				Decimal places	Integer places
Arithmetic	ADD or SUBTRACT (+) or (−)	MAX (Ad, Bd)	MAX (Ai+1, Bi+1)	Fd	30−Fd
	MULTIPLY (*)	Ad+Bd	Ai+Bi	Fd	30−Fd
	DIVIDE (/)	MAX (Fd+1,Bd)	Bi+Ad	Bd−Ad	Bi+Ad
	EXPONEN-TIATION (**)	Fd	(total digits in final result item) less Fd	(not applicable)	(not applicable)
IF or PERFORM	ADD or SUBTRACT (+) or (−)	MAX (Ad, Bd)	30−d	(not applicable)	(not applicable)
	MULTIPLY (*)	Ad+Bd	30−d		
	DIVIDE (/)	Bd	30−d		
	EXPONEN-TIATION (**)	12	18		

Table 3-14: Calculating the integer and decimal places in intermediate results

i = Calculated integer places

Ai = Integer places in first operand [1)]

Bi = Integer places in second operand [2)]

Fi = Integer places in final result

d = Calculated decimal places

Ad = Decimal places in first operand [1)]

Bd = Decimal places in second operand [2)]

Fd = Decimal places in final result

MAX= The greater value of the specified operands in each case

[1)] Divisor in division operation

[2)] Dividend in division operation

3.9.6 Options in arithmetic statements

CORRESPONDING phrase

The CORRESPONDING phrase enables the user to write one statement to perform operations on several elementary items of the same name in different groups.

1. The word CORRESPONDING may be abbreviated as CORR.
2. All identifiers must refer to group items.
3. The descriptions of the identifiers must not contain data items with level numbers 66, 77, or 88, or the USAGE IS INDEX clause.
4. Pairs of data items correspond if the following conditions exist; all other items are ignored for the operation:
5. Both data items have the same name and qualification, up to, but not necessarily including, identifier-1 and identifier-2.
6. None of the data items is declared with FILLER.
7. In case of MOVE CORRESPONDING, at least one of the data items is an elementary item; both data items are elementary items in case of ADD CORRESPONDING or SUBTRACT CORRESPONDING.
8. A data item that is subordinate to identifier-1 or identifier-2 and that is defined with a REDEFINES, OCCURS, or USAGE IS INDEX clause will be ignored; any items which are subordinate to such items are also ignored. However, identifier-1 or identifier-2 may be defined with REDEFINES or OCCURS clauses or be subordinate to data items defined with REDEFINES or OCCURS clauses.
9. The CORRESPONDING phrase cannot be applied to identifiers that are subjected to reference modification.

Example 3-61

In this example, elementary items in EMPLOYEE-RECORD are subtracted from corresponding items in PAYROLL-CHECK.

Procedure Division statement:

```
SUBTRACT CORRESPONDING EMPLOYEE-RECORD FROM PAYROLL-CHECK.
```

Data Division entries:

```
01  EMPLOYEE-RECORD.
    02  EMPLOYEE-NUMBER.
        03  PLANT-LOCATION...
        03  CLOCK-NUMBER.
            04  SHIFT-CODE...
            04  CONTROL-NUMBER...
    02  WAGES.
        03  HOURS-WORKED...
        03  PAY-RATE...
    02  FICA-RATE...
    02  DEDUCTIONS...
```

```
01  PAYROLL-CHECK.
    02  EMPLOYEE-NUMBER.
        03  CLOCK-NUMBER...
        03  FILLER...
    02  DEDUCTIONS.
        03  FICA-RATE...
        03  WITHHOLDING-TAX...
        03  PERSONAL-LOANS...
    02  WAGES.
        03  HOURS-WORKED...
        03  PAY-RATE...
    02  NET-PAY...
    02  EMPLOYEE-NAME.
        03  SHIFT-CODE...
```

According to the rules for the CORRESPONDING phrase, the following subtractions take place:

1st operand	**2nd operand**
HOURS-WORKED OF WAGES OF EMPLOYEE-RECORD	HOURS-WORKED OF WAGES OF PAYROLL-CHECK
PAY-RATE OF WAGES OF EMPLOYEE-RECORD	PAY-RATE OF WAGES OF PAYROLL-CHECK

The following items are not subtracted, for the reasons stated:

Item	**Reason**
EMPLOYEE-NUMBER	Item not elementary item in either group
PLANT-LOCATION OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Name does not appear under PAYROLL-CHECK
CLOCK-NUMBER OF EMPLOYEE-NUMBER	Item is not elementary in one group
SHIFT-CODE OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK
CONTROL-NUMBER OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Name does not appear under PAYROLL-CHECK
WAGES	Name is not elementary in either group
TAX-RATE OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK
DEDUCTIONS	Item not elementary in one group

GIVING phrase

Syntax rules

1. The identifier following the word GIVING may be a numeric-edited item since it is not itself involved in the calculation.
2. When the GIVING phrase is supplied, the result of the arithmetic operation is assigned to the specified identifier.
3. The result stored in the identifier replaces its previous contents. Therefore, it is not necessary to reset the identifier to zero.

Example 3-62

```
ADD A B GIVING C.
```

The value C is set to the sum of A + B, and A and B are not changed.

ROUNDED phrase

Syntax rules

1. If, after decimal point alignment, the number of places following the decimal point in the result of an arithmetic operation is greater than the number of decimal places provided in the resultant identifier, truncation is performed according to the size of this identifier. If rounding is specified, the absolute value of the last significant digit of the resultant identifier is incremented by 1 if the most significant digit of those to be truncated is greater than or equal to 5.
2. If rounding is not desired but truncation of excess digits is required, the last digit of the resultant identifier remains unchanged.
3. When the least significant digits of a resultant identifier are represented by P in the PICTURE character-string for that identifier, rounding or truncation takes place relative to the rightmost digit position for which internal storage is allocated (see example).

   ROUNDED is assumed for COMPUTATIONAL-1 or COMPUTATIONAL-2 result items, and need not be specified for them.

Example 3-63

Calculated result [1)]	Description of result item	Description after rounding	Result without rounding
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)] & represents the operational decimal point.

ON SIZE ERROR phrase

A size error condition exists if, after decimal point alignment, the integer digits in the computed result exceed the number of places provided for them and thus cause an overflow.

Syntax rules

1. Violation of the rules for evaluation of exponentiation always terminates the arithmetic operation and always causes a size error condition (see "Arithmetic statements", general rule 4, page 236f).
2. The ON SIZE ERROR phrase contains an imperative-statement which specifies what actions are to be taken in the event of a size error.
3. The size error condition applies only to the final results of an arithmetic operation and not to intermediate results, except in the case of the MULTIPLY and DIVIDE statements.
4. If the ROUNDED phrase is specified, rounding takes place before the size error check.
5. If the ON SIZE ERROR phrase is specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers remain unchanged from the values they had before execution of the arithmetic statement. The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the imperative-statement specified in the ON SIZE ERROR phrase and execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement and the NOT ON SIZE ERROR phrase, if specified, is ignored.
6. If ON SIZE ERROR is not specified and a size error condition exists after the execution of the arithmetic operations specified by an arithmetic statement, the values of the affected resultant identifiers are undefined. The values of resultant identifiers for which no size error condition exists are the same as they would have been if the size error condition had not resulted for any of the resultant identifiers. After completion of the arithmetic operations, control is transferred to the end of the arithmetic statement, and the NOT ON SIZE ERROR phrase, if present, is ignored.
7. If the size error condition does not exist, control is transferred to the end of the arithmetic statement or to the imperative-statement specified in the NOT ON SIZE ERROR

phrase if it is specified. In the latter case, execution continues according to the rules for each statement specified in that imperative-statement. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred in accordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement specified in the NOT ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement.

8. For an ADD statement with the CORRESPONDING phrase or a SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, the imperative-statement specified in the ON SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.
9. Division by zero always causes a size error condition.

   For COMPUTATIONAL-1 or COMPUTATIONAL-2 data items, division by zero will cause the imperative-statement in the ON SIZE ERROR phrase to be executed.

Example 3-64

```
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 "Size error!" UPON T.
    DISPLAY A UPON T.
    DISPLAY B UPON T.
```

Current value of A: 44

Current value of B: 72

Calculated result: 116

The result item B is too small to accommodate the calculated result and a size error condition occurs. Since ON SIZE ERROR is specified, the statement PERFORM PROC-A is executed. Result item B is unchanged.

3.9.7 Overlapping operands

The following rule applies to all statements:

If a sending and a receiving item in any statement share a part or all of the same storage areas, yet are not defined by the same data description entry, the result of the execution of such a statement is undefined. With certain statements, the results will also be undefined if the sending and receiving items are defined by the same data description entry. Further information is contained in the rules associated with the individual statements.

3.9.8 Incompatible data

Except for the class condition test, the following applies when a data item is referenced in the Procedure Division: If the content of a data item is not compatible with the data class defined for that data item by its PICTURE clause, the result of the operation is undefined.

General rule

Every operation involving a numeric data item which may possibly have nonnumeric contents (e.g. due to a redefinition of the data item or following a MOVE statement using a group item as operand) should be preceded by the IF NUMERIC class test. The operation can only be performed successfully if the class test yields the truth value TRUE.

3.9.9 Statements

ACCEPT statement

Function

The ACCEPT statement transfers small amounts of data to a data item. The data is either read from a system file or made available by the compiler or operating system.

Format 1 reads user input by means of appropriate mnemonic names or supplies information of the operating system and compiler..

Format 2 supplies date and time specifications of the operating system..

Format 3 are used to access the command line of the POSIX subsystem.
Format 4

Format 5 supplies the contents of a BS2000 or POSIX environment variable or the contents of a specific argument from the POSIX command line..

Format 1

```
ACCEPT identifier [ FROM  mnemonic-name ]
```

Syntax rules for format 1

1. identifier can be a group item or an alphabetic, alphanumeric, external decimal or external floating-point data item.
2. mnemonic-name must be specified in the SPECIAL-NAMES paragraph and be associated with one of the following implementor-names:

```
SYSIPT
TERMINAL
CONSOLE
job-variable-name (BS2000 job variable)
COMPILER-INFO
CPU-TIME, PROCESS-INFO, TERMINAL-INFO, DATE-ISO4
```

3. Data is stored aligned to the left in the area indicated by identifier, regardless of the PICTURE character-string associated with the identifier. Incoming data is not edited, and no error checking is performed.
   The only exception to this is CPU-TIME: the CPU time is moved in accordance with the rules of the MOVE statement from a field with the description PIC 9(6)V9(4).
4. SYSIPT, TERMINAL, or CONSOLE specifies the system file from which data is to be read.
   SYSIPT refers to the system file of that name.
   TERMINAL refers to the system file SYSDTA (normally assigned to the data terminal).
   CONSOLE refers to the system console.
5. When entered for a job-variable-name, mnemonic-name references the associated operating system job variable which is to be read in. If the job variable cannot be read in for some reason, the runtime system issues an error message, and the program is then either continued or aborted, as determined by an appropriate compiler directive (see [1]). In the former case, /* is assumed as the value of identifier.
6. When entered for COMPILER-INFO, CPU-TIME, PROCESS-INFO, or TERMINAL-INFO, mnemonic-name specifies the information which is to be requested.

   COMPILER-INFO refers to information provided by the compiler.

   CPU-TIME, PROCESS-INFO, TERMINAL-INFO and DATE-ISO4 refer to information provided by the operating system.
7. If the FROM phrase is omitted, data is read by default from the logical input file SYSIPT. Data can also be read from the logical input file SYSDTA by means of an appropriate compiler directive (see [1]).
8. The execution of ACCEPT statements and the structure of the information provided for the individual functions are described in the "COBOL85 User Guide".

General rules for format 1

1. If the system file specified for an ACCEPT statement is the same as one designated for a READ statement, the results will be unpredictable.
2. An ACCEPT statement for job-variable will be rejected with an error message at object time if the job variables are not present in the operating system.

Example 3-65

for format 1

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES:
    TERMINAL IS T.
  .
  .
PROCEDURE DIVISION.
  ...
    ACCEPT INPUT-DATA FROM T.
```

The mnemonic-name T is linked to the implementor-name TERMINAL in the SPECIAL-NAMES paragraph. The subsequent ACCEPT statement requests data from the system file SYSDTA, which is assigned to TERMINAL and moves this data to the data item called INPUT-DATA.

Format 2

```
ACCEPT identifier  FROM { DATE        }
                        { DAY         }
                        { DAY-OF-WEEK }
                        { TIME        }
```

Syntax rules for format 2

1. The data item specified by identifier must not be an alphabetic elementary item.
2. The ACCEPT statement causes the requested information to be moved to the data item specified by the identifier, in accordance with the rules governing the MOVE statement. DATE, DAY, DAY-OF-WEEK and TIME are special data items and thus are not described in the source program.

General rules for format 2

1. DATE is composed of the data elements "year" of century, "month" of year, and "day" of month. The sequence of these conceptual elementary items is as follows, from left to right: year, month, day. Thus, for example, April 1, 1996 would be expressed as 960401. DATE, if referenced in a COBOL program, is interpreted as if it had been described as an unsigned elementary numeric integer data item, six digits in length (PIC 9(6)).

2. DAY is composed of the data elements "year" of century, and "day" of year. The sequence of these conceptual elementary items is as follows, from left to right: year, day. Thus, for example, April 1, 1998 would be expressed as 98091. DAY, if referenced in a COBOL program, is interpreted as if it had been described as an unsigned elementary numeric integer data item, five digits in length (PIC 9(5)).
3. DAY-OF-WEEK is composed of a single data element whose content represents the day of the week. If referenced in a COBOL program, DAY-OF-WEEK is interpreted as if it had been described as an unsigned elementary numeric integer data item with a length of one digit (PIC 9). In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, ... , 7 represents Sunday.
4. TIME is composed of the data elements "hours", "minutes", "seconds" and "hundredths of a second". TIME is based on the 24-hour clock; thus 2:41 pm, for example, would be expressed as 14410000. If referenced in a COBOL program, TIME is interpreted as if it had been described as an unsigned elementary numeric integer data item, 8 digits in length (PIC 9(8)). The minimum value for TIME is 00000000, the maximum value is 23595900. The last two digits are not supplied by the system, and are therefore always set to zero.

The following three formats of the ACCEPT statement are extensions from the X/Open Portability Guide. They allow access to environment variables and command lines. Access to command lines is meaningful only if the object program is executing in the POSIX subsystem available as of BS2000/OSD V2.0. Execution of the COBOL85 compiler and of any programs generated by it under POSIX is described in the “COBOL85 User Guide” [1].

Format 3

This suppies the current number of arguments in the command line.

```
ACCEPT identifier-1 [ FROM  mnemonic-name-3 ]
    [END-ACCEPT]
```

Syntax rules

1. identifier-1 must refer to an elementary item that is described as an unsigned integer.
2. mnemonic-name-3 must be linked in the SPECIAL-NAMES paragraph with the implementor-name ARGUMENT-NUMBER.

Format 4

This suppies (consecutively) the contents of the arguments in the command line.

```
ACCEPT identifier-2 [ FROM mnemonic-name-4 ]

    [ON EXCEPTION imperative-statement-1       ]
    [NOT ON EXCEPTION imperative-statement-2]
   [END-ACCEPT]
```

Syntax rules

1. identifier-2 must refer to an alphanumeric elementary item.
2. mnemonic-name-4 must be linked in the SPECIAL-NAMES paragraph with the implementor-name ARGUMENT-VALUE.
3. NOT ON EXCEPTION can only be specified if ON EXCEPTION is also specified.

Format 5

This supplies the contents of an environment variable or the contents of a specific argument from the command line. The name of the specified environment variable or the number of the specified argument in the command line must have been defined before by an appropriate DISPLAY statement.

```
ACCEPT identifier-2 [ FROM mnemonic-name-6 ]

    [ON EXCEPTION imperative-statement-1       ]
    [NOT ON EXCEPTION imperative-statement-2]
   [END-ACCEPT]
```

Syntax rules

1. identifier-2 must refer to an alphanumeric elementary item.
2. mnemonic-name-6 must be linked in the SPECIAL-NAMES paragraph with the implementor-name ARGUMENT-VALUE or ENVIRONMENT-VALUE.
3. NOT ON EXCEPTION can only be specified if ON EXCEPTION is also specified.

Note

A detailed example illustrating access to command lines and environment variables can be found in chapter 13 of the “COBOL85 User Guide” [1].

ADD statement

Function

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

Format 1 of the ADD statement stores the sum in one of the operand items. More than one addition may be expressed by specifying more than one result item in the same ADD statement.

Format 2 of the ADD statement stores the sum in a separate result item.

Format 3 of the ADD statement adds the data items of one group item to the corresponding data items of another group item.

Format 1

```
ADD {identifier-1}
    {literal-1   }... TO {identifier-2 [ROUNDED]}...

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-ADD]
```

Syntax rules for format 1

1. Each identifier must refer to an elementary numeric item.
2. The composite of operands is determined by using all of the operands in a given statement and must not contain more than 18 digits (see "Arithmetic statements", page 236).
3. The values of the operands preceding the word TO are added together and the sum is added to the current value of identifier-2... The result is stored in identifier-2 ...
4. END-ADD delimits the scope of the ADD statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the ROUNDED phrase and the (NOT) ON SIZE ERROR phrase are described.

Example 3-66

for format 1

Statement	PICTURE IS of result item	Calculation
ADD A, B TO C, D		A + B + C stored in C A + B + D stored in D
ADD A, B, C TO D	S9999V99	A + B + C + D stored in D as SnnnnVnn
ADD A, 14 TO C ROUNDED	99999	A + 14 + C stored in C as nnnnn; rounded if necessary

Format 2

```
ADD {identifier-1}...  TO  {identifier-2}  GIVING {identifier-3 [ROUNDED]}...
    {literal-1   }         {literal-2   }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-ADD]
```

Syntax rules for format 2

1. Each identifier preceding the GIVING phrase must refer to an elementary numeric item.
2. identifier-3 may refer either to an elementary numeric item or to an elementary numeric-edited data item.
3. The composite of operands is determined by using all of the operands in a given statement, excluding the data items which follow the word GIVING, and must not contain more than 18 digits (see "Arithmetic statements", page 236).
4. The values of the operands preceding the word GIVING are added together, and the sum is stored as the new value of identifier-3.
5. END-ADD delimits the scope of the ADD statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the GIVING, ROUNDED, and (NOT) ON SIZE ERROR phrases are described.

Example 3-67

for format 2

Statement	PICTURE IS of result item	Calculation
ADD A, B, C GIVING D.	9999.99	A + B + C stored in D as nnnn.nn
ADD A, B, 43.6 GIVING D ON SIZE ERROR GO TO O-FLOW END-ADD.	99V99	A + B + 43.6 stored in D. If the integer result is greater than 2 digits, the size error condition occurs and the GO TO statement specified in the SIZE ERROR phrase is executed.

Format 3

```
ADD {CORR          } identifier-1 TO identifier-2 [ROUNDED]
    {CORRESPONDING}

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-ADD]
```

Syntax rules for format 3

1. Each identifier must refer to a group item.
2. The composite of operands is determined separately for each pair of corresponding data items, and must not contain more than 18 digits (see "Arithmetic statements", page 236).
3. Elementary items within the first operand (identifier-1) are added to the corresponding elementary items in the second operand (identifier-2 ...). The results are stored in the items of the second operand.
4. END-ADD delimits the scope of the ADD statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the CORRESPONDING, ROUNDED, and (NOT) ON SIZE ERROR phrases are described.

Example 3-68

for format 3

Refer to the description of the CORRESPONDING phrase for an example of the use of this option (page 239).

ALTER statement

Function

The ALTER statement modifies one or more GO TO statements, thereby altering a predetermined sequence of operations.

Format

```
ALTER  {procedure-name-1 TO [PROCEED TO] procedure-name-2}...
```

Syntax rules

1. procedure-name-1... must be names of paragraphs which contain only one sentence consisting of a GO TO statement without the DEPENDING phrase.
2. procedure-name-2... must be paragraph names or section names in the Procedure Division.
3. During the execution of the program, the ALTER statement modifies the GO TO statement specified under procedure-name-1... so that subsequent executions of the modified GO TO statement cause control to be transferred to procedure-name-2... (see "GO TO statement", page 278).

General rules

1. A GO TO statement in a section whose segment number is greater than or equal to 50 must not be referenced by an ALTER statement in a section with a different segment number.
2. A GO TO statement in a section whose segment number is less than 50 may be referenced by an ALTER statement in any section, even if the GO TO statement thus referred to is contained in a program segment which has not yet been called for execution.

COMPUTE statement

Function

The COMPUTE statement is used to assign the value of a data item, literal, or arithmetic expression to a data item.

Format

```
COMPUTE  {identifier-1 [ROUNDED]}... = {identifier-2          }
                                       {literal-1             }
                                       {arithmetic-expression }

       [ON SIZE ERROR imperative-statement-1]

       [NOT ON SIZE ERROR imperative-statement-2]

       [END-COMPUTE]
```

Syntax rules

1. identifier-1... must refer to an elementary numeric item or an elementary numeric-edited data item.
2. identifier-2 must refer to an elementary numeric item.
3. The arithmetic-expression specified in the COMPUTE statement permits the use of any meaningful combination of identifiers (which must satisfy the general rules for data-names in simple arithmetic operations), literals, and arithmetic operands; if necessary, they may also be in parentheses (see also "Arithmetic expressions", page 215).
4. If identifier-2 or literal-1 is specified, the value of identifier-1 is set equal to the value of identifier-2 or literal-1.
5. When an arithmetic-expression is used, the value of that arithmetic-expression is first calculated and then stored as the new value of identifier-1...
6. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite of operands and/or receiving data items which are imposed by the ADD and SUBTRACT statements.
7. Up to 50 data-names may be specified in a COMPUTE statement.
8. END-COMPUTE delimits the scope of the COMPUTE statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the ROUNDED and (NOT) ON SIZE ERROR phrases are described.

Example 3-69

Statement	Calculation
COMPUTE A = (B + C) / D − E.	The value of the expression (B + C) / D − E is assigned to A. The precedence rules for evaluating expressions apply when calculating values.
COMPUTE A = 2.	The value 2 is assigned to A.

CONTINUE statement

The CONTINUE statement is a no operation statement. It indicates that no executable statement is present. Processing is continued with the next executable statement.

Format

```
CONTINUE
```

Syntax rule

The CONTINUE statement may be used anywhere a conditional statement or an imperative-statement may be used.

General rule

The CONTINUE statement has no effect on the execution of the program.

Example 3-70

```
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.
INPUT-PAR.
    DISPLAY "Enter upper limit N" UPON T.
    ACCEPT N FROM T.
    IF N NUMERIC
    THEN
      CONTINUE
    ELSE
      DISPLAY "Incorrect entry" UPON T
      PERFORM INPUT-PAR
    END-IF.
COMPUTATION.
    PERFORM WITH TEST BEFORE VARYING K FROM 1 BY 1 UNTIL K > N
      COMPUTE E = K ** 3
      ADD E TO Z
    END-PERFORM
    DISPLAY "Result = " Z UPON T.
FINISH-PAR.
    STOP RUN.
```

The effect of CONTINUE is to make the IF statement syntactically correct even though the THEN branch does not contain an executable statement.

Example 3-71

```
READ INPUT-FILE AT END CONTINUE.
```

AT END is used in order to avoid program abortion at the end of the file; CONTINUE specifies the unconditional statement which is required by the statement syntax, even though nothing is to be done at this point in the program.

DISPLAY statement

Function

Format 1 is used to output small quantities of data..

Format 2 is used to access a POSIX command line.

Format 3 are used to access a BS2000 or POSIX environment variable.

Format 4

Format 1

```
DISPLAY {literal-1   }... [ UPON mnemonic-name ]  [WITH NO ADVANCING]
        {identifier-1}
```

Syntax rules

1. literal-1 or identifier-1 serves to specify the operands in the order they are to be output. If necessary, the contents of the data item specified by "identifier" are converted to external formats according to the following rules:

   Internal decimal and binary items are converted to external decimal data items. Internal floating-point data items are converted to external floating-point data items.

   No other data items require conversion.

   If one of the operands is a figurative constant (except ALL literal), it is output with length 1 . If one of the operands is the figurative constant ALL literal, the literal is output once. If literal-1 is numeric, it must be an unsigned integer.

2. mnemonic-name must be specified in the SPECIAL-NAMES paragraph and be associated with one of the following implementor-names:

   CONSOLE, PRINTER, PRINTER01 - PRINTER99, SYSOPT, TERMINAL, job-variable-name (BS2000 job variable).

3. The mnemonic-name for SYSOPT, TERMINAL, CONSOLE, PRINTER and PRINTER01 - PRINTER99 specifies the system file into which the data is to be written.
   SYSOPT specifies the system file with the same name.
   TERMINAL specifies the system file SYSOUT.
   CONSOLE specifies the system operator console.
   PRINTER specifies the system file SYSLST, and PRINTER01 - PRINTER99 refer to the system files SYSLST01 - SYSLST99.

4. If the UPON phrase is omitted, the data is written by default to the logical output file SYSLST. Data can also be written to the logical output file SYSOUT by means of an appropriate compiler directive (see [1]).

General rules

1. A maximum logical record size is assumed for each device. These sizes are listed in Table 3-15.

Device	Maximum record size
CONSOLE	180 characters
PRINTER PRINTER01_PRINTER99	132 characters + 1 control byte
SYSOPT	80 characters: 72 data bytes; bytes 73-80 contain the first 8 bytes of the PROGRAM-ID name.
TERMINAL	8192 characters

Table 3-15: Maximum logical record size for the DISPLAY statement

2. When a DISPLAY statement contains more than one operand, the contents of the specified operands and literals are displayed adjacent to each other, from left to right.
3. For output to printer, the following entries cause a line feed: DISPLAY, WRITE and WRITE AFTER ADVANCING. A WRITE statement without ADVANCING phrase and a WRITE statement with BEFORE ADVANCING phrase causes the printer to space after printing. Therefore, mixed use of DISPLAY and WRITE statements on the same device within the same program may cause two or more lines to overprint. Overprinting is not possible on laser printers.
4. The maximum record length for job variables is 256 characters. If the total number of characters in the operands exceeds the maximum record length, the record will be truncated to the maximum length.
5. When a job variable is used as a monitoring job variable (MONJV), the system protects the first 128 bytes (system portion) of this job variable against write access. Therefore, only that portion of a record that begins at position 129 will be written from position 129 of the monitoring job variable. In all other respects, general rule 4 applies for monitoring job variables.

For further information see the "COBOL85 User Guide" [1].

Example 3-72

```
SPECIAL-NAMES.
     TERMINAL IS SPECIAL-OUTPUT.
     ...
PROCEDURE DIVISION.
     ...
     DISPLAY OUTPUT-MESSAGE UPON SPECIAL-OUTPUT.
```

Here, the mnemonic-name SPECIAL-OUTPUT is associated with the implementor-name TERMINAL in the SPECIAL-NAMES paragraph. The DISPLAY statement writes the current contents of OUTPUT-MESSAGE on SYSOUT.

Example 3-73

```
DISPLAY "Hello world".
```

Since the UPON phrase is omitted, the literal "Hello world" is written to the logical output device SYSLST. If the compiler directive COMOPT REDIRECT-ACCEPT-DISPLAY=YES (in SDF: ACCEPT-DISPLAY-ASSGN=*TERMINAL) is specified, the literal is written to the output file SYSOUT (see [1]).

The following three formats of the DISPLAY statement are extensions from the X/Open Portability Guide. They allow access to environment variables and command lines. Access to command lines is meaningful only if the object program is executing in the POSIX subsystem available as of BS2000/OSD V2.0. Execution of the COBOL85 compiler and of any programs generated by it under POSIX is described in the “COBOL85 User Guide” [1].

Format 2

This format sets the number of the argument in the command line which is subsequently accessed via an ACCEPT statement.

Syntax rules

1. identifier-3 must refer to an elementary item that is described as an unsigned integer.
2. integer-1 must be unsigned.
3. mnemonic-name-3 must be linked in the SPECIAL-NAMES paragraph with the implementor-name ARGUMENT-NUMBER.

Format 3

This format sets the name of the environment variable which is subsequently accessed by an ACCEPT or DISPLAY statement.

```
DISPLAY { identifier-4 } UPON mnemonic-name-5 [END-DISPLAY]
        { literal-1    }
```

Syntax rules

1. identifier-4 must refer to an alphanumeric elementary item.
2. literal-1 must be a nonnumeric literal.
3. mnemonic-name-5 must be linked in the SPECIAL-NAMES paragraph with the implementor-name ENVIRONMENT-NAME.

Format 4

This format writes to the environment variable specified previously in a format-3 DISPLAY statement.

```
DISPLAY { identifier-2 } UPON mnemonic-name-6
        { literal-2    }

    [ON EXCEPTION imperative-statement-1       ]
    [NOT ON EXCEPTION imperative-statement-2]
  [END-DISPLAY]
```

Syntax rules

1. identifier-2 must refer to an alphanumeric elementary item.
2. literal-2 must be a nonnumeric literal.
3. mnemonic-name-6 must be linked in the SPECIAL-NAMES paragraph with the implementor-name ENVIRONMENT-VALUE.
4. NOT ON EXCEPTION can only be specified if ON EXCEPTION is also specified.

A detailed example of accessing command lines and environment variables can be found in chapter 13 of the COBOL User Guide [1].

DIVIDE statement

Function

The DIVIDE statement is used to divide one numeric operand by another and store the result.

Format 1 of the DIVIDE statement stores the quotient in the dividend item.

Format 2 of the DIVIDE statement stores the quotient in more than one separate result item.

Format 3 of the DIVIDE statement uses the GIVING phrase for storing the quotient and generates the division remainder by means of the REMAINDER phrase.

Format 1

```
DIVIDE {identifier-1} INTO {identifier-2 [ROUNDED]}...
       {literal-1   }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-DIVIDE]
```

Syntax rules for format 1

1. Each identifier must refer to an elementary numeric item.
2. The value of identifier-2 is divided by the value of identifier-1 or literal-1. The quotient then replaces the current value of identifier-2 and so on.
3. The maximum size of the quotient after decimal point alignment is 18 decimal digits.
4. Division by zero always results in overflow (SIZE ERROR).
5. In the case of division with ON SIZE ERROR, it is still possible for a DIVIDE-ERROR to occur since no test is made for quotient overflow (only for division by zero).
6. END-DIVIDE delimits the scope of the DIVIDE statement.

Additional rules are given under "Options in arithmetic expressions" (page 239ff), where the ROUNDED, (NOT) ON SIZE ERROR, and GIVING phrases are described.

Example 3-74

for format 1

Statement	PICTURE of result item	Calculation
DIVIDE A INTO B	9(4)V9(2)	B / A stored as nnnnVnn in B

Format 2

```
DIVIDE {identifier-1} {INTO} {identifier-2}
       {literal-1   } { BY } {literal-2   }

        GIVING {identifier-3 [ROUNDED]}...

       [ON SIZE ERROR imperative-statement-1]

       [NOT ON SIZE ERROR imperative-statement-2]

       [END-DIVIDE]
```

Syntax rules for format 2

1. identifier-1 or identifier-2 must refer to an elementary data item.
2. identifier-3... may refer to an elementary numeric item or to an elementary numeric-edited item.
3. When the INTO phrase is used, the value of identifier-2 or literal-2 is divided by the value of identifier-1 or literal-1; when the BY phrase is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. The quotient is stored in identifier-3... .
4. The maximum size of the quotient after decimal point alignment is 18 decimal digits.
5. Division by zero always results in overflow (SIZE ERROR).
6. In the case of division with ON SIZE ERROR, it is still possible for a divide error to occur since no test is made for quotient overflow (only for division by zero).
7. END-DIVIDE delimits the scope of the DIVIDE statement.

Additional rules are given under "Options in arithmetic expressions" (page 239ff), where the ROUNDED, (NOT) ON SIZE ERROR, and GIVING phrases are described.

Example 3-75

for format 2

Statement	PICTURE IS of result item (C):	Calculation
DIVIDE A INTO B GIVING C ROUNDED	S999V99 for C	B / A stored in C as nnnVnn after rounding, if necessary
DIVIDE A BY B, GIVING C, D ROUNDED	9(5) for C 9(4) for D	A / B stored in C as nnnnn, in D as nnnn, after rounding the rightmost character, if necessary.

Format 3

```
DIVIDE {identifier-1} {INTO} {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   } {    } {literal-2   }
        REMAINDER identifier-4
       [ON SIZE ERROR imperative-statement-1]
       [NOT ON SIZE ERROR imperative-statement-2]
       [END-DIVIDE]
```

Syntax rules for format 3

1. identifier-1 or identifier-2 must refer to an elementary numeric data item.
2. identifier-3 or identifier-4 may refer to an elementary numeric data item or to an elementary numeric-edited data item.
3. When INTO is used, the value of identifier-2 or literal-2 is divided by the value of identifier-a or literal-1. When the BY phrase is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2. The quotient is stored in identifier-3.
4. When the REMAINDER phrase is used, the remainder of the division is stored in identifier-4.
   The remainder is calculated by subtracting the product of the quotient and the divisor from the dividend.
   If identifier-3 is defined as an elementary numeric-edited item, then the remainder is calculated by using an intermediate item for the quotient, containing the value in an unedited format.
   If both the ROUNDED and the REMAINDER phrases are supplied, then the remainder is calculated by using an intermediate item for the quotient, containing the quotient of the DIVIDE statement in a truncated rather than rounded format.

5. If ON SIZE ERROR is specified, and overflow occurs in the quotient, then the remainder will not be calculated. In this case, the contents of the data items referenced by identifier-3 and identifier-4 are therefore unchanged.

   If overflow occurs in the remainder, the value of the data item referenced by identifier-4 is not changed.

6. The precision of the data item required for the REMAINDER phrase (identifier-4) is determined by the calculations described above. Appropriate decimal point alignment and truncation (rather than rounding) are performed as necessary for the contents of the data item referenced by identifier-4.
7. The maximum size of the quotient after decimal point alignment is 18 decimal digits.
8. Division by zero always results in overflow (SIZE ERROR).
9. END-DIVIDE delimits the scope of the DIVIDE statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff, where the ROUNDED, (NOT) ON SIZE ERROR, and GIVING phrases are described.

Example 3-76

for format 3

Statement	**Result item (C) PICTURE IS:**	**Calculation**
DIVIDE A BY B, GIVING C REMAINDER D	9(5) for C 9(2) for D	A / B stored in C as nnnnn, the remainder, e.g. A - C * B, stored in D as nn.

EVALUATE statement

Function

The EVALUATE statement describes a multi-branch, multi-join structure. It can cause multiple conditions to be evaluated. The subsequent action of the program depends on the results of these evaluations.

Format

```
EVALUATE { identifier-1 | literal-1 | expression-1 | TRUE | FALSE }
         [ ALSO { identifier-2 | literal-2 | expression-2 | TRUE | FALSE } ] ...

  { WHEN { ANY | condition-1 | subcondition-1 | TRUE | FALSE }
         { [NOT] { identifier-3 | literal-3 | arithm-expressn-1 }
                 [ { THROUGH | THRU } { identifier-4 | literal-4 | arithm-expressn-2 } ] } }

  [ ALSO { ANY | condition-2 | subcondition-2 | TRUE | FALSE }
         { [NOT] { identifier-5 | literal-5 | arithm-expressn-3 }
                 [ { THROUGH | THRU } { identifier-6 | literal-6 | arithm-expressn-4 } ] } ] ... } ...

   imperative-statement-1} ...

  [WHEN OTHER imperative-statement-2]

  [END-EVALUATE]
```

Syntax rules

1. The operands or the words TRUE and FALSE which appear before the first WHEN phrase of the EVALUATE statement are referred to individually as selection subjects and collectively, for all those specified, as the set of selection subjects.
2. The operands or the words TRUE, FALSE and ANY which appear in a WHEN phrase of an EVALUATE statement are referred to individually as selection objects and collectively, for all those specified in a single WHEN phrase, as the set of selection objects.
3. The words THROUGH and THRU are equivalent.
4. Two operands connected by a THROUGH phrase must be of the same class. The two operands thus connected constitute a single selection object.

5. The number of selection objects within each set of selection objects must be equal to the number of selection subjects.
6. Each selection object within a set of selection objects must correspond to the selection subject having the same ordinal position within the set of selection subjects according to the following rules:
   a) Identifiers, literals, or arithmetic expressions appearing within a selection object must be valid operands (according to the rules for relation conditions) for comparison with the corresponding operand in the set of selection subjects.
   b) condition-1, condition-2, or the words TRUE or FALSE appearing as a selection object must correspond to a conditional expression or the words TRUE or FALSE in the set of selection subjects.
   c) The word ANY may correspond to a selection subject of any type.
   d) subcondition-1 or subcondition-2 may not start with an arithmetic operator.
   e) subcondition-1 or subcondition-2 appearing within a selection object must correspond to an identifier, a literal or an arithmetic expression in the set of selection subjects. They must be specified in such a way that a valid, simple condition can be created by inserting the corresponding subject in front of the subcondition.

General rules

1. The execution of the EVALUATE statement operates as if each selection subject and selection object were evaluated and assigned a numeric or nonnumeric value, a range of numeric or nonnumeric values, or a truth value. These values are determined as follows:
   a) Any selection subject specified by identifier-1, identifier-2, and any selection object specified by identifier-3, identifier-5, without either the NOT or the THROUGH phrase, are assigned the value and class of the data item referenced by the identifier.
   b) Any selection subject specified by literal-1, literal-2, and any selection object specified by literal-3, literal-5, without either the NOT or the THROUGH phrase, are assigned the value and class of the specified literal. If literal-3, literal-5, is the figurative constant ZERO, it is assigned the class of the corresponding selection subject.
   c) Any selection subject in which expression-1, expression-2, is specified as an arithmetic expression and any selection object, without either the NOT or the THROUGH phrase, in which arithmetic-expression-1, arithmetic-expression-3, is specified are assigned a numeric value according to the rules for evaluating an arithmetic expression.

d) Any selection subject in which expression-1, expression-2, is specified as a conditional expression and any selection object in which condition-1, condition-2, is specified are assigned a truth value according to the rules for evaluating conditional expressions.

e) Any selection subject or any selection object specified by the words TRUE or FALSE is assigned a truth value. The truth value "true" is assigned to those items specified with the word TRUE, and the truth value "false" is assigned to those items specified with the word FALSE.

f) Any selection specified by the word ANY is not further evaluated.

g) If the THROUGH phrase is specified for a selection object, without the NOT phrase, the range of values includes all permissible values of the selection subject that are greater than or equal to the first operand and less than or equal to the second operand according to the rules for comparison.

h) If the NOT phrase is specified for a selection object, the values assigned to that item are all permissible values of the selection subject not equal to the value, or not included in the range of values, that would have been assigned to the item had the NOT phrase not been specified.

i) If a subcondition is specified for a selection object, it is assigned the truth value of the condition resulting from the addition of the object to the result of the evaluation of the related subject.

2. The execution of the EVALUATE statement then proceeds as if the values assigned to the selection subjects and selection objects were compared to determine if any WHEN phrase satisfies the set of selection subjects. This comparison proceeds as follows:

   a) Each selection object within the set of selection objects for the first WHEN phrase is compared with the selection subject having the same ordinal position within the set of selection subjects. One of the following conditions must be satisfied if the comparison is to be satisfied:

      – If the items being compared are assigned numeric or nonnumeric values, or a range of numeric or nonnumeric values, the comparison is satisfied if the value, or one of the range of values, assigned to the selection object is equal to the value assigned to the selection subject according to the rules for comparison (see "Relation conditions", page 223.)

      – If the object is a subcondition, the comparison is satisfied if the object was assigned the truth value "true".

      – If the items being compared are assigned truth values, the comparison is satisfied if the items are assigned the identical truth value.

      – If the selection object being compared is specified by the word ANY, the comparison is always satisfied regardless of the value of the selection subject.

b) If the above comparison is satisfied for every selection object within the set of selection objects being compared, the WHEN phrase containing that set of selection objects is selected as the one satisfying the set of selection subjects.

c) If the above comparison is not satisfied for one or more selection objects within the set of selection objects being compared, that set of selection objects does not satisfy the set of selection subjects.

d) This procedure is repeated for subsequent sets of selection objects, in the order of their appearance in the source program, until either a WHEN phrase satisfying the set of selection subjects is selected or until all sets of selection objects are exhausted.

3. After the comparison operation is completed, execution of the EVALUATE statement proceeds as follows:

   a) If a WHEN phrase is selected, execution continues with the first imperative-statement following the selected WHEN phrase. There is a branch to END-EVALUATE after the last imperative-statement is executed.

   b) If no WHEN phrase is selected and a WHEN OTHER phrase is specified, execution continues with the imperative-statement of the WHEN OTHER phrase.

   c) An EVALUATE statement is terminated when one of the following conditions is satisfied:

      – imperative-statement-1 of the selected WHEN phrase is executed,

      – imperative-statement-2 is executed,

      – no WHEN phrase is selected and no WHEN OTHER phrase is specified.

Example 3-77

```
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 "Enter value for A" UPON T.
    ACCEPT A FROM T.
    DISPLAY "Enter value for B" UPON T.
    ACCEPT B FROM T.
    DISPLAY "Enter value for C" UPON T.
    ACCEPT C FROM T.
    DISPLAY "Enter value for D" UPON T.
    ACCEPT D FROM T.
TEST1.
    EVALUATE A + B ALSO C + D
    WHEN 5 ALSO 5
      DISPLAY "Values correct" UPON T
    WHEN OTHER
      DISPLAY "Values incorrect" UPON T
    END-EVALUATE.
FINISH-PAR.
    STOP RUN.
```

Example 3-78

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BSP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
 SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TYPE-OF-ORDER PIC 9.
    88 ON-SITE       VALUE 1.
    88 IN-WRITING    VALUE 2 THRU 4.
01  CUSTOMER-TYPE PIC X.
    88 PRIVATE        VALUE "1".
    88 BUSINESS       VALUE "2".
01  WEIGHT        PIC 9999.
01 SHIPPING-MODE  PIC 9.
   88 PICK-UP     VALUE 1.
   88 MAIL        VALUE 2.
   88 RAIL        VALUE 3.
   88 UPS         VALUE 4.
PROCEDURE DIVISION.
PROC SECTION.
DIALOG.
    DISPLAY "Enter type-of-order" UPON T.
    DISPLAY "  on-site = 1, in-writing = 2-4 " UPON T.
    ACCEPT TYPE-OF-ORDER FROM T.
    DISPLAY "Customer-type" UPON T.
    DISPLAY "Business = 2 , Private = 1 " UPON T.
    ACCEPT CUSTOMER-TYPE FROM T.
    DISPLAY "Enter weight" UPON T.
    ACCEPT WEIGHT FROM T.
DETERMINATION-SHIPPING-MODE.
    EVALUATE TRUE ALSO TRUE ALSO TRUE
        WHEN PRIVATE ALSO ON-SITE ALSO ANY
        WHEN BUSINESS ALSO ON-SITE ALSO ANY
            SET PICK-UP TO TRUE
        WHEN PRIVATE ALSO IN-WRITING ALSO WEIGHT < 5
            SET MAIL TO TRUE
        WHEN BUSINESS ALSO IN-WRITING ALSO WEIGHT < 10
            SET UPS TO TRUE
        WHEN OTHER SET RAIL TO TRUE
    END-EVALUATE.
OUTPUT-PAR.
    DISPLAY "Shipping-mode = " SHIPPING-MODE UPON T.
    STOP RUN.
```

Explanation:

The selection objects ON-SITE, IN-WRITING, PRIVATE, BUSINESS (condition-names) are evaluated with respect to their truth value. The selection subjects are represented by the three TRUEs; all three selection subjects (= set of selection subjects) have the truth value "true". A set of selection objects satisfies the condition of the set of selection subjects when all selection objects in the set (except those specified by ANY) are assigned the truth value "true" (a WHEN phrase corresponds to a set of selection objects). The statement which follows this WHEN phrase will then be executed. If none of the specified sets of selection objects satisfies the comparison, the statement of the WHEN OTHER phrase is executed. The ANY phrase must be used in the first two WHEN phrases, because only two condition-names are tested for their truth value in these WHEN phrases, whereas three condition-names are tested for their truth value in the other WHEN phrases, and the number of selection objects within the set of selection objects must correspond to the number of selection subjects.

EXIT statement

Function

The EXIT statement provides a general exit at the end of a series of procedures.

Format

```
EXIT.
```

Syntax rules

1. The EXIT statement must be preceded by a paragraph-name. It must be the only statement in the paragraph.
2. The EXIT statement is used only to assign a procedure-name to a given point in the program. It has no other effect on the execution of the program.

General rules

1. The EXIT statement, when supplied at the end of a series of procedures, enables the normal execution of that sequence of procedures to be interrupted, passing control directly to the end of the procedure sequence.
2. If control reaches an "EXIT paragraph" and no associated PERFORM or USE statement is active, then control passes to the first sentence of the next paragraph.

Example 3-79

```
PROCEDURE DIVISION.
    ...
    PERFORM X-PAR THRU Y-PAR.
    ...
X-PAR.
    ...
    IF A IS ZERO, GO TO Y-PAR.
    ...
Y-PAR.
    EXIT.
Z-PAR.
    ...
```

Here, the "EXIT paragraph" is the last procedure covered by the PERFORM statement. If the value of A is zero, then the GO TO statement interrupts the normal flow of execution of the statements ranging from X-PAR to Y-PAR, and passes control directly to the end of the range of procedures specified by the PERFORM statement. After this, program execution resumes with the next statement after PERFORM.

EXIT PERFORM statement

Function

The EXIT PERFORM statement makes it possible to branch to the end of the PERFORM statement or to a repetition of the loop from an in-line PERFORM statement.

Format

```
EXIT [TO TEST OF] PERFORM.
```

Syntax rules

1. An EXIT [TO TEST OF] PERFORM statement can only be specified within an in-line PERFORM.
2. An EXIT TO TEST OF PERFORM statement can only refer to format 2, 3, or 4 PERFORM statements (see "PERFORM statement", page 303).

General rules

1. The associated in-line PERFORM statement is exited during an EXIT PERFORM.
2. Depending on the format of the PERFORM statement, different branches are made as the result of EXIT TO TEST OF PERFORM:
   a) In a format 2 PERFORM, control passes to the test of "end of loop".
   b) In a format 3 PERFORM, control passes to the test of "UNTIL condition".
   c) When TEST AFTER is specified in format 4, control passes to the test of "UNTIL condition". If the condition is satisfied, the PERFORM statement is terminated. If the condition is not satisfied, augmentation is carried out.
   d) When TEST BEFORE is specified in format 4, control is passed to the increment counter. The test of "UNTIL condition" then follows.

Example 3-80

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXITP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  INPUTA          PIC 99.
01  INPUT-STATUS    PIC X VALUE LOW-VALUE.
  88  INPUT-FINISH  VALUE HIGH-VALUE.
01  AMOUNT          PIC 9(3).
01  COUNTER         PIC 99.
01  AVERAGE         PIC Z9.9(2).
PROCEDURE DIVISION.
PROC SECTION.
COMPUTATION.
    MOVE 0 TO COUNTER AMOUNT
    DISPLAY "Calculation of the average of numbers" UPON T
    PERFORM WITH TEST AFTER UNTIL INPUT-FINISH
       DISPLAY "Input numbers (2 digits, end at input 00)" UPON T
       ACCEPT INPUTA FROM T
       IF INPUTA IS NOT NUMERIC
       THEN
         DISPLAY "Input is not numeric or not with 2 digits!" UPON T
         EXIT TO TEST OF PERFORM
       END-IF
       IF INPUTA = 0
       THEN
         SET INPUT-FINISH TO TRUE
         EXIT TO TEST OF PERFORM
       END-IF
       ADD 1 TO COUNTER
       ADD INPUTA TO AMOUNT
    END-PERFORM
    IF COUNTER > 0
    THEN
      COMPUTE AVERAGE ROUNDED = AMOUNT / COUNTER
      DISPLAY "Average = " AVERAGE UPON T
    ELSE
      DISPLAY "No calculation of average performed" UPON T
    END-IF
    STOP RUN.
```

The program calculates the average of numbers that have been input at the terminal. End criterion is the input of 00; invalid input is reported.

GO TO statement

Function

The GO TO statement is used to transfer control from one part of the Procedure Division to another.

Format 1 of the GO TO statement passes control to a specific procedure.

Format 2 of the GO TO statement passes control to one of a series of specified procedures, depending on the value of a data item.

Format 1

```
GO TO [ procedure-name ]
```

Syntax rules for format 1

1. If the GO TO statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement within that sequence.
2. A GO TO statement referenced by an ALTER statement must be preceded by a paragraph-name, and it must be the only statement in this paragraph (see "ALTER statement", page 254).
3. A GO TO statement without specification of procedure-name must be preceded by a paragraph-name, and it must be the only statement in this paragraph.

General rules

1. When a GO TO statement is executed, control is transferred to procedure-name.
2. If procedure-name is not specified, an ALTER statement, referring to this GO TO statement, must be executed prior to the execution of this GO TO statement.
3. A GO TO statement in which procedure-name is not specified must be supplied with a procedure-name destination by the ALTER statement. Otherwise, the program will terminate with error message 9142.

Example 3-81 for format 1

```
    GO TO END-ROUTINE.
        ...
END-ROUTINE.
    CLOSE CARD-FILE, PRINTER-FILE.
    STOP RUN.
```

In this example, the GO TO statement transfers control to the procedure named END-ROUTINE; the CLOSE statement is executed immediately after the GO TO statement.

Format 2

```
GO TO {procedure-name-1}...

      DEPENDING ON identifier
```

Syntax rules for format 2

1. identifier is the name of an elementary numeric data item described as an integer. The USAGE must be either DISPLAY, COMPUTATIONAL, COMPUTATIONAL-5, BINARY, COMPUTATIONAL-3 or PACKED-DECIMAL.
2. When a GO TO statement is executed, control is transferred to procedure-name-1..., depending on the value of the identifier, which may be 1, 2, ... n.

   If the identifier has any value other than 1, 2, ... n, no transfer of control takes place, and processing continues with the next statement in the normal sequence for execution (n represents the specified number of procedure-names).

General rule

The GO TO statement may specify up to 512 procedure-names.

Example 3-82 for format 2

```
77  A PICTURE 9.
       ...
    MOVE 3 TO A.
       ...
    GO TO X-PAR Y-PAR Z-PAR DEPENDING ON A.
```

Since the value of A is 3 when the GO TO statement is executed, control is transferred to Z-PAR, the third procedure-name in the series.

IF statement

Function

The IF statement causes a condition to be evaluated (see under "Conditions", page 218). The subsequent action of the program depends upon whether the condition is true or false.

Format 1

```
IF condition THEN statement-1 [ELSE statement-2]
    END-IF
```

Format 2

```
                    {statement-1    } [      {statement-2   }]
IF condition THEN   {                 } [ELSE  {                }]
                    {NEXT SENTENCE  }  [      {NEXT SENTENCE}]
```

Syntax rules

1. statement-1 and statement-2 may consist of one or more imperative statements and/or one conditional statement.
2. An IF statement with the NEXT SENTENCE phrase must not be contained within statement-1 or statement-2 of a format 1 IF statement which is immediately terminated with END-IF.

General rules

1. statement-1 and/or statement-2 may contain an IF statement. In this case the IF statement is said to be nested.
2. The scope of the IF statement may be terminated by any of the following:
   a) An END-IF phrase at the same level of nesting.
   b) A separator period.
   c) If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.

3. IF statements within IF statements may be considered as paired IF, ELSE, and END-IF combinations, proceeding from left to right. Thus, any ELSE or END-IF encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE or END-IF.
4. When an IF statement is executed, the following transfers of control occur:
   a) If the condition is true and statement-1 is specified, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-1, the ELSE phrase, if specified, is ignored and control passes to the end of the IF statement.
   b) If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.
   c) If the condition is false and statement-2 is specified, statement-1 or its surrogate NEXT SENTENCE is ignored, control is transferred to the first statement of statement-2, and execution continues according to the rules for each statement specified in >statement-2. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-2, control passes to the end of the IF statement.
   d) If the condition is false and the ELSE phrase is not specified, statement-1 is ignored and control passes to the end of the IF statement.
   e) If the condition is false and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored and control passes to the next executable sentence.

Example 3-83

```
IF A = B
THEN
  statement-1
END-IF
statement-2.
```

If A = B is true, statement-1 and statement-2 are executed.

If A = B is false, only statement-2 is executed.

Example 3-84

```
IF A = B
THEN
  statement-1
ELSE
  statement-2
END-IF
statement-3.
```

If A = B is true, statement-1 and statement-3 are executed.

If A = B is false, statement-2 and statement-3 are executed.

Example 3-85

```
IF A = B
THEN
  CONTINUE
ELSE
  statement-1
END-IF
statement-2.
```

If A = B is true, statement-2 is executed.

If A = B is false, statement-1 and statement-2 are executed.

Example 3-86

```
┌─> IF MALE
│
│   ┌─> IF MARRIED
│   │       ADD 1 TO MALES-MARRIED
│   ├─> ELSE
│   │   ┌─> IF DIVORCED
│   │   │       ADD 1 TO MALES-DIVORCED
│   │   ├─> ELSE
│   │   │       ADD 1 TO MALES-SINGLE
│   │   └─> END-IF
│   └─> END-IF
├─> ELSE
│       ADD 1 TO FEMALE
└─> END-IF
    next statement
```

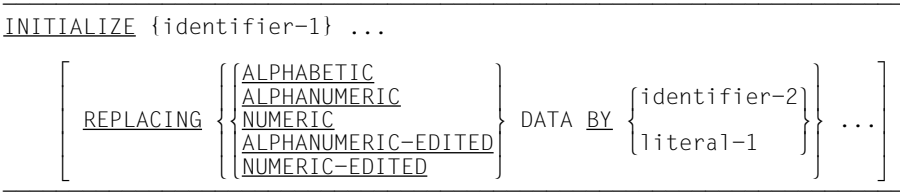
This is an example of the structure of a nested IF statement. The arrows are used to indicate the IF, ELSE and END-IF assignments.

INITIALIZE statement

Function

The INITIALIZE statement enables particular categories of data items to be preset with initial values, e.g. zeros for numeric items or blanks for alphanumeric items.

Format



Syntax rules

1. literal-1 and identifier-2 represent sending items; identifier-1 is the receiving item.
2. Each data category named in the REPLACING phrase must be allowed as the data category of a receiving item in a MOVE statement in which the sending item is represented by identifier-2 or literal-1.
3. Each data category can only be named once in the REPLACING phrase.
4. The data description entry of identifier-1 or a data item subordinate to identifier-1 must not contain the DEPENDING phrase of the OCCURS clause. identifier-1 must not be subordinate to a table that is defined with the DEPENDING phrase.
5. An index data item must not occur as the operand of an INITIALIZE statement.
6. The data description entry of identifier-1 must not contain the RENAMES clause.

General rules

1. The keyword following REPLACING identifies a data category (see page 68).
2. Regardless of whether identifier-1 represents an elementary item or a group item, all move operations are performed as though a sequence of MOVE statements had been written, each of them with an elementary item as receiving item.

   The following rules apply if the REPLACING phrase is specified:

   a) If identifier-1 is a group, each elementary item in this group is initialized only if it belongs to the data category named in the REPLACING phrase.

b) If identifier-1 represents an elementary item, this elementary item is initialized only if it belongs to the data category named in the REPLACING phrase.

- initialization is performed as follows:
  identifier-2 or literal-1 functions as the sending item of an implicit MOVE statement whose receiving item is an elementary item subordinate to identifier-1.
- except for the cases cited in general rules 3 and 4, all these elementary receiving items are initialized, including all table elements in the group.

3. Index data items and elementary FILLER items are skipped when the INITIALIZE statement is executed.
4. Every data item which is subordinate to a receiving item and whose data description entry contains the REDEFINES clause, or every data item subordinate to an item of this sort, is excluding from the initialization process. The receiving item itself, however, may contain the REDEFINES clause in its data description entry, or it may be subordinated to a data item with a REDEFINES clause.
5. If the REPLACING phrase is omitted, data items of the categories alphabetic, alphanumeric and alphanumeric-edited are initialized with blanks, those of the categories numeric and numeric-edited with zeros.
6. The elementary items represented by identifier-1 are initialized from left to right in the order in which they occur in the INITIALIZE statement. If identifier-1 is a group, the relevant elementary items within this sequence are initialized in the order in which they were defined in the group.
7. If identifier-1 and identifier-2 occupy the same storage space, this statement will produce unpredictable results when executed (see "Overlapping operands", page 245).

Example 3-87

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INIT1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION>
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  WAGE-RATE.
    02  SURNAME    PIC X(30).
    02  NAME       PIC X(30).
    02  ADDRESS    PIC X(30).
    02  DATE-OF-BIRTH.
        03  BDAY   PIC 99.
        03  MONTH  PIC 99.
        03  YEAR   PIC 99.
    02  HIRING-DATE.
        03  HDAY    PIC 99.
        03  MONTH   PIC 99.
        03  YEAR    PIC 99.
    02  NO-OF-HOURS   PIC 9(3).
    02  HOURLY-RATE   PIC 9(2)V99.
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    INITIALIZE WAGE-RATE.
    DISPLAY WAGE-RATE UPON T.
    STOP RUN.
```

The statement INITIALIZE WAGE-RATE means:

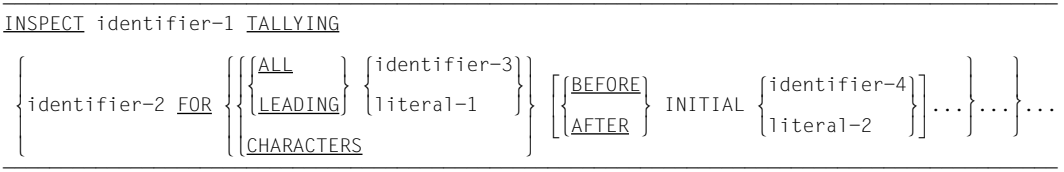
```
    MOVE SPACE TO SURNAME NAME ADDRESS
    MOVE ZERO TO BDAY OF DATE-OF-BIRTH
                 MONTH OF DATE-OF-BIRTH
                 YEAR OF DATE-OF-BIRTH
                 HDAY OF HIRING-DATE
                 MONTH OF HIRING-DATE
                 YEAR OF HIRING-DATE
                 NO-OF-HOURS, HOURLY-RATE
```

INSPECT statement

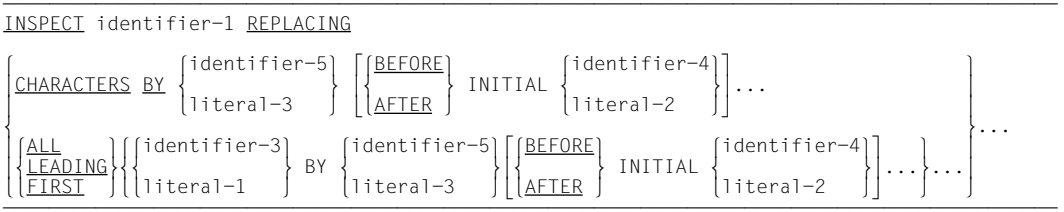
Function

The INSPECT statement enables single characters or groups of characters within a data item to be tallied, replaced, or tallied and replaced.

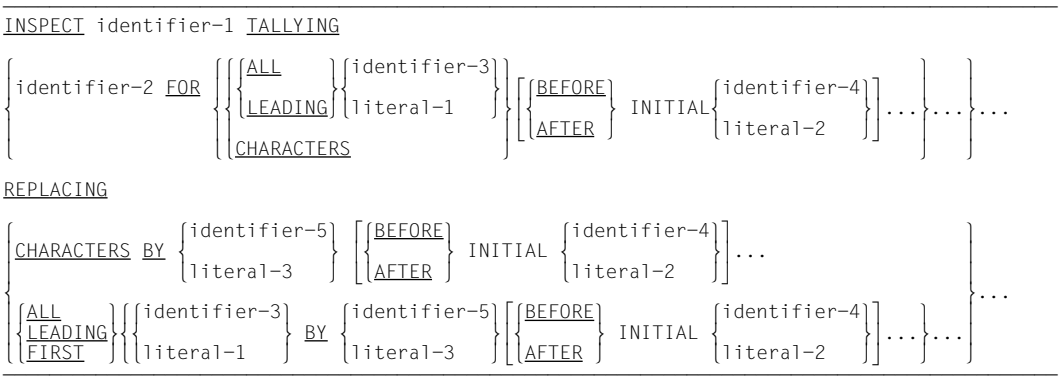
Format 1 (tallying)



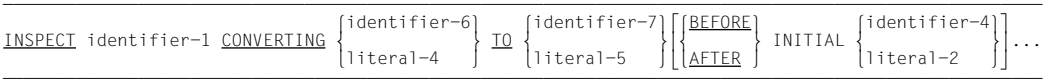
Format 2 (replacing)



Format 3 (tallying and replacing)



Format 4 (converting)



1. identifier-1 may reference either a group item or an elementary item, but it must be described (implicitly or explicitly) with USAGE IS DISPLAY.
2. identifier-3, ..., identifier-n may reference either a group item or an elementary item, but it must be described (implicitly or explicitly) with USAGE IS DISPLAY.
3. literal-1, literal-2, literal-3, literal-4, literal-5 must be either nonnumeric literals or figurative constants which do not begin with ALL.

   If literal-1, literal-2 or literal-4 is a figurative constant, it implicitly represents a single-character item.
4. Only one BEFORE and/or AFTER entry may be assigned to each individual ALL, LEADING, CHARACTERS, FIRST, or CONVERTING phrases.

Syntax rule for formats 1 and 3

5. identifier-2 must be an elementary numeric data item.

Syntax rules for formats 2 and 3

6. literal-3 or identifier-5 must be equal in size to literal-1 or identifier-3. When a figurative constant is used as literal-3, it is implicitly equal in size to literal-1 or identifier-3.
7. When CHARACTERS is used, literal-2, literal-3, identifier-4 and identifier-5 must each be one character in length.

Syntax rules for format 4

8. literal-5 or identifier-7 must be equal in size to literal-4 or identifier-6.

   When a figurative constant is used as literal-5, it is implicitly equal in size to literal-4 or identifier-6.
9. No character may appear more than once in literal-4 or identifier-6.

General rules for all formats

1. identifier-1 is processed from left to right regardless of its data class.
2. The contents of the data items indicated by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, identifier-7 are treated as follows:
   a) If any of these identifiers is described as alphabetic or alphanumeric, its contents will be treated as a character-string.
   b) If any of these identifiers is described as alphanumeric-edited, numeric-edited, or unsigned numeric, it will be treated as though it had been redefined as alphanumeric.
   c) If any of these identifiers is described as signed numeric, it will be treated as though it had been moved to an unsigned numeric data item of the same length (not counting sign position) and this item had then been redefined as alphanumeric (see "MOVE statement", page 294).
   d) If identifier-1 is described as a signed numeric data item, its original sign will be retained until the INSPECT statement has executed.
3. If any of these identifiers is indexed, the index value for these items will be calculated once only, namely immediately following execution of the INSPECT statement.
4. In general rules 5 to 15, everything that applies to literal-1, literal-2, literal-3, literal-4, or literal-5 applies equally to identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7.

General rules for formats 1, 2 and 3

5. While the contents of identifier-1 are being checked, each occurrence of literal-1 will be tallied (format 1) or all characters matching literal-1 will be replaced by literal-3 (format 2). If CHARACTERS is used, the characters in identifier-1 are tallied or replaced by literal-3 one at a time, depending on where the comparison operation is currently positioned.
6. The TALLYING or REPLACING operands are processed from left to right in the order in which they were specified in the INSPECT statement (comparison cycle). The first comparison cycle starts at the leftmost character in identifier-1.
7. If BEFORE or AFTER are omitted, the comparison operation to determine the occurrences of literal-1 in identifier-1 takes place as follows:
   a) The first literal-1 is compared to a series of contiguous characters within identifier-1 starting with the leftmost character, where the length of this series is equal to the length of literal-1. Only if literal-1 and this portion of identifier-1 are identical, character-for-character, does a match occur.

b) If no match between literal-1 and identifier-1 occurs, the comparison is repeated with each successive literal-1 until either a match is found or there is no next successive literal-1.

c) If no match whatsoever occurs between literal-1 and identifier-1, the character position within identifier-1 is shifted one position to the right and the comparison cycle starts again with the first literal-1.

d) Whenever a match occurs, the comparison cycle is terminated. Identifier-2 is incremented by 1 and/or the characters in identifier-1 which match literal-1 are replaced by literal-3. The character position within identifier-1 is then shifted to the right by the number of characters in literal-1 and the comparison cycle starts again with the first literal-1.

e) The comparison operation continues until the rightmost character in identifier-1 has either participated successfully in a match or is the character at which a comparison cycle begins.

f) If ALL is used, points a) to e) apply without restrictions.

If LEADING is used, the corresponding literal-1 is always involved in the first run-through of the comparison cycle. It only takes part is subsequent comparison cycles if the preceding cycle has produced a match with literal-1.

If CHARACTERS is used, an implicit single-character operand takes part in the comparison cycle as though it were entered as literal-1. However, no comparison with the contents of identifier-1 takes place; instead, this operand is always considered to match the character in identifier-1 at which the comparison cycle is currently positioned.

8. If BEFORE or AFTER is used, the following restrictions apply to point 7 above:

a) If BEFORE is used, the operation proceeds as follows: literal-1 or (if CHARACTERS has been specified) the implicit operand participates only in those comparison cycles which would have been performed if identifier-1 had ended with the character located immediately in front of the first occurrence of literal-2 within identifier-1.

If literal-2 does not occur at all within identifier-1, the comparison proceeds as if BEFORE had never been entered.

b) If AFTER is used, the same considerations apply as in a). That is, a search is made in identifier-1 for literal-2. If literal-2 is located, the record pointer is positioned to the character within identifier-1 which is located immediately to the right of literal-2. From this point on, literal-1 or (if CHARACTERS has been specified) the implicit operand participates in the subsequent comparison cycles.

If literal-2 does not occur at all within identifier-1, literal-1 or (if CHARACTERS has been specified) the implicit operand is not involved in a comparison cycle.

General rules for formats 1 and 3

9. The contents of identifier-2 are not initialized when the INSPECT statement is executed.

10. If identifier-1, identifier-3 or identifier-4 occupies the same memory area as identifier-2, the results of the INSPECT statement will be unpredictable, even when these identifiers are defined in the same data description entry (see "Overlapping operands", page 245).

General rules for formats 2 and 3

11. The mandatory words ALL, LEADING and FIRST are adjectives which apply to all subsequent BY phrases until the next adjective is entered.

12. If FIRST is used, literal-1 will be replaced by literal-3 within identifier-1 only at the position where it occurs for the first time. This rule applies to all successive FIRST phrases, regardless of the value of literal-1.

13. If identifier-3, identifier-4 or identifier-5 occupies the same memory area as identifier-1, the results of the INSPECT statement will be unpredictable, even when these identifiers are described in the same data description entry (see "Overlapping operands", page 245).

General rule for format 3

14. A format 3 INSPECT statement is executed as though it were two successive INSPECT statements referring to the same identifier-1, namely one format 1 INSPECT statement (with TALLYING phrase) and a format 2 INSPECT statement (with REPLACING phrase).
    The rules given for formats 1 and 2 apply accordingly. Subscripting associated with any identifier in the format 2 statement is evaluated only once before executing the format 1 statement.

General rules for format 4

15. A format-4 INSPECT statement is interpreted and executed as though it were a format-2 INSPECT statement containing a series of ALL phrases (one for each character in literal-4 or identifier-6) which refer to the same identifier-1.

    Thus, each character in literal-4 or identifier-6 and the corresponding character in literal-5 or identifier-7 is interpreted as though it were a self-contained literal-1 (or identifier-3) or literal-3 (identifier-5) in format 2.

    The unique assignment of characters from literal-4 (identifier-6) and literal-5 (identifier-7) results from their position within the data item.

16. If identifier-4, identifier-6 or identifier-7 occupies the same memory area as identifier-1, the results of the INSPECT statement will be unpredictable, even when the identifiers are defined in the same data description entry (see "Overlapping operands", page 245).

Example 3-88 for all formats

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  INSP.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 COUNTER1  PIC 99    VALUE ZEROES.
01 COUNTER2  PIC 99    VALUE 0.
01 COUNTER3  PIC 99    VALUE 0.
01 FIELD     PIC X(20) VALUE SPACES.
   ...

PROCEDURE DIVISION.
PROC SECTION.
COUNT-PAR.
    MOVE "BBYZYZBBYZAXBXBBX" TO FIELD.
    INSPECT FIELD TALLYING
       COUNTER1 FOR ALL "X" AFTER INITIAL "A"
       COUNTER2 FOR LEADING "YZ" AFTER INITIAL "BB"              (1)
       COUNTER3 FOR CHARACTERS BEFORE INITIAL "A".
    DISPLAY "After INSPECT" UPON T.
    DISPLAY "Counter1 = *" COUNTER1 "*" UPON T.
    DISPLAY "Counter2 = *" COUNTER2 "*" UPON T.
    DISPLAY "Counter3 = *" COUNTER3 "*" UPON T.
REPLACE-1.
    MOVE "MR. COBOLUSER" TO FIELD.
    DISPLAY "Before INSPECT" UPON T
    DISPLAY "Field = *" FIELD "*" UPON T.
    INSPECT FIELD REPLACING
       CHARACTERS BY "X" AFTER INITIAL "MR. "
                         BEFORE INITIAL "U".
    DISPLAY "After INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*" UPON T.
REPLACE-2.
    MOVE "ALGOL-PROGRAM" TO FIELD.
    DISPLAY "Before INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*" UPON T.
    INSPECT FIELD REPLACING
       ALL "A" BY "C" BEFORE INITIAL "P"
```

```
       ALL "L" BY "O" BEFORE INITIAL "G"
       ALL "G" BY "B" BEFORE INITIAL "P".
    DISPLAY "After INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*" UPON T.
REPLACE-3.
    MOVE "XXYZYZXXYZ-XYZXYZ" TO FIELD.
    DISPLAY "Before INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*".
    INSPECT FIELD REPLACING
       LEADING "YZ" BY "AB" BEFORE INITIAL "-"
                            AFTER INITIAL "XX"                    (2)
       FIRST "YZ" BY "CD" AFTER INITIAL "-".
    DISPLAY "After INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*" UPON T.
CONVERT.
    MOVE "CE#CGDHDEF-CD#F" TO FIELD.
    DISPLAY "Before INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*" UPON T.
    INSPECT FIELD CONVERTING                                      (3)
       "CDEF" TO "UVWU" AFTER "#"
                        BEFORE "-".
    DISPLAY "After INSPECT" UPON T.
    DISPLAY "Field = *" FIELD "*" UPON T.
ENDE.
    STOP RUN.
```

Result:

After INSPECT (1)

```
COUNTER1 = *03*
COUNTER2 = *02*
COUNTER3 = *06*
```

Before INSPECT	**After INSPECT**	
FIELD = *MR. COBOLUSER *	FIELD = *MR. XXXXXUSER *	
FIELD = *ALGOL-PROGRAM *	FIELD = *COBOL-PROGRAM *	(2)
FIELD = *XXYZYZXXYZ-XYZXYZ *	FIELD = *XXABABXXYZ-XCDXYZ *	
FIELD = *CE#CGDHDEF-CD#F *	FIELD = *CE#UGVHVWU-CD#F *	(3)

Explanation:

(1) In the case of COUNTER2, the instances of YZ underlined in the following string were tallied in FIELD:

BBYZYZBBYZAXBXBBX

In other words, with each of the underlined YZ pairs there was a match in the sense of general rule 7f. Hence, on both occasions the compare cycle resumed with the first literal-1, i.e. with COUNTER1.

The result is as follows:

COUNTER3 is not incremented for the underlined YZ pairs. Hence, only the following underlined characters are tallied:

BBYZYZBBYZAXBXBBX

Thus, COUNTER3 is equal to 6.

COUNTER3 would be equal to 10 following an INSPECT statement in which COUNTER3 was the first or the only literal-1.

(2) The replacement of leading YZ pairs by AB is caused by entering AFTER INITIAL "XX". BEFORE INITIAL "-" has no effect since no leading YZ pairs are present from the outset.

(3) This INSPECT statement has the same effect as the following statement:

```
INSPECT FIELD 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 "-".
```

MOVE statement

Function

The MOVE statement transfers data from one data item to one or more other data items.

Format 1 of the MOVE statement moves one data item to one or more other data items.

Format 2 of the MOVE statement moves corresponding data items from one group to another.

Format 1

```
MOVE {identifier-1} TO {identifier-2}...
     {literal     }
```

Syntax rules for format 1

1. identifier-1 or literal represents the sending item. The data in this area is moved to the receiving item specified by identifier-2.

   The same data is also moved to every additional receiving item that is specified (identifier-3...identifier-n), if such receiving items are supplied in the statement.

The rules specified below for identifier-2 also apply to all other receiving items:

2. An index data item must not appear as an operand of a MOVE statement.
3. Any subscripting or indexing associated with identifier-2 is evaluated immediately before the data is moved to the relevant data item.
4. Any necessary conversion of data from one form of internal representation to another takes place during legal moves, along with any editing specified for the receiving data item. This is described more fully in the general rules which follow.

General rules

See "General rules for both formats" (page 297).

Format 2

```
MOVE  {CORRESPONDING}  identifier-1 TO {identifier-2}...
      {CORR         }
```

Syntax rules for format 2

The following rules for identifier-2 also apply to any additional identifiers which follow identifier-2.

1. CORR is the abbreviation for CORRESPONDING.
2. identifier-1 specifies a group item which contains the elementary data items to be moved.
3. identifier-2 specifies a group item which contains the receiving items for the move.
4. The data items selected from the first operand (identifier-1) are moved to corresponding data items within the second operand (identifier-2). Data items from each group are considered to be corresponding when both data items have the same name and qualification up to, but not necessarily including, identifier-1 and identifier-2. The results of the MOVE CORRESPONDING statement are the same as if the user had specified each pair of corresponding identifiers in a separate MOVE statement of format 1 (for further rules see the CORRESPONDING phrase, page 239).
5. An index data item cannot appear as an operand in a MOVE statement.
6. Any subscripting or indexing associated with identifier-2 is evaluated immediately before the data is moved to the relevant data item.
7. Any necessary conversion of the data from one form of internal representation to another takes place during legal moves, along with any editing specified for the receiving data item. This is described more fully in the general rules which follow.

General rules for format 2

1. At least one of the data items in each pair of corresponding data items must be an elementary data item (note that the MOVE statement differs from arithmetic statements in this respect: in arithmetic statements using the CORRESPONDING phrase, both corresponding items must be elementary data items).
2. A data item that is subordinate to identifier-1 or identifier-2 and contains an OCCURS, REDEFINES, USAGE IS INDEX or RENAMES clause will be ignored. However, identifier-1 or identifier-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

For further **general rules** see "General rules for both formats" (page 297).

Example 3-89

for format 2

Procedure Division statement:

```
     MOVE CORRESPONDING EMPLOYEE-RECORD TO PAYROLL-CHECK.

Data Division entries:

01  EMPLOYEE-RECORD.                        01  PAYROLL-CHECK.
    02  EMPLOYEE-NUMBER.                        02  EMPLOYEE-NUMBER.
        03  PLANT-LOCATION...                       03  CLOCK-NUMBER...
        03  CLOCK-NUMBER.                           03  FILLER...
            04  SHIFT-CODE...                   02  DEDUCTIONS.
            04  CONTROL-NUMBER...                   03  FICA-RATE...
    02  WAGES.                                      03  WITHHOLDING-TAX...
        03  HOURS-WORKED...                         03  PERSONAL-LOANS...
        03  PAY-RATE...                         02  WAGES.
    02  FICA-RATE...                                03  HOURS-WORKED...
    02  DEDUCTIONS...                               03  PAY-RATE...
                                                 02  NET-PAY...
                                                 02  EMPLOYEE-NAME...
                                                     03  SHIFT-CODE...
```

According to the MOVE CORRESPONDING rules, the following data items would be moved:

Sending area	**Receiving area**
CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	CLOCK-NUMBER OF EMPLOYEE-NUMBER OF PAYROLL-CHECK
HOURS-WORKED OF WAGES OF EMPLOYEE-RECORD	HOURS-WORKED OF WAGES OF PAYROLL-CHECK
PAY-RATE OF WAGES OF EMPLOYEE-RECORD	PAY-RATE OF WAGES OF PAYROLL-CHECK
DEDUCTIONS OF EMPLOYEE-RECORD	DEDUCTIONS OF PAYROLL-CHECK

The following items are not moved, for the reasons stated:

Field	Reason
EMPLOYEE-NUMBER	Item is not elementary in either group.
PLANT-LOCATION OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Item does not appear in PAYROLL-CHECK.
SHIFT-CODE OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK.
CONTROL-NUMBER OF CLOCK-NUMBER OF EMPLOYEE-NUMBER OF EMPLOYEE-RECORD	Item does not appear in PAYROLL-CHECK.
WAGES	Item is not elementary in either group.
FICA-RATE OF EMPLOYEE-RECORD	Qualification is not identical in PAYROLL-CHECK.

General rules for both formats

1. Any move in which the sending and receiving items are both elementary items is an **elementary move**. Table 3-16 lists the classes to which elementary items belong.

Item	Category
Elementary data item	Numeric, alphabetic or alphanumeric, alphanumeric edited, numeric edited
Numeric literal	Numeric
Nonnumeric literal	Alphanumeric

Table 3-16: Categories of elementary items

   Any move in which either or both items are group items is a **group move**.

2. Table 3-17 lists all elementary and group moves, and indicates which moves are legal. The remaining general rules describe the execution of legal moves.

Sending item	Receiving item									
	GR	AL	AN	ED	BI	NE	ANE	ID	EF	IF
Group (GR)	Y	Y	Y	Y1	Y1	Y1	Y1	Y1	Y1	Y1
Alphabetic (AL)	Y	Y	Y	N	N	N	Y	N	N	N
Alphanumeric (AN)	Y	Y	Y	Y4	Y4	Y4	Y	Y4	Y4	Y4
External decimal (ED)	Y1	N	Y2	Y	Y	Y	Y2	Y	Y	Y
Binary (BI)	Y1	N	Y2	Y	Y	Y	Y2	Y	Y	Y
Numeric edited (NE)	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
Alphanumeric edited (ANE)	Y	Y	Y	N	N	N	Y	N	N	N
ZEROS (numeric or alphanumeric)	Y	N	Y	Y3	Y3	Y3	Y	Y3	Y3	Y3
SPACES (AN)	Y	Y	Y	N	N	N	Y	N	N	N
HIGH-VALUE, LOW-VALUE or QUOTES	Y	N	Y	N	N	N	Y	N	N	N
ALL literal	Y	Y	Y	Y5	Y5	Y5	Y	Y5	N	N
Numeric literal	Y1	Y	Y	Y5	Y5	Y5	Y	Y5	N	N
Nonnumeric literal	Y	Y	Y	Y5	Y5	Y5	Y	Y5	N	N
Internal decimal (ID)	Y1	N	Y2	Y	Y	Y	Y2	Y	Y	Y
Ext. fl. point (EF)	Y1	N	N	Y	Y	Y	N	Y	Y	Y
Int. fl. point (IF)	Y1	N	N	Y	Y	Y	N	Y	Y	Y
Fl. point literal	Y1	N	N	Y	Y	Y	N	Y	Y	Y

Table 3-17: Legality of types of elementary and group moves

Y denotes legal move, and N denotes illegal move
Y1 Move without conversion (like AN to AN)
Y2 Only if the decimal point is in the rightmost position (to the right of the least significant digit)
Y3 Numeric move
Y4 The alphanumeric item is treated as an ED item and may only contain numeric characters.
Y5 The literal must consist only of numeric characters
Y6 Numeric move with rounding

3. Any editing specified for the receiving data item takes place during an elementary move (some examples are given below).

4. There are two types of elementary moves: alphanumeric moves and numeric moves.

   a) An **alphanumeric move** takes place when an alphanumeric edited, alphanumeric, or alphabetic item is the receiving item. Table 3-18 lists the rules for this type of move.

	Sending item		**Receiving item**	
Rule	PICTURE IS	Value	PICTURE IS	Value
The characters are placed in the receiving item from left to right, unless the receiving item is specified as JUSTIFIED RIGHT. If the receiving item is not completely filled by the data being moved, the remaining positions are filled with spaces.	XXX	M8N	XXXXX	M8N␣␣
If the sending item is longer than the receiving item, the move is terminated as soon as the receiving item is filled. Excess characters are truncated. If the receiving item is described as JUSTIFIED RIGHT, truncation is performed on the left.	AAAAAA	XYZABC	AAA AAA JUST RIGHT	XYZ ABC
If the sending item has an operational sign, its absolute value is moved.	S999	−333	XXX	333

Table 3-18: Rules for alphanumeric moves

   b) A **numeric move** takes place when there is a move from a numeric item or numeric edited item to a numeric item or numeric edited item. Table 3-19 lists the rules for this type of move. When there is a move from a numeric edited item to a numeric item, those characters in the numeric edited item which represent signs or digits are taken into consideration. The move of a numeric edited item to a numeric edited item is treated as a move of a numeric edited item to a numeric item, followed by the move of a numeric item to a numeric edited item.

Rule	Sending item *)		Receiving item *)	
	PICTURE IS	Value	PICTURE IS	Value
The items are aligned on their decimal points. If the sending item is larger than the receiving item, truncation may occur at either end. If the receiving item is larger than the sending item, unused character positions are either zero-filled or are replaced as specified by editing characters.	9V999	8&765	V99	V76
	9V9	1&2	99V99	01V20
	99V99	67&89	$$$99.99	␣␣$67.89
If the sending item has an operational sign and the receiving item has no operational sign, then the absolute value of the sending item is moved.	AAAAAA	XYZABC	AAA AAA JUST RIGHT	XYZ ABC
If no assumed or actual decimal points are specified, data is right-justified in the receiving item.	S999	−333	XXX	333

Table 3-19: Rules for numeric moves

*) V is the assumed decimal point. Any necessary conversion of data from one form of internal representation to another takes place during the move.

5. Any move involving group items is treated exactly as though it were an alphanumeric elementary move, except that there is no conversion of data.
   If a group item contains a subordinate item whose description includes an OCCURS...DEPENDING ON clause, then the group item is said to be of variable length.

   If such a variable-length group item is involved in a move, only the current active length of the group item is used in the move. The length is determined by the DEPENDING ON phrase. This applies to both sending and receiving item.

6. If the sending or receiving operands of a MOVE statement share the same storage area (i.e. if the operands "overlap"), then execution of that statement produces unpredictable results.

MULTIPLY statement

Function

The MULTIPLY statement is used to perform multiplication of two numeric operands and store the result.

Format 1 of the MULTIPLY statement stores the products in the specified multiplier.

Format 2 of the MULTIPLY statement uses the GIVING phrase.

Format 1

```
MULTIPLY {identifier-1} BY {identifier-2 [ROUNDED]}...
         {literal-1   }

    [ON SIZE ERROR imperative-statement-1]

    [NOT ON SIZE ERROR imperative-statement-2]

    [END-MULTIPLY]
```

Syntax rules for format 1

1. Each identifier must refer to a numeric elementary item.
2. The value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The multiplication product replaces the current value of identifier-2.
3. The maximum size of the product after decimal point alignment is 18 decimal digits.
4. END-MULTIPLY delimits the scope of the MULTIPLY statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the ROUNDED, (NOT) ON SIZE ERROR, and GIVING phrases are discussed.

Example 3-90

for format 1

Statement	PICTURE of result item	Calculation
`MULTIPLY A BY B`	`999`	`A * B` stored in `B` as nnn

Format 2

```
MULTIPLY {identifier-1} BY {identifier-2}
         {literal-1   }    {literal-2   }

         GIVING {identifier-3 [ROUNDED]}...

         [ON SIZE ERROR imperative-statement-1]

         [NOT ON SIZE ERROR imperative-statement-2]

         [END-MULTIPLY]
```

Syntax rules for format 2

1. Each identifier preceding the word GIVING must refer to a numeric elementary item.
2. identifier-3... may refer to a numeric elementary item or a numeric edited elementary item.
3. The value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2, and the product is stored in identifier-3 (the same applies to additional receiving items).
4. The maximum size of the product after decimal point alignment is 18 decimal digits.
5. END-MULTIPLY delimits the scope of the MULTIPLY statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the ROUNDED, (NOT) ON SIZE ERROR, and GIVING phrases are discussed.

Example 3-91

for format 2

Statement	PICTURE of result item (C)	Calculation
`MULTIPLY A BY B GIVING C`	`9(5)`	`A * B` stored in `C` as nnnnn

PERFORM statement

Function

The PERFORM statement is used to execute one or more procedures or a set of statements.

Format 1 of the PERFORM statement executes the specified procedures statements one time.

Format 2 of the PERFORM statement executes the specified procedures or statements a specified number of times.

Format 3 of the PERFORM statement executes the specified procedures or statements until a specified condition is true.

Format 4 of the PERFORM statement changes the values of one or more identifiers or index names in ascending or descending order, and executes a series of procedures or the specified statements one or more times, based on this action.

Format 1

```
"out-of-line"

PERFORM  procedure-name-1  [{THRU     }procedure-name-2]
                           {        }
                           {THROUGH }

"in-line"

PERFORM  [imperative-statement END-PERFORM]
```

Syntax rules for format 1

1. The words THROUGH and THRU are equivalent.
2. Where both procedure-name-1 and procedure-name-2 are specified, and either is the name of a procedure within the declaratives portion of the Procedure Division, then both procedure-names must be within the same declarative section.

General rules for format 1

1. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.
2. An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.
3. When a PERFORM statement is executed, control is transferred to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made as follows:
   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.
   b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.
   c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.
   d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.
   e) If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.
4. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.
5. The statements within the range of the PERFORM statement are executed once, and control returns to the statement following the PERFORM statement.
6. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all

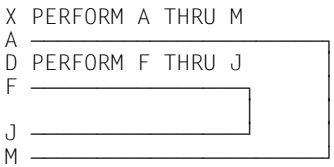
statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

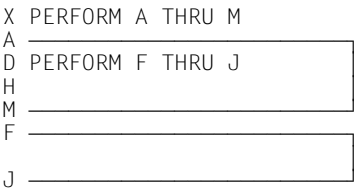
7. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:

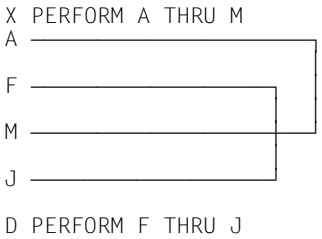
   a) That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and

   b) The EXIT PROGRAM statement is within the range of the PERFORM statement.

8. If the range of a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements must not have a common exit. See the following illustrations for examples of legal PERFORM constructs:







9. The following information on segmentation is relevant here (for further details, refer to "Segmentation", page 635ff):

   a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may contain within its range only the following procedures:

      – either sections all having a segment number less than 50, or

      – sections which are wholly contained in a single independent segment whose segment number is greater than 49.

b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may contain within its range only the following procedures:

   – either sections which all have the same segment number as the section containing the PERFORM statement or

   – sections all having a segment number less than 50.

c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 3-92

for format 1

```
    PERFORM X-PAR.
    ADD A TO B.
```

Let X-PAR be a paragraph-name. In this case, all statements in the paragraph named X-PAR are executed, and control is then returned to the ADD statement following the PERFORM statement.

Example 3-93

```
        ...
    PERFORM X1-PAR THRU X3-PAR.
        ...
  X-KAP SECTION.
X1-PAR.
        ...
X2-PAR.
        ...
X3-PAR.
        ...
Y-KAP SECTION.
        ...
```

The PERFORM statement has the effect that all statements in the paragraphs named X1-PAR, X2-PAR and X3-PAR are executed.

Example 3-94

The same effect would result from the execution of the statement:

```
   PERFORM X-KAP.
```

Format 2

```
"out-of-line"

PERFORM  procedure-name-1  [ { THRU      } procedure-name-2 ]  { identifier-1 }  TIMES
                           { THROUGH   }                     { integer-1    }

"in-line"

PERFORM  { identifier-1 }  TIMES  imperative-statement  END-PERFORM
         { integer-1    }
```

Syntax rules for format 2

1. identifier-1 specifies an elementary numeric data item described as an integer.
2. The words THROUGH and THRU are equivalent.
3. Where both procedure-name-1 and procedure-name-2 are specified, and either is the name of a procedure in the declaratives portion of the Procedure Division, then both procedure-names must be within the same declarative section.

General rules for format 2

1. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.
2. An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.
3. When a PERFORM statement is executed, control is transferred to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made as follows:
   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.
   b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.
   c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

e) If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

4. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.

5. The specified set of statements is performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. Following the execution of the specified set of statements, control is transferred to the end of the PERFORM statement.

6. If at the time of the execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the end of the PERFORM statement.

7. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

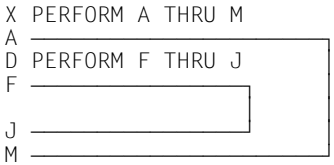
8. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:

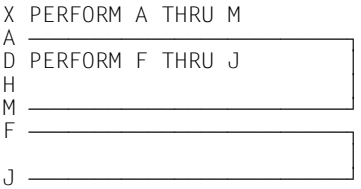
   a) That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and

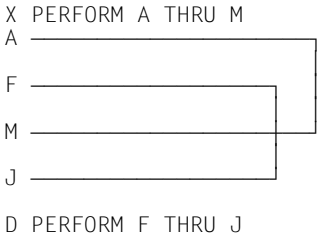
   b) The EXIT PROGRAM statement is within the range of the PERFORM statement.

9. If the range of a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass

to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements must not have a common exit. See the following illustrations for examples of legal PERFORM constructs:







10. The following information on segmentation is relevant here (for further details, refer to "Segmentation", page 635ff):

    a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may contain within its range only the following procedures:

       – either sections all having a segment number less than 50, or
       – sections which are wholly contained in a single independent segment whose segment number is greater than 49.

    b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may contain within its range only the following procedures:
       – either sections which all have the same segment number as the section containing the PERFORM statement or
       – sections all having a segment number less than 50.

    c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 3-95

for format 2

```
   PERFORM X-PAR 5 TIMES.
```

All statements in the paragraph named X-PAR are performed five times. Control then passes to the statement following the PERFORM statement.

Example 3-96

for format 2

```
...
77 A PICTURE 9.
    ...
    MOVE 3 TO A.
    ...
    PERFORM X-PAR A TIMES.
    ...
X-PAR.
    ...
    ADD 1 TO A.
Y-PAR.
    ...
```

Since the value of A is 3 when the PERFORM statement is executed, the paragraph named X-PAR is executed three times.

The reference to A within X-PAR has no effect on the PERFORM statement.

Format 3

```
"out-of-line"

PERFORM procedure-name-1 [ [ { THRU      } procedure-name-2 ] ]
                         [ { THROUGH     }                  ]

   [ WITH TEST { BEFORE } ] UNTIL condition-1
               { AFTER  }

"in-line"

PERFORM    [ WITH TEST { BEFORE } ] UNTIL condition-1
                       { AFTER  }

   imperative-statement END-PERFORM
```

Syntax rules for format 3

1. The words THROUGH and THRU are equivalent.
2. When both procedure-name-1 and procedure-name-2 are specified, and either is the name of a procedure within the declaratives portion of the Procedure Division, then both procedure-names must be within the same declarative section.
3. If neither TEST BEFORE nor TEST AFTER is specified, TEST BEFORE is assumed.

General rules for format 3

1. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.
2. An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.
3. When a PERFORM statement is executed, control is transferred to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made as follows:
   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.

b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

e) If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

4. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.

5. The statements within the range of the PERFORM statement are executed once, and control returns to the statement following the PERFORM statement.

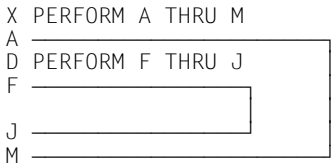
6. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control to the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

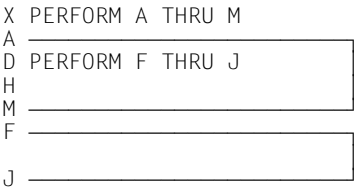
7. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:

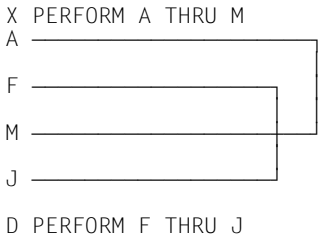
   a) That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and

   b) The EXIT PROGRAM statement is within the range of the PERFORM statement.

8. If the range of a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements must not have a common exit. See the following illustrations for examples of legal PERFORM constructs:







9. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, and the TEST BEFORE phrase is specified or implied, no transfer to procedure-name-1 takes place, and control is passed to the end of the PERFORM statement.
10. If the TEST AFTER phrase is specified, the PERFORM statement functions as if the TEST BEFORE phrase were specified except that the condition is tested after the specified set of statements has been executed.
11. Any subscripting or reference modification associated with the operands specified in condition-1 is evaluated each time the condition is tested.
12. The following information on segmentation is relevant here (for further details, refer to "Segmentation", page 635ff):

    a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may contain within its range only the following procedures:

       – either sections all having a segment number less than 50, or

- sections which are wholly contained in a single independent segment whose segment number is greater than 49.

b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may contain within its range only the following procedures:

- either sections which all have the same segment number as the section containing the PERFORM statement or
- sections all having a segment number less than 50.

c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 3-97

for format 3

```
    PERFORM X-PAR UNTIL A GREATER THAN 3.
    ...
X-PAR.
    ...
    COMPUTE A = A + 1.
    ...
Y-PAR..
```

Assume A = 1, when the PERFORM statement is initiated. In this case, the statements in the paragraph named X-PAR are performed three times:

The first time X-PAR is performed, A is set to 2.

The second time X-PAR is performed, A is set to 3.

The third time X-PAR is performed, A is set to 4.

Since 4 is greater than 3, the condition specified in the PERFORM statement is true. Thus, control passes to the statement which follows the PERFORM statement.

Format 4

```
"out-of-line"
PERFORM  procedure-name-1 [ [ { THRU      } procedure-name-2 ] [ WITH TEST { BEFORE } ] ]
                          [ { THROUGH }                      ]             { AFTER  }

    VARYING { index-1      } FROM { index-2      } BY { literal-2      } UNTIL condition-1
            { identifier-2 }      { literal-1    }    { identifier-4 }
                                  { identifier-3 }

   [ AFTER  { index-3      } FROM { index-4      } BY { literal-4      } UNTIL condition-2 ]...
            { identifier-5 }      { literal-3    }    { identifier-7 }
                                  { identifier-6 }

"in-line"

PERFORM   [ WITH TEST { BEFORE } ]
          [           { AFTER  } ]

  VARYING { index-1      } FROM { index-2      } BY { literal-2      } UNTIL condition-1
          { identifier-2 }      { literal-1    }    { identifier-4 }
                                { identifier-3 }

  imperative-statement  END-PERFORM
```

Syntax rules for format 4

1. The words THROUGH and THRU are equivalent.
2. When both procedure-name-1 and procedure-name-2 are specified, and either is the name of a procedure within the declaratives portion of the Procedure Division, then both procedure-names must be within the same declarative section.
3. If neither TEST BEFORE nor TEST AFTER is specified, TEST BEFORE is assumed.
4. Each identifier represents a numeric item described as an integer. However, the compiler allows each identifier to be described as a noninteger numeric item.
5. Each literal must be numeric.
6. The literal in the associated BY phrase must be a non-zero integer.
7. If an index-name is specified in the VARYING or AFTER phrase, then:
   a) The identifier in the associated FROM and BY phrases must reference an integer data item.
   b) The literal in the associated FROM phrase must be a positive integer.
   c) The literal in the associated BY phrase must be a non-zero integer.
8. If an index-name is specified in the FROM phrase, then:
   a) The identifier in the associated VARYING or AFTER phrase must reference an integer data item.
   b) The identifier in the associated BY phrase must reference an integer data item.

c) The literal in the associated BY phrase must be an integer.

9. condition-1, condition-2 may be any conditional expression.
10. A maximum of 6 AFTER phrases are permitted in the out-of-line PERFORM statement.

General rules for format 4

1. The data items referenced by identifier-4 and identifier-7 must not have a zero value.
2. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, the data item referenced by the identifier must have a positive value.
3. Format 4 of the PERFORM statement is used to increment or decrement the values of one or more identifiers or index-names in a specific manner while the PERFORM statement is executing. Execution depends on the number of identifiers or index-names that are varied. The following rules describe what happens when one, two and three identifiers or index-names are varied.
4. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.
5. An in-line PERFORM statement functions according to the following general rules for an otherwise identical out-of-line PERFORM statement, with the exception that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specially qualified by the word in-line or out-of-line, all the general rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.
6. When a PERFORM statement is executed, control is passed to the first statement of the procedure named by procedure-name-1. An implicit return to the next executable statement following the PERFORM statement is made as follows:

   a) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return takes place after the last statement of procedure-name-1.

   b) If procedure-name-1 is a section-name, and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

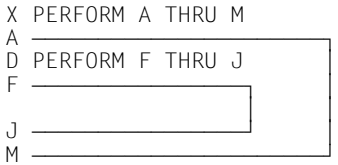
   c) If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

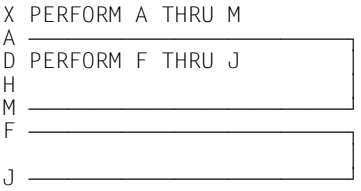
   d) If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

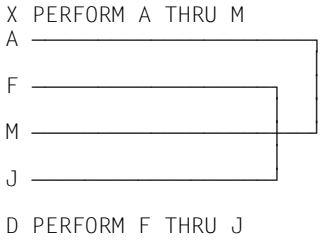
   e) If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

7. There is no necessary relationship between procedure-name-1 and procedure-name-2, except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. Even GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to which all of these paths must lead.
8. In the following discussion, every reference to identifier as the object of the VARYING, AFTER, and FROM (current value) phrases also refers to index-names.
   a) If index-name-1 or index-name-3 is specified, the value of the associated index at the beginning of the PERFORM statement must be set to an occurrence number of an element in the table.
   b) If index-name-2 or index-name-4 is specified, the value of the data item referenced by identifier-2 or identifier-5 at the beginning of the PERFORM statement must be equal to an occurrence number of an element in a table associated with index-name-2 or index-name-4.
   c) Subsequent augmentation, as described below, of index-name-1 or index-name-3 must not result in the associated index being set to a value outside the range of the table associated with index-name-1 or index-name-3; except that, at the completion of the PERFORM statement, the index associated with index-name-1 may contain a value that is outside the range of the associated table by one increment or decrement value.
   d) If identifier-2 or identifier-5 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is set or augmented.
   e) If identifier-3, identifier-4, identifier-6, or identifier-7 is subscripted, the subscripts are evaluated each time the content of the data item referenced by the identifier is used in a setting or augmenting operation.
9. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement are not considered to be part of the range of the PERFORM statement when:
   a) That EXIT PROGRAM statement is specified in the same program in which the PERFORM statement is specified, and
   b) The EXIT PROGRAM statement is within the range of the PERFORM statement.
10. If the range of a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass

to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements must not have a common exit. See the following illustrations for examples of legal PERFORM constructs:







11. If the TEST BEFORE phrase is specified or implied:

    When the data item referenced by **one** identifier is varied:

    – The content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of initial execution of the PERFORM statement.

    – Then, if the condition of the UNTIL phrase is false, the specified set of statements is executed once.

    – The value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and condition-1 is evaluated again .

    – This cycle continues until this condition is true, at which point control is transferred to the end of the PERFORM statement.

    – If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the end of the PERFORM statement.

    When the data items referenced by **two** identifiers are varied:

    – At the start of execution of the PERFORM statement, the content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3.

    – Then the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6.

- After the contents of the data items referenced by the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the end of the PERFORM statement.
- If condition-1 is false, condition-2 is evaluated.
- If condition-2 is false, the specified set of statements is executed once.
- Then the content of the data item referenced by identifier-5 is augmented by literal-4 or the content of the data item referenced by identifier-7 and condition-2 is evaluated again.
- This cycle of evaluation and augmentation continues until condition-2 is true.
- When condition-2 is true, the content of the data item referenced by identifier-2 is augmented by literal-2 or the content of the data item referenced by identifier-4, and the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6.
- condition-1 is reevaluated.
- The PERFORM statement is completed if condition-1 is true; if not, the cycle continues until condition-1 is true.

At the termination of the PERFORM statement, the data item referenced by identifier-5 contains literal-3 or the current value of the data item referenced by identifier-6. The data item referenced by identifier-2 contains a value that exceeds the last used setting by one increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case the data item referenced by identifier-2 contains literal-1 or the current value of the data item referenced by identifier-3.

12. If the TEST AFTER phrase is specified or implied:

    When the data item referenced by **one** identifier is varied:

    - The content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3 at the point of execution of the PERFORM statement.
    - Then the specified set of statements is executed once and condition-1 of the UNTIL phrase is tested.
    - If the condition is false, the value of the data item referenced by identifier-2 is augmented by the specified increment or decrement value (literal-2 or the value of the data item referenced by identifier-4) and the specified set of statements is executed again.
    - The cycle continues until condition-1 is tested and found to be true, at which point control is transferred to the end of the PERFORM statement.

When the data items referenced by **two** identifiers are varied:

- The content of the data item referenced by identifier-2 is set to literal-1 or the current value of the data item referenced by identifier-3.
- Then the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6.
- The specified set of statements is then executed.
- Condition-2 is then evaluated.
- If condition-2 is false, the content of the data item referenced by identifier-5 is augmented by literal-4 or the content of data item referenced by identifier-7 and the specified set of statements is again executed.
- The cycle continues until condition-2 is again evaluated and found to be true.
- At this time, condition-1 is evaluated.
- If condition-1 is false, the content of the data item referenced by identifier-2 is augmented by literal-2 or the content of the data item referenced by identifier-4, and the content of the data item referenced by identifier-5 is set to literal-3 or the current value of the data item referenced by identifier-6.
- The specified set of statements is then executed again.
- This cycle continues until condition-1 is again evaluated and found to be true, at which time control is transferred to the end of the PERFORM statement.

After the completion of the PERFORM statement, each data item varied by an AFTER or VARYING phrase contains the same value it contained at the end of the most recent execution of the specified set of statements.

13. During the execution of the specified set of statements associated with the PERFORM statement, any change to the VARYING variable (the data item referenced by identifier-2 and index-name-1), the BY variable (the data item referenced by identifier-4), the AFTER variable (the data item referenced by identifier-5 and index-name-3), or the FROM variable (the data item referenced by identifier-3 and index-name-2) will be taken into consideration and will affect the operation of the PERFORM statement.

    When the data items referenced by two identifiers are varied, the data item referenced by identifier-5 goes through a complete cycle (FROM, BY, UNTIL) each time the content of the data item referenced by identifier-2 is varied. When the contents of three or more data items referenced by identifiers are varied, the mechanism is the same as for two identifiers except that the data item being varied by each AFTER phrase goes through a complete cycle each time the data item being varied by the preceding AFTER phrase is augmented.

14. The following information on segmentation is relevant here (for further details, refer to "Segmentation", page 635ff):

    a) If a PERFORM statement appears in a section whose segment number is less than the value specified in the SEGMENT-LIMIT clause, then it may contain within its range only the following procedures:

       – either sections which all have a segment number less than 50, or

       – sections which are wholly contained in a single independent segment whose segment number is greater than 49.

    b) If a PERFORM statement appears in a section whose segment number is greater than 49, then it may contain within its range only the following procedures:

       – either sections which all have the same segment number as the section containing the PERFORM statement or

       – sections all having a segment number less than 50.

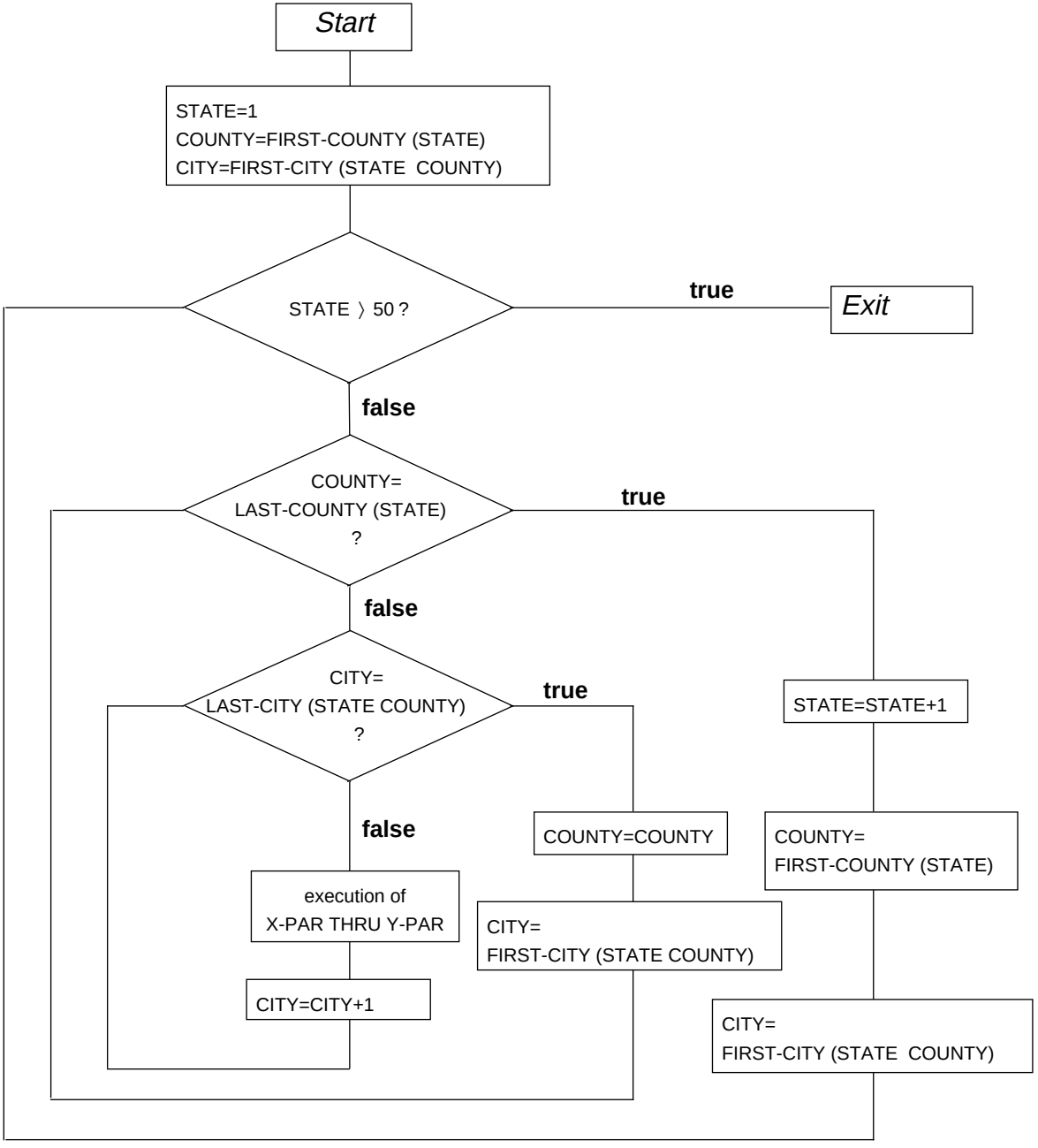
    c) If a procedure-name in a segment with a segment number greater than 49 is referenced by a PERFORM statement in a segment with a different segment number, then the referenced segment is made available in its initial state (that is, any GO TO statements in this segment which have been modified by the execution of ALTER statements will be reset to their original values).

Example 3-98

for format 4

```
    ...

01  STATES.
    02  COUNTIES OCCURS 51 INDEXED STATE.
        03  FIRST-COUNTY     PIC 9(3).
        03  LAST-COUNTY      PIC 9(3).
        03  CITIES OCCURS 100 INDEXED COUNTIES.
            04  FIRST-CITY   PIC 9(3).
            04  LAST-CITY    PIC 9(3).
01  CITY                     PIC 9(3).

     ...

PROCEDURE DIVISION.
K1.
    PERFORM X-PAR THRU Y-PAR
        VARYING STATE FROM 1 BY 1
            UNTIL STATE IS GREATER THAN 50;
        AFTER COUNTY FROM FIRST-COUNTY (STATE) BY 1
            UNTIL COUNTY IS EQUAL TO LAST-COUNTY (STATE)
        AFTER CITY FROM FIRST-CITY (STATE COUNTY)
            UNTIL CITY IS EQUAL TO LAST-CITY (STATE COUNTY)
     . . .
```

Flowchart:


Start
STATE=1
COUNTY=FIRST-COUNTY (STATE)
CITY=FIRST-CITY (STATE COUNTY)
STATE 〉 50 ?
true
Exit
false
COUNTY=
LAST-COUNTY (STATE)
?
true
false
CITY=
LAST-CITY (STATE COUNTY)
?
true
STATE=STATE+1
false
COUNTY=COUNTY
COUNTY=
FIRST-COUNTY (STATE)
execution of
X-PAR THRU Y-PAR
CITY=
FIRST-CITY (STATE COUNTY)
CITY=CITY+1
CITY=
FIRST-CITY (STATE COUNTY)

Example 3-99

for format 3 and 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.
COMPUTATION.
    MOVE 0 TO Z.
    DISPLAY "Enter single-digit number as upper bound N" 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 "Result =" Z UPON T.
FIN.
    STOP RUN.
```

The program calculates the nth sum of the cube of an integer K. Only after the COMPUTE and ADD statements are executed is there a test to see whether the termination condition is satisfied.

If TEST BEFORE is specified, the termination condition is tested first. If K <= N, the in-line statements are executed. If K > N, the PERFORM statement is terminated with END-PERFORM.

SEARCH statement

Function

The SEARCH statement is used to search a table for a table element that satisfies a specified condition, and to adjust the value of the associated index to indicate that table element, i.e. to set it to the corresponding occurrence number.

Format 1 is used to perform a serial search of a table. The search for identifier-1 begins at the current value of the index assigned to identifier-1.

Format 2 is used to perform a binary search of a table.

Format 1

```
SEARCH identifier-1 [VARYING {index-1     }]
                             {identifier-2}

       [AT END imperative-statement-1]

       {WHEN condition-1 {imperative-statement-2}}...
                         {NEXT SENTENCE         }

    [END-SEARCH]
```

Syntax rules for format 1

1. identifier-1 specifies the table to be searched.
2. The data description entry of identifier-1 must include an OCCURS clause with an INDEXED BY phrase.
3. identifier-1 must not be subscripted or indexed or subjected to reference modification.
4. index-1 or identifier-2 specifies an item whose value is to be varied during execution of the SEARCH statement.
5. index-1 may be one of the indices for identifier-1, or an index for another table entry.
6. identifier-2 must therefore be described as an index data item (USAGE IS INDEX), or it must be a fixed-point numeric item described as an integer.
7. The AT END phrase specifies a statement which is to be executed if the search is unsuccessful.
8. Each condition (condition-1, condition-2,...) must be a valid condition (see "Conditions", page 218).

9. condition-1... specify the conditions to be satisfied during the execution of the SEARCH statement.
10. imperative-statement-2... or NEXT SENTENCE specifies an action to be taken when the associated WHEN condition is satisfied: control passes to the imperative-statement or to the next sentence (that is, the statement following the SEARCH statement), depending on the option specified.
11. A serial search of a table begins at the table element pointed to by the index associated with identifier-1.
12. If, at the start of a SEARCH statement, the value of the index associated with identifier-1 is greater than the highest permissible occurrence number for identifier-1, the search will terminate immediately. If the AT END phrase is specified, imperative-statement-1 is executed. If this phrase is omitted, processing continues with the next statement.
13. If, at the start of a SEARCH statement, the value of the index associated with identifier-1 corresponds to a valid occurrence number for identifier-1, the serial search takes place as follows:
    a) The WHEN conditions are evaluated in the order in which they are written.
    b) If none of the conditions is satisfied, the index-name for identifier-1 is incremented to refer to the next occurrence of a table element; and step a) is repeated, unless the new value of the index corresponds to an occurrence number outside the valid range, in which case step d) is performed.
    c) If one of the WHEN conditions is satisfied, the search terminates immediately. The index points to the table element that satisfied the condition. The imperative statement associated with that condition is executed.
    d) If the end of the table is reached without the WHEN condition being satisfied, the search terminates. If AT END is specified, imperative-statement-1 is executed. If this phrase is omitted, control passes to the next sentence.
14. When identifier-1 is a data item subordinate to a data item that contains an OCCURS clause, then multi-dimensional tables can be searched. In this case, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Execution of a SEARCH statement modifies only the setting of the index associated with identifier-1 (and, if present, of index-1 or identifier-2). Therefore, in order to search a two- or three-dimensional table, a SEARCH statement must be executed for each possible value of the superordinate index.
15. Before the SEARCH statements are executed, the corresponding indices must be preset with the required values by means of SET statements or with PERFORM VARYING (cf. example 3-101).

16. If, in the AT END phrase and the WHEN conditions, none of the imperative-statements specified terminates with a GO TO statement, then control will pass to the next statement after the imperative statement is executed.
17. If the VARYING index-1 phrase is specified, the following takes place:
    a) If index-1 is one of the indices for identifier-1, index-1 is used for the search. No other indices are incremented.
    b) If index-1 is an index for another table entry, the first, or only, index associated with identifier-1 is used for the search. When the index associated with identifier-1 is incremented, index-1 is simultaneously incremented to represent the next element in its table.
18. If the VARYING identifier-2 phrase is specified, the following actions take place:
    a) The first, or only, index associated with identifier-1 is used for the search.
    b) When the index associated with identifier-1 is incremented, identifier-2 is simultaneously incremented.
    c) If identifier-2 is a numeric item, it is incremented by 1.
    d) If identifier-2 is an index data item, it is incremented by a value equal to that used to increment the index associated with identifier-1.

    If the VARYING phrase is not specified, the first, or only, index associated with identifier-1 (i.e. defined in the INDEXED BY phrase of the data description entry of identifier-1) will be used for the search.
19. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.
20. The scope of a SEARCH statement may be terminated by any of the following:
    a) An END-SEARCH phrase at the same level of nesting.
    b) A separator period.
    c) An ELSE or END-IF phrase associated with a previous IF statement.

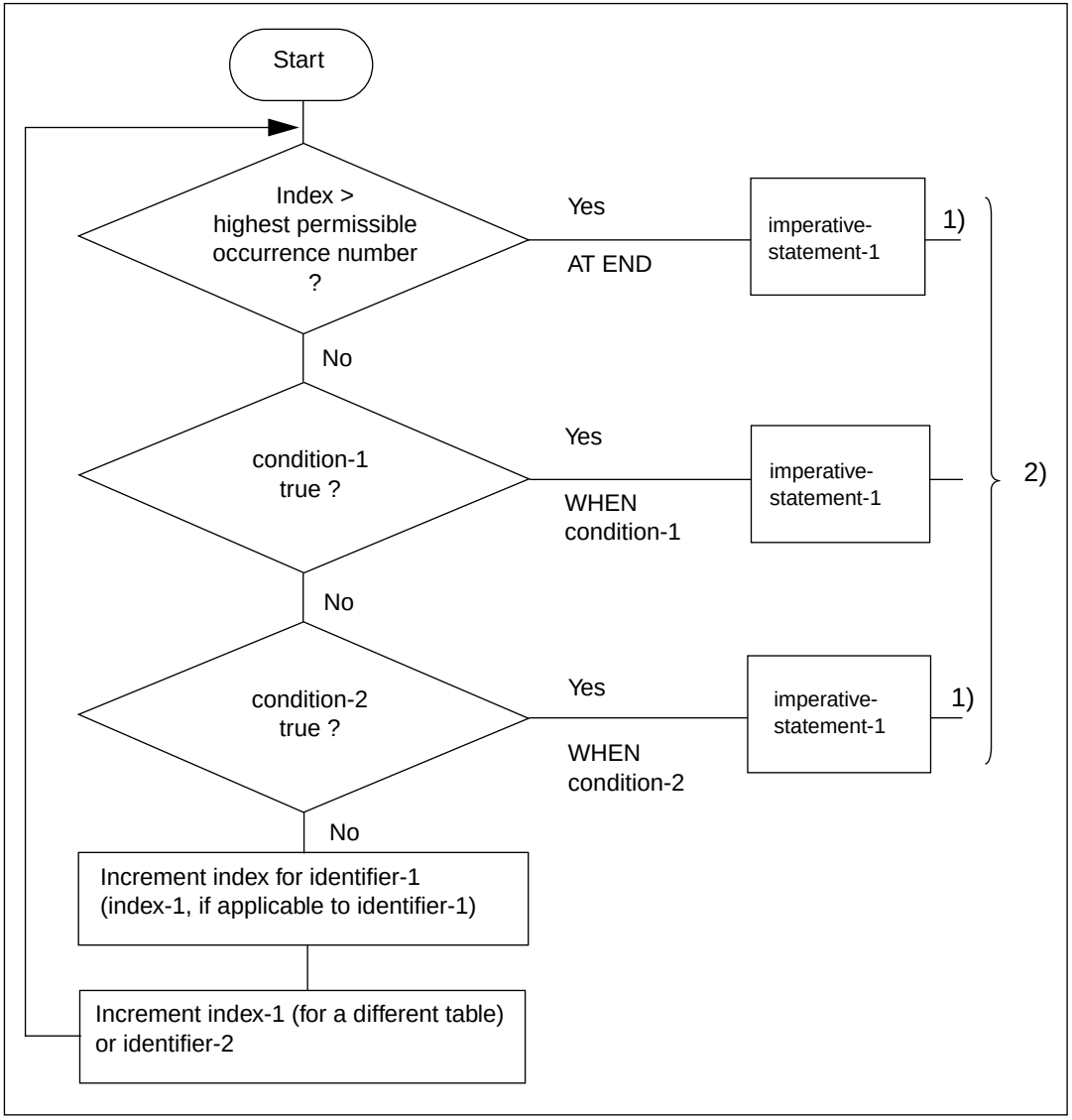


Fig. 3-1 Format 1 SEARCH statement

1) These operations are included only when specified in the SEARCH statement.

2) Each of these control transfers is to the next sentence, unless the imperative-statement ends with a GO TO statement.

Example 3-100

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SEARCH1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LETTER-TEST PIC X VALUE LOW-VALUE.
    88 LETTER-FOUND VALUE HIGH-VALUE.
01  INPUT-LETTER PICTURE A.
01  LETTER-WEIGHT-TABLE.
        03  LETTER-TABLE 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  LETTER PICTURE A.
            04  VAL PICTURE 99.
PROCEDURE DIVISION.
MAIN SECTION.
LETTER-SEARCH.
    PERFORM UNTIL LETTER-FOUND
      ACCEPT INPUT-LETTER FROM T
      SET PI TO 1
      SEARCH LETTER-TABLE VARYING PI
         AT END DISPLAY "Letter is not alphabetic" UPON T
                EXIT TO TEST OF PERFORM
         WHEN LETTER (PI) = FUNCTION UPPER-CASE (INPUT-LETTER)
              SET LETTER-FOUND TO TRUE
      END-SEARCH
    END-PERFORM.
FOUND.
    DISPLAY LETTER (PI) " is assigned to " VAL (PI) UPON T.
    STOP RUN.
```

In this example, the table named LETTER-TABLE consists of 26 elements.

Each element contains a letter of the alphabet followed by a value associated with the letter. The table is indexed by the index-names LI and PI.
The SEARCH statement searches the table for the element whose LETTER matches the current contents of the area called INPUT-LETTER. The associated index is PI.
The search starts at the beginning of the table since PI points to the first table element.
If the search is successful, the statement GO TO FOUND is executed. In this case, the index PI points to the element satisfying the condition. For example, if INPUT-LETTER contains B, the index points to the second table element.
If the search is unsuccessful, the statements following the AT END phase are executed.

Format 2

```
SEARCH ALL identifier-1 [AT END imperative-statement-1]

       WHEN condition {imperative-statement-2}
                      {NEXT SENTENCE         }

    [END-SEARCH]
```

Syntax rules for format 2

1. identifier-1 specifies the table to be searched.
2. The description of identifier-1 must contain an OCCURS clause with the INDEXED BY and ASCENDING/DESCENDING phrases.
3. identifier-1 must not be subscripted or indexed or subjected to reference modification.
4. The AT END phrase specifies the statement to be executed if condition cannot be satisfied for any setting of the index within the valid range (see rule 10).
5. condition specifies the condition that must be satisfied during the execution of the SEARCH statement.
6. condition must be one of the following types of conditions (see also under "Conditions", page 218):
   a) A relation condition incorporating EQUAL TO or the equal sign (=) as relational operator.

      Either the subject or the object (but not both) of the relation condition must consist solely of one of the data-names that appear in the ASCENDING/DESCENDING phrase of identifier-1. Each data-name must be indexed by the first index associated with identifier-1. It may be qualified, but not subjected to reference modification.

   b) A condition-name condition in which the VALUE clause describing the condition-name contains only a single literal.

      The conditional variable associated with the condition-name must be one of the data-names that appear in the ASCENDING/DESCENDING phrase of identifier-1.

   c) A combine condition formed from simple conditions of the types described above, with AND as the only connective.
7. Any data-name that appears in the ASCENDING/DESCENDING phrase of identifier-1 may be tested in condition. However, all data-names in the ASCENDING/DESCENDING phrase preceding the data-name to be tested, must also be tested in condition. No other tests can be made in condition.

8. imperative-statement-2 or NEXT SENTENCE specifies an action to be taken when condition is satisfied. Control passes to imperative-statement-2 or to the statement following the SEARCH statement.
9. The **first** index associated with identifier-1 is used for the search. This index does not have to be initialized with a SET statement since its initial value is ignored for the search.
10. The SEARCH ALL statement is executed as follows, whereby the table must be arranged in ascending or descending order of the key fields listed in the ASCENDING/DESCENDING phrase:
    a) During the search the value of the index associated with identifier-1 is varied.
    b) This setting is never less than the value corresponding to the first table element, and never greater than the value corresponding to the last table element.
    c) If condition cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 if the AT END phrase is specified, or to the next statement if the AT END phrase is omitted.
11. If condition can be satisfied, the index points to the table element satisfying the condition. Control then passes to imperative-statement-2 or to the next sentence.
12. When identifier-1 is a data item subordinate to a data item that contains an OCCURS clause, then two- or three-dimensional tables can be searched. In this case, an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Execution of the SEARCH ALL statement modifies only the setting of the index-name associated with identifier-1. Therefore, in order to search a two- or three-dimensional table, a SEARCH ALL statement must be executed for each possible value of the superordinate index.
13. If, in the AT END phrase and the WHEN conditions, none of the statements specified terminates with a GO TO statement, then control will pass to the next statement after the imperative-statement is executed.
14. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.
15. The scope of a SEARCH statement may be terminated by any of the following:
    a) An END-SEARCH phrase at the same level of nesting.
    b) A separator period.
    c) An ELSE or END-IF phrase associated with a previous IF statement.

Example 3-101

(WHEN conditions)

Data Division entries:

```
       ...
77  A-VALUE PICTURE 9.
       ...
    02  TABLE-ITEM OCCURS 5 TIMES ASCENDING KEY IS A B C;
                INDEXED BY I.
        03  A  PICTURE 99.
        03  B  PICTURE 9.
            88 UNDER-30 VALUE 1.
            88 OVER-30 VALUE 2.
        03  C  PICTURE 9.
       ...
```

Valid WHEN phrases (in Procedure Division)

```
     WHEN A(I) = 10
     WHEN A(I) = 20 AND UNDER-30(I)
     WHEN A(I) = 15 AND OVER-30(I) AND C(I) = A-VALUE
```

Example 3-102

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SRCHALL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T
    SYSIPT IS INFILE.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  IPD PIC 9(3).
77  INPUT-LN PIC 9(6).
01  EMPLOYEE-TABLE.
    02  PERSON OCCURS 100 TIMES INDEXED BY PI,
                    ASCENDING KEY IS DEPARTMENT LIFE-NUMBER.
        03  DEPARTMENT  PIC 9(3).
        03  LIFE-NUMBER PIC 9(6),
        03  NAME        PIC X(20).
PROCEDURE DIVISION.
MAIN SECTION.
P1.
    PERFORM VARYING PI FROM 1 BY 1 UNTIL PI > 100
      ACCEPT PERSON (PI) FROM INFILE
    END PERFORM
```

```
SEARCH ALL PERSON
AT END
  DISPLAY "Person is missing" UPON T
WHEN DEPARTMENT (PI) = IPD AND LIFE-NUMBER (PI) = INPUT-LN
  DISPLAY DEPARTMENT LIFE-NUMBER NAME UPON T
END-SEARCH
STOP RUN.
```

In this example, the table named PERSON consists of 2000 elements.

Each element in the table consists of a 3-byte numeric item called DEPARTMENT, a 6-byte numeric item called LIFE-NUMBER and a 20-byte alphanumeric item called NAME.

The table is arranged in ascending order by DEPARTMENT and, within DEPARTMENT, in ascending order by LIFE-NUMBER.

The first portion of the table might have the following contents:

DEPARTMENT	LIFE-NUMBER	NAME
101	123456	ADAM, D.
101	234561	LANGEWIESCHE, W.
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.

The SEARCH statement searches the table for an element whose DEPARTMENT matches the current contents of the area called IPD, and whose LIFE-NUMBER matches the current contents of the area called INPUT-LN. If the search is successful, the DISPLAY statement is executed. The index-name, PI, points to the element satisfying the condition.

For example, if IPD contains 183 and INPUT-LN contains 328512, the index-name points to the fifth element of the table.

If the search is unsuccessful, an appropriate message is issued.

SET statement

Function

The SET statement defines reference points for table handling operations by setting indices associated with table elements. The SET statement must be used for initializing an index prior to the execution of a SEARCH statement. The SET statement can also be used to change the status of external switches or to set the value of conditional variables.

Format 1	sets an integer data item, index or index data item to a specified value.
Format 2	increments or decrements the value of an index-name to represent a new occurrence number.
Format 3	changes the status of external switches
Format 4	sets the value of conditional variables

Format 1

```
     {index-1      }                {index-2      }
SET  {             }...   TO   {identifier-2}
     {identifier-1}                {integer-1    }
```

Syntax rules for format 1

In the following notes, all references to index-1 and identifier-1 apply equally to all recursions thereof.

1. Each index must be specified in an INDEXED BY phrase of an OCCURS clause.
2. Each identifier must reference either an index data item or a fixed-point numeric elementary item described as an integer.
3. The value of integer-1 or identifier-2 must be a valid occurrence number in the corresponding table. The compiler does, however, permit other integer values (e.g. 0 or negative numbers) within the permissible value range for index-1 (see page 94).
4. Indices or identifiers preceding the TO specify the item whose value is to be set.
5. index-2, identifier-2 and integer-1 (which follow the TO) specify the value to which the receiving item (e.g. index-1) is to be set.
6. When the SET statement is executed, one of the following actions occurs:
   a) index-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-2, identifier-2, or integer-1.

If identifier-2 references an index data item, or if index-2 is related to the same table as index-1, no conversion takes place.

b) If identifier-1 is an index data item, it is set equal to either the contents of index-2 or identifier-2, where identifier-2 is also an index data item. No conversion takes place. integer-1 cannot be used in this case.

c) If identifier-1 is not an index data item, it is set to an occurrence number that corresponds to the value of index-2. Neither identifier-2 nor integer-1 can be used in this case.

This process is repeated for each recurrence of index-1 or identifier-1. Each time, the value of index-2 or identifier-2 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1 is evaluated before the value of the respective data item is changed.

7. The table below indicates the validity of various operand combinations in the SET statement. The letters following the slashes refer to the rules listed above under point 7; for example, valid/c indicates that a combination of items is valid according to rule c) above.

Sending item	**Receiving item**		
	Integer data item PIC9(n)	Index-name INDEXED BY	Index-name USAGE IS INDEX
Integer literal	no/c	valid/a	no/b
Integer data item	no/c	valid/a	no/b
Index-name	valid/c	valid/a	valid/b*
Index data item	no/c	valid/a*	valid/b*

Table 3-20: Valid uses of the SET statement

* No conversion takes place

8. If index-2 is used, its value prior to execution of the SET statement must correspond to an occurrence number of an element in the associated table.

Example 3-103

Data Division entries:

```
    02 TABLE-A PICTURE XXX OCCURS 50,
               INDEXED BY IN-A1, IN-A2, IN-A3.
    02 TABLE-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.
```

The Procedure Division statements are shown in the tabular presentation below:

Statement	Operands used	Action taken [1]
`SET IN-A2 TO IN-A1`	Index-name set to index-name	Simple move (displacement to displacement).
`SET IN-A1 TO D-1`	Index-name set to numeric data item	Multiply (numeric item minus 1) by item length to get displacement.
`SET IN-A1 TO ID-1`	Index-name set to index data item	Simple move (displacement to displacement).
`SET IN-A1 TO 4`	Index-name set to literal	Multiply (literal minus 1) [2] by item length to get displacement.
`SET ID-1 TO IN-A1`	Index data item set to index-name	Simple move (displacement to displacement)
`SET D-1 TO IN-A1`	Numeric data item set to index-name	Divide displacement by item length and add 1, to get occurrence number.
`SET IN-B TO IN-A1`	Index-name set to index-name (different tables)	Divide displacement (i.e., contents of IN-A1) by item length for TABLE-A and add 1, to get occurrence number. Then, multiply (occurrence number minus 1) by item length for TABLE-B, to get displacement.

1) See "Indexing" (page 94) for the relationships between occurrence number and displacement.

2) Calculated at compilation time; simple move during program run.

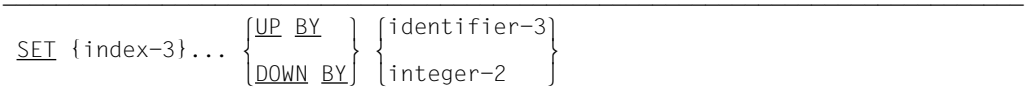
Example 3-104

```
02  TABLE-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,TABLE-A (IND-1) TO IND-2.
```

The third SET statement is equivalent to the following two statements, which are executed in the order in which they appear:

Statement	Action
`SET IND-1 TO IND-2`	IND-1 is set to the occurrence number 7, the current value of IND-2.
`SET TABLE-A (IND-1) TO IND-2`	Since the first SET statement sets IND-1 to 7, TABLE-A (IND-1) = TABLE-A (7). Thus, this statement sets TABLE-A (7) to 7.

Format 2



Syntax rules for format 2

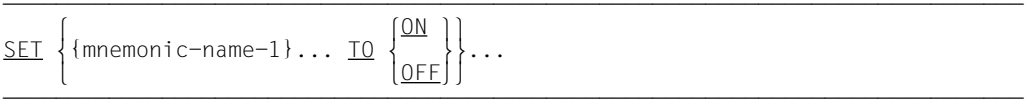
1. Each index must be specified in an OCCURS clause with INDEXED BY phrase.
2. index-3... specifies the storage index whose value is to be changed.
3. identifier-3 must be a fixed-point numeric elementary item described as an integer.
4. integer-2 must be a non-zero integer and may contain a positive sign.
5. The UP BY or DOWN BY phrase specifies that the contents of the index specified is to be either incremented (UP BY) or decremented (DOWN BY) by a value which corresponds to the occurrence number representing the value of identifier-3 or integer-2.

Example 3-105

```
02  TABLE-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.
```

The first SET statement sets index IND-1 to occurrence number 4; the second, to 6 (i.e. 4 + 2); and the third, to 3 (i.e. 6 − 3).

Format 3



Syntax rule

Each mnemonic-name must be associated with an external switch whose status can be altered (see "SPECIAL-NAMES paragraph", page 125).

Format 4

Syntax rule

1. condition-name-1 must be associated with a conditional variable (see "Condition-name condition", page 219).
2. If FALSE is specified in the SET statement, it must also be specified in the VALUE clause of the data description of the condition-name.

General rules

1. The literal specified in the VALUE clause and associated with condition-name-1 is entered in the conditional variable according to the rules governing the VALUE clause (see page 202).

If more than one literal is specified in the VALUE clause, the conditional variable is set to the value of the first literal appearing in the VALUE clause.

2. If two or more condition-names are specified, they are treated as though a SET statement had been written for each individual condition-name.
3. For "SET...TO FALSE", the literal specified with FALSE in the VALUE clause which is associated with `condition-name` is entered in the conditional variable according to the rules governing the VALUE clause (see "SET condition-name TO TRUE").

Example 3-106

for format 4

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DAYSET.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  EXAMPLE1.
    02  WORKDAY  PICTURE X.
        88 MONDAY  VALUE "1".
        88 FRIDAY VALUE "5".
01  EXAMPLE2.
    02  WEEKDAY PIC X.
        88 WORKDAYS VALUE "1" "2" "3" "4" "5".
PROCEDURE DIVISION.
P1 SECTION.
SETTING.
    SET FRIDAY TO TRUE.
    DISPLAY "Workday =" WORKDAY UPON T.
    SET WORKDAYS TO TRUE.
    DISPLAY "Weekday =" WEEKDAY UPON T.
FINISH-PAR.
    STOP RUN.
```

Once the first SET statement has been performed, the data item WORKDAY (conditional variable) will contain the literal assigned in the VALUE clause to the condition-name FRIDAY: "5".

Once the second SET statement has been performed, the data item WEEKDAY will contain the literal "1".

STOP statement

Function

The STOP statement causes a permanent or temporary suspension of the execution of the object program (run unit).

Format

```
STOP { RUN     }
     { literal }
```

Syntax rules

1. The literal may be numeric, nonnumeric, or any figurative constant except ALL literal.
2. If the literal is numeric, it must be an unsigned integer.
3. If a STOP statement with RUN phrase appears in a sentence, then it must be the only statement in that sentence, or it must be the last statement in a sequence of imperative statements.
4. The STOP statement with the RUN phrase specified terminates execution of the program, and returns control to the operating system.
5. If STOP literal is used, the literal is communicated to the system operator. In this case, only the system operator can resume the program. Continuation of the program begins with the next executable statement.

General rules

1. If the number of characters of the nonnumeric literal exceeds the hardware capability of the master console or subconsole, then more than one physical output operation will be performed to output the literal.
2. During the execution of a STOP RUN statement, an implicit CLOSE statement without any optional phrases is executed for each file that is in the open mode in the run unit. Any USE procedures associated with any of these files are not executed.

STRING statement

Function

The STRING statement moves and juxtaposes the partial or complete contents of two or more data items into a single data item.

Format

```
STRING {{identifier-1}                  {identifier-2}  }
       {{            }... [DELIMITED BY{literal-2   }] }...
       {{literal-1   }                  {SIZE        }  }

       INTO identifier-3 [WITH POINTER identifier-4]

       [ON OVERFLOW imperative-statement-1]

       [NOT ON OVERFLOW imperative-statement-2]

       [END-STRING]
```

Syntax rules

1. Each literal may be any figurative constant, except ALL literal.
2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-4, must be described implicitly or explicitly as USAGE IS DISPLAY.
3. Where identifier-1... are elementary numeric data items, they must be described as integers without the symbol P in their PICTURE character-strings.
4. If DELIMITED BY is not explicitly specified, DELIMITED BY SIZE is applied.

All references below to identifier-1, literal-1 apply equally to identifier-2, literal-2 and to all recursions thereof.

5. identifier-1..., literal-1... represent the sending items; identifier-3 represents the receiving item.
6. identifier-2, literal-2 represent delimiters, i.e. they mark a character string up to which the contents of a sending item should be moved. If the SIZE phrase is used, the contents of the complete data item defined by identifier-1, literal-1 are moved. When a figurative constant is used as the delimiter, it is a single character nonnumeric literal.
7. When a figurative constant is specified as literal-1, literal-2, it refers to an implicit one character nonnumeric data item whose usage is DISPLAY.

8. identifier-3 must not represent an edited data item and must not be described with the JUSTIFIED clause.
9. identifier-4 is a counter item and must be described as an elementary numeric integer data item of sufficient size to accommodate the size of the data item referenced by identifier-3 plus the value 1.
   The symbol P is prohibited in the PICTURE character-string of identifier-6.
10. identifier-3 must not be subjected to reference modification.
11. END-STRING delimits the scope of the STRING statement.

General rules

1. When the STRING statement is executed, the transfer of data is governed by the following rules:
   a) The sending items literal-1 or the contents of identifier-1 are transferred to the receiving item referenced by identifier-3 in accordance with the rules for alphanumeric-to-alphanumeric moves, except that no space filling will be provided.
   b) If the DELIMITED BY phrase is specified without the SIZE phrase, the content of identifier-1... or the value of literal-1 is transferred character-by-character to the receiving data item, beginning with the leftmost character and continuing from left to right until the end of the sending data item is reached or until the character(s) specified by the delimiter referenced by literal-2, or the contents of identifier-2, are encountered. The delimiter is not transferred.
   c) If the DELIMITED BY phrase is specified with the SIZE phrase, the entire contents of the sending items referenced by literal-1 or identifier-1 are transferred to the receiving item referenced by identifier-3 until all data has been transferred or the end of identifier-3 has been reached. The transfer takes place in the sequence specified in the STRING statement.
2. If the POINTER phrase is specified, the counter item referenced by identifier-4 is explicitly available to the user, i.e. an initial value greater than 0, and of adequate size to contain a value of the length of identifier-3 plus one byte, must be set by the user.
3. The subscripts of an identifier in the POINTER phrase are evaluated before the STRING statement is executed.
4. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-4 referencing a data item with an initial value of 1.
5. When moved to the receiving item identifier-3, each character is transferred separately from the sending item to the character position of identifier-3 which is determined by the value of the counter item identifier-4. Each time a character is moved, identifier-4 is incremented by 1. This is the only manner in which the value of identifier-4 changes during execution of the STRING statement.

6. At the end of execution of the STRING statement, only the portion of the receiving item referenced by identifier-3 into which characters were moved is changed. The rest of identifier-3 remains the same.
7. If at any time during or after the initialization of the STRING statement, but before processing is completed, the value associated with the counter item referenced by identifier-4 is either less than one or exceeds the number of character positions in the receiving item referenced by identifier-3, no (further) data is transferred to identifier-3, and the imperative statement in the ON OVERFLOW phrase, if specified, is executed.
8. If the ON OVERFLOW phrase is not specified when one of the conditions described above is encountered, control is transferred to the end of the STRING statement.
9. The imperative statement in the NOT ON OVERFLOW phrase is executed if the STRING statement ends without one of the conditions described above having been encountered.

Example 3-107

```
IDENTIFICATION DIVISION.
PROGRAM-ID. STRNG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  FIELD1   PIC X(16)  VALUE "INITIAL RESTRICTION".
77  FIELD2   PIC X(12)  VALUE "VALUE RANGE".
77  FIELD3   PIC X(25)  VALUE SPACES.
77  FIELD4   PIC 99     VALUE 3.
PROCEDURE DIVISION.
PROC SECTION.
MAIN.
    DISPLAY "Before STRING: " UPON T.
    PERFORM DISPLAY-FIELDS.
    STRING FIELD1, FIELD2 DELIMITED BY "B",
           " INVALID" DELIMITED BY SIZE
      INTO FIELD3 WITH POINTER FIELD4
    ON OVERFLOW
      DISPLAY "Error" UPON T
    END-STRING
    DISPLAY "After STRING" UPON T.
    PERFORM DISPLAY-FIELDS.
    STOP RUN.
DISPLAY-FIELDS.
    DISPLAY "Field1 = *" FIELD1 "*" UPON T.
    DISPLAY "Field2 = *" FIELD2 "*" UPON T.
    DISPLAY "Field3 = *" FIELD3 "*" UPON T.
    DISPLAY "Field4 = *" FIELD4 "*" UPON T.
```

Result:

Before STRING	After STRING
FIELD1 = *INITIAL-RESTRICTION FIELD2 = *VALUE-RANGE* FIELD3 = * * FIELD4 = *03*	FIELD1 = *INITIAL-RESTRICTION* FIELD2 = *VALUE-RANGE* FIELD3 = * SET-INITIAL-VALUES * FIELD4 = *22*

SUBTRACT statement

Function

The SUBTRACT statement is used to subtract the value of a numeric item, or the sum of two or more values of numeric data items, from one or more items.

Format 1 of the SUBTRACT statement stores the difference in one of the operands. In a SUBTRACT statement, more than one subtraction may be specified by supplying more than one result item.

Format 2 of the SUBTRACT statement uses the GIVING phrase.

Format 3 of the SUBTRACT statement subtracts the items in one group item from the corresponding items in another group item.

Format 1

```
SUBTRACT {identifier-1}
         {literal-1   }...

          FROM {identifier-n [ROUNDED]}...

         [ON SIZE ERROR imperative-statement-1]

         [NOT ON SIZE ERROR imperative-statement-2]

         [END-SUBTRACT]
```

Syntax rules for format 1

1. Each identifier must refer to an elementary numeric data item.
2. The composite of operands determined by using all of the operands in a given statement, with the exception of the data item following the word GIVING, must not contain more than 18 digits (see "Arithmetic statements", page 236).
3. All literals and identifiers preceding the word FROM are added together, and the sum is subtracted from the current value of identifier-n... . The results of the subtraction are stored as the new value of identifier-n.
4. END-SUBTRACT delimits the scope of the SUBTRACT statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the ROUNDED and (NOT) ON SIZE ERROR phrases are described.

Example 3-108

for format 1

Statement	PICTURE of result item	Calculation
SUBTRACT A, B FROM D	999	D − (A + B) stored in D as nnn
SUBTRACT A FROM B,C	9(3) for B 9(5) for C	B − A stored in B as nnn C − A stored in C as nnnnn

Format 2

```
SUBTRACT {identifier-1}... FROM {identifier-m}
         {literal-1   }         {literal-m   }

          GIVING {identifier-n [ROUNDED]}...

        [ON SIZE ERROR imperative-statement-1]

        [NOT ON SIZE ERROR imperative-statement-2]

        [END-SUBTRACT]
```

Syntax rules for format 2

1. Each identifier preceding the word GIVING must refer to an elementary numeric item.
2. identifier-n may refer either to an elementary numeric item or to an elementary numeric edited item.
3. The composite of operands determined by using all of the operands in a given statement, except for the data items which follow the word GIVING, must not contain more than 18 digits (see "Arithmetic statements").
4. All literals or identifiers preceding the word FROM are added together, and the sum is subtracted from literal-m or identifier-m. The result of the subtraction is the new value of identifier-n... .
5. END-SUBTRACT delimits the scope of the SUBTRACT statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the GIVING, ROUNDED, and (NOT) ON SIZE ERROR phrases are described.

Example 3-109

for format 2

Statement	PICTURE of result item (C)	Calculation
SUBTRACT A, B FROM 100 GIVING C.	9(5)	100 − (A + B) stored in C as nnnnn

Format 3

```
SUBTRACT {CORR          } identifier-1
         {CORRESPONDING}

          FROM identifier-2 [ROUNDED]

         [ON SIZE ERROR imperative-statement-1]

         [NOT ON SIZE ERROR imperative-statement-2]

         [END-SUBTRACT]
```

Syntax rules for format 3

1. Each identifier must refer to a group item.
2. The composite of operands, which is determined separately for each pair of corresponding data items, must not be greater than 18 digits (see "Arithmetic statements", page 236).
3. When the CORRESPONDING phrase is used, the elementary items within the first operand (identifier-1) are subtracted from the corresponding elementary items within the second operand (identifier-2). The results are stored in the items of the second operand.
4. END-SUBTRACT delimits the scope of the SUBTRACT statement.

Additional rules are given under "Options in arithmetic statements" (page 239ff), where the CORRESPONDING, ROUNDED, and (NOT) ON SIZE ERROR phrases are described.

Example 3-110

for format 3

Refer to the description of the CORRESPONDING phrase for an example of the use of this phrase (page 239).

UNSTRING statement

Function

The UNSTRING statement causes contiguous data in a sending item to be separated and placed into multiple receiving items.

Format

```
UNSTRING identifier-1

   [                    {identifier-2} [          {identifier-3}]    ]
   [DELIMITED BY [ALL] {              } [OR [ALL] {              }]...]
   [                    {literal-1   }  [          {literal-2   }]    ]

    INTO {identifier-4 [DELIMITER IN identifier-5] [COUNT IN identifier-6]}...

   [WITH POINTER identifier-7] [TALLYING IN identifier-8]

   [ON OVERFLOW imperative-statement-1]

   [NOT ON OVERFLOW imperative-statement-2]

   [END-UNSTRING]
```

Syntax rules

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant, except ALL literal.
2. identifier-1, identifier-2, identifier-3, identifier-5 must reference data items described, implicitly or explicitly, as alphanumeric.
3. identifier-4 may be described as either
   – alphabetic (except that the symbol B must not be used in the PICTURE character-string), or
   – alphanumeric or numeric (except that the symbol P must not be used in the PICTURE character-string) and must be described, implicitly or explicitly, as USAGE IS DISPLAY.
4. identifier-6, identifier-7, identifier-8 must reference integer data items.
   The symbol P must not be used in the PICTURE character-string.
5. All references to identifier-2, literal-1, identifier-4, identifier-5, identifier-6 apply analogously to all recursions thereof, and to identifier-3 and literal-2.
6. identifier-1 represents the sending area.

7. identifier-4 represents the receiving area, identifier-5 the receiving area for delimiters.
8. literal-1 or the data item referenced by identifier-2 specifies a delimiter
9. identifier-6 represents the count of the number of characters encountered within identifier-1 up to the delimiter, i.e. the number of characters within identifier-1 which are to be moved to identifier-4. This value does not include a count of the delimiter character(s).
10. The data item referenced by identifier-7 contains a value that indicates a character position within the area referenced by identifier-1 (relative to the beginning of that area).
11. The data item referenced by identifier-8 is a counter for the number of receiving items accessed during an UNSTRING operation.
12. literal-1 or identifier-2 may contain any character from the computer's character set.
13. identifier-1 must not be subjected to reference modification.
14. DELIMITER IN and COUNT IN can be used only in conjunction with DELIMITED BY.
15. END-UNSTRING delimits the scope of the UNSTRING statement.

General rules

1. No identifier may be defined with the level number 88.
2. When a figurative constant occurs as a delimiter, it represents a single-character nonnumeric literal.
3. When the ALL phrase is specified, contiguous occurrences of literal-1 or identifier-2 are treated as if they were only one occurrence, and one occurrence of literal-1 or identifier-2 is moved to the data item referenced by identifier-5.
4. When two contiguous delimiters are encountered, the current receiving area is space-filled if it is described as alphabetic or alphanumeric, or zero-filled if it is described as numeric.
5. literal-1, identifier-2 represent delimiters. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given, to be recognized as a delimiter.
6. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared with the sending item. If a match occurs, the character(s) in the sending item is (are) considered to be a single delimiter. No character(s) in the sending item can be considered as part of more than one delimiter. Delimiters cannot overlap.
   Each delimiter is applied to the sending item in the sequence specified in the UNSTRING statement.

7. When the UNSTRING statement is initiated, identifier-4 represents the current receiving area. Data is transferred from identifier-1 to identifier-4 according to the following rules:

   a) If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by identifier-7.

      If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position of identifier-1.

   b) If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by literal-1 or the value of the data item referenced by identifier-2 is encountered.

      If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area.

      However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

      If the end of identifier-1 is reached before a delimiter is encountered, the examination terminates with the character examined last.

   c) The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement (see "MOVE statement", page 294).

   d) If the DELIMITER IN phrase is specified, the delimiting character(s) are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement (see "MOVE statement", page 294).

   e) If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into the area referenced by identifier-6 according to the rules for an elementary move.

   f) If the DELIMITED BY phrase is specified, the string of characters referenced by identifier-1 is further examined beginning with the first character to the right of the delimiter.

      If the DELIMITED BY phrase is not specified, the examination will continue from the character immediately following the last character to be moved.

   g) After the data is transferred to identifier-4, the current receiving area is represented by the next recurrence of identifier-4.

      The behavior described above is repeated until either all the characters in identifier-1 are exhausted or until there are no more receiving areas.

8. The initialization of the contents of the data items associated with the POINTER or TALLYING phrase is the responsibility of the user.
9. The contents of the data item referenced by identifier-7 will be incremented by 1 for each character examined in the data item referenced by identifier-1.

   When the execution of an UNSTRING statement with a POINTER phrase is completed, identifier-7 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.
10. When the execution of an UNSTRING statement with a TALLYING phrase is completed, identifier-8 contains a value equal to its initial value plus the number of receiving data items accessed.
11. Either of the following situations causes an overflow condition:

    a) If an UNSTRING is initiated, and the value in the data item referenced by identifier-7 is less than 1 or greater than the size of the data item referenced by identifier-1.

    b) If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.
12. When an OVERFLOW condition exists, the UNSTRING operation is terminated.

    If an ON OVERFLOW phrase has been specified, the imperative statement given in this phrase is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next statement to be executed.
13. The imperative statement specified in the NOT ON OVERFLOW phrase is executed if the UNSTRING statement is terminated and none of the conditions described above has been encountered.
14. Subscripts and indexes for the identifiers are analyzed as follows:

    a) If the items identifier-1, identifier-7, identifier-8 are subscripted or indexed, the index value for these items is calculated once only, immediately before the UNSTRING statement is executed.

    b) Any subscripting of identifiers in the DELIMITED BY, INTO, DELIMITER IN, and COUNT phrases is evaluated immediately before the data is transferred to the respective data item.

Example 3-111

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UNSTRING.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FIELD      PIC  X(12)  VALUE  "ABCDEFGHIJKL".
01  AREA1.
    02  PART1   PIC X       VALUE  SPACES.
    02  PART2   PIC XX      VALUE  SPACES.
    02  PART3   PIC XXX     VALUE  SPACES.
01  NUMB        PIC 99      VALUE  ZERO.
PROCEDURE DIVISION.
PROC SECTION.
MAIN.
    DISPLAY "Before UNSTRING" UPON T.
    PERFORM DISPLAY-FIELDS.
*
    UNSTRING FIELD DELIMITED BY "E"  OR  "H"  OR  "K"  OR "L"
        INTO PART3, PART2, PART1
        TALLYING IN NUMB.
    END-UNSTRING
*
DISPLAY "After UNSTRING" UPON T.
    PERFORM DISPLAY-FIELDS.
    STOP RUN.
DISPLAY-FIELDS.
    DISPLAY  "Field  = *"  FIELD  "*" UPON T.
    DISPLAY  "Part1 = *"  PART1 "*" UPON T.
    DISPLAY  "Part2 = *"  PART2 "*" UPON T.
    DISPLAY  "Part3 = *"  PART3 "*" UPON T.
    DISPLAY  "Numb  = *"  NUMB  "*" UPON T.
```

Result:

Before UNSTRING	After UNSTRING
FIELD = *ABCDEFGHIJKL* PART1 = * * PART2 = * * PART3 = * * NUMB= *00*	FIELD = *ABCDEFGHIJKL* PART1 = *I* PART2 = *FG* PART3 = *ABC* NUMB= *03*

3.10 Debugging

As debugging aids, the compiler provides the user with debugging lines and a compile time switch for debugging lines.

Debugging lines

A debugging line is any line with a ’D’ in the indicator area of the line.

Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.
The content of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

After all COPY statements have been processed, a debugging line will be considered to have all the characteristics of a comment line if the WITH DEBUGGING MODE clause is not specified.
Successive debugging lines are allowed.
A debugging line is only permitted in the separately compiled program after the OBJECT-COMPUTER paragraph.

Compile time switch

The WITH DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph. It serves as a compile time switch over the debugging lines written in the separately compiled program.
When the WITH DEBUGGING MODE clause is specified in a separately compiled program, all debugging lines are compiled as specified in the rules described above. When the WITH DEBUGGING MODE clause is not specified, all debugging lines are compiled as if they were comment lines.

4 Sequential file organization

4.1 File concepts

A file is a collection of records that can be moved to, or read from, a volume. The user defines the organization of the file as well as the mode and order in which the records are processed.

The organization of a file describes its logical structure. There are sequential, indexed, and relative types of file organization. The file organization which is defined at the time a file is created cannot be changed later on.

A sequential file can only be processed sequentially, i.e. the records are either read or written in the order predetermined by the file. In the case of sequential files on disk storage devices, records may also be updated in place. This requires a READ statement followed immediately by a REWRITE.

4.1.1 Sequential organization

When sequential file organization is used, the logical records are placed on the file or read sequentially either forwards or backwards (REVERSED) in the order in which they were generated.

This type of file organization must be used for magnetic tape or unit record files and may be used for disk storage files. Sequentially organized files require no key for record processing.

4.1.2 Line sequential organization

The line sequential organization of COBOL files is one of the language elements supported by X/Open standards.It serves to generate and process text files that can be processed and generated by the text editors of the operating system.

4.1.3 I-O status

The I-O status is a value that can be used in a COBOL program to check the status of an input/output operation. In order to do this, the FILE STATUS clause must be specified in the FILE CONTROL paragraph of the Environment Division.
The I-O status value is transferred to a two-character data item

– during the execution of a CLOSE, OPEN, READ, REWRITE, or WRITE statement,

– prior to the execution of any associated imperative statement, and

– prior to the execution of any corresponding USE AFTER STANDARD EXCEPTION procedure.

The table below shows the I-O status values and their meanings:

I-O status	Meaning
	Execution successful
00	The I-O statement terminated normally. No further information regarding the I-O operation is available.
04	Record length conflict: A READ statement terminated normally. However, the length of the record read lies outside the limits defined in the record description entry for this file.
05	Successful execution of an OPEN INPUT/I-O/EXTEND on a file; however, the referenced file indicated by the OPTIONAL phrase was not present at the time the OPEN statement was executed.
07	1. Successful OPEN statement with NO REWIND clause on a file that is on a UNIT-RECORD medium. 2. Successful CLOSE statement with NO REWIND, REEL/UNIT, or FOR REMOVAL clause on a file that is on a UNIT-RECORD medium.
	Execution unsuccessful: at end condition
10	An attempt was made to execute a READ statement. However, no next logical record existed, because the end-of-file was encountered. A sequential READ statement with the OPTIONAL phrase was attempted for the first time on a nonexistent file.
	Execution unsuccessful: unrecoverable error
30	1. No further information regarding the I-O operation is available (the DMS code provides further information). 2. In the case of line sequential processing: unsuccessful attempt to access PLAM item
34	An attempt was made to write outside the sequential file boundaries set by the system.
35	An OPEN statement with the INPUT/I-O/EXTEND phrase was attempted on a nonexistent file.

I-O status	Meaning
37	OPEN statement on a file that cannot be opened in any of the following ways: 1. OPEN OUTPUT/I-O/EXTEND on a write-protected file (password, RETENTION-PERIOD, ACCESS=READ in catalog) 2. OPEN I-O on a tape file 3. OPEN INPUT on a read-protected file (password)
38	An attempt was made to execute an OPEN statement for a file previously closed with the LOCK phrase.
39	The OPEN statement was unsuccessful as a result of one of the following conditions: 1. One or more of the operands ACCESS-METHOD, RECORD-FORMAT or RECORD-SIZE were specified in the SET-FILE-LINK command with values deviating from the corresponding explicit or implicit program specifications. 2. record length errors occurred for input files (catalog check if RECFORM=F), or 3. The record size is greater than the BLKSIZE entry in the catalog (in the case of input files). 4. The catalog entry of one of the FCBTYPE, RECFORM or RECSIZE (if REC-FORM=F) operands for an input file is in conflict with the corresponding explicit or implicit program specifications or with the specifications in the SET-FILE-LINK command.
	Execution unsuccessful: logical error
41	An attempt was made to execute an OPEN statement for a file which was already open.
42	An attempt was made to execute a CLOSE statement for a file which was not open.
43	While accessing a disk file opened with OPEN I-O, the most recent I-O statement executed prior to a REWRITE statement was not a successfully executed READ statement.
44	Boundary violation: 1. An attempt was made to execute a WRITE statement. However, the length of the record is outside the range allowed for this file. 2. An attempt was made to execute a REWRITE statement. However, the record to be rewritten did not have the same length as the record to be replaced.
46	An attempt was made to execute a READ statement for a file in INPUT or I-O mode. However, there is no valid next record since: 1. the preceding READ statement was unsuccessful without causing an at end condition 2. the preceding READ statement resulted in an at end condition.
47	An attempt was made to execute a READ statement for a file not in INPUT or I-O mode.
48	An attempt was made to execute a WRITE statement for a file not in OUTPUT or EXTEND mode.

I-O status	Meaning
49	An attempt was made to execute a REWRITE statement for a file not open in I-O mode.
	Other conditions with unsuccessful execution
90	System error; no further information available regarding the cause.
91	System error; a system call terminated abnormally; either an OPEN error or no free device; the actual cause is evident from the DMS code (see "FILE STATUS clause", page 365)
95	The specifications in the BLOCK-CONTROL-INFO or BUFFER-LENGTH operands of the SET-FILE-LINK command are not consistent with the file format, the block length, or the format of the volume being used.

4.2 Language elements of the Environment Division

INPUT-OUTPUT SECTION

Function

The INPUT-OUTPUT SECTION deals with the following:

– the definition of each file used in the program,
– the assignment of these files to external devices, and
– the transmission and handling of data between external devices and the object program.

This section is divided into two paragraphs:

– the FILE-CONTROL paragraph, which names the files used in the program and assigns them to external devices, and
– the I-O-CONTROL paragraph, which defines special input/output techniques.

Format

```
 A       Margin indication
 |
 |
 |
 |
 INPUT-OUTPUT SECTION.

 FILE-CONTROL. {file-control-entry}...
[I-O-CONTROL.  [input-output-control-entry]]
```

Syntax rules

1. All sections and paragraphs must begin at in area A.
2. The entire INPUT-OUTPUT SECTION is optional. If the INPUT-OUTPUT SECTION is defined, the FILE-CONTROL paragraph must also be specified.

FILE-CONTROL paragraph

Function

The FILE-CONTROL paragraph is used to give each file a name. The files are assigned to one or more external devices, and the information required for file processing is made available. This information indicates how the data is organized and how it is to be accessed.

Format

```
 A   B   Margin indication
 |   |
 |   |
 ↓   ↓
 FILE-CONTROL.

     SELECT clause
     ASSIGN clause
    [ORGANIZATION clause]
    [PADDING CHARACTER clause]
    [RECORD DELIMITER clause]
    [ACCESS MODE clause]
    [RESERVE clause]
    [FILE STATUS clause].
```

Syntax rules

1. The FILE-CONTROL heading must be written in area A; all subsequent entries must be written in area B.
2. The SELECT clause must be the first entry in the FILE-CONTROL paragraph. All other clauses may appear in any order.
3. The PADDING CHARACTER, RECORD DLIMITER and RESERVE clauses are treated as comment lines by the compiler.

In the pages that follow, the SELECT and ASSIGN clauses are described first, followed by the remaining clauses in alphabetical order.

SELECT clause

Function

The SELECT clause is used to name each file in a program.

Format 1 applies to all files except sort files.

Format 2 is suitable for sort files

(see chapter 10, "Sorting of records", page 652).

Format 1

```
SELECT [OPTIONAL] file-name
```

Syntax rules

1. file-name stands for the name by which a file is referenced in the source program (internal file-name). Each file-name used in a program may only occur in one SELECT clause.
2. Each file specified in a SELECT clause must have a file description (FD) entry in the Data Division of the source program.
3. The OPTIONAL phrase is required for files that are not necessarily present at object time. When a file is not present at object time, the first READ statement for that file passes control to the associated at end condition.

General rules

1. If no SET-FILE-LINK command is given for a file when the program is executed, the runtime system creates a file named FILE.COB85.linkname at OPEN OUTPUT. The link name is formed from the specifications in the ASSIGN clause (see "ASSIGN clause", page 362).
2. If the OPTIONAL phrase refers to an *external* file, OPTIONAL must be specified in all programs that describe this external file.

ASSIGN clause

Function

The ASSIGN clause assigns an external device to a file of the COBOL program. One ASSIGN clause is required for each file in the program.

Format

```
            ⎧PRINTER [literal-1]⎫
            ⎪implementor-name-1 ⎪
ASSIGN TO   ⎨                   ⎬...
            ⎪literal-2          ⎪
            ⎩data-name-1        ⎭
```

Syntax rules

1. PRINTER specified without literal-1 refers to the logical system file SYSLST.

   PRINTER literal-1 refers to a print file.

   In both cases, the compiler reserves the feed control character, which is not accessible to the user.

2. implementor-name-1 refers to devices which are named and assigned as follows:

Device name	Assigned system file
PRINTER01 - PRINTER99	SYSLST01 - SYSLST99
SYSIPT	SYSIPT
SYSOPT	SYSOPT

3. Files assigned by means of PRINTER or implementor-name-1 must not be external files.
4. literal-1, literal-2 or the contents of data-name-1 specifies the link name for the file. The name must be alphanumeric, must be entered in uppercase letters and may not be a figurative constant. The link name is formed from the first eight characters of the literal and must therefore be unique within the program. If the last character of the link name thus formed is a hyphen (-), then it is replaced by a # character.
5. data-name-1 must not be qualified.
6. data-name-1 must be defined as an alphanumeric data item in the WORKING-STORAGE SECTION or LINKAGE SECTION.

General rules

1. The type of file organization must be specified in the ORGANIZATION clause (see page 366).
2. Only the first entry in the ASSIGN clause is evaluated; all other entries are ignored (PRINTER literal-1 is regarded as one entry).

ACCESS MODE clause

Function

The ACCESS MODE clause determines the manner in which the records of a file are to be accessed.

Format

```
ACCESS MODE IS SEQUENTIAL
```

General rules

1. If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is assumed.
2. SEQUENTIAL means that records are read or written sequentially, i.e. the next logical record of the file is made available when a READ statement is executed, or the next logical record is placed on that file when a WRITE statement is executed.
3. If sequential access is defined for an *external* file, then sequential access must also be defined in all other programs that describe this external file.

FILE STATUS clause

Function

The FILE STATUS clause specifies a data item that indicates the status of input/output operations during processing. In addition, by specifying a further item, an additional error code is made available.

Format

```
FILE STATUS IS data-name-1 [, data-name-2]
```

Syntax rules

1. data-name-1 and data-name-2 must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION of the Data Division.
2. data-name-1 must be a two-byte numeric (USAGE DISPLAY only) or alphanumeric item.
3. data-name-2 must be a 6-character group item with the following format:

```
01  data-name-2.
    02  data-name-2-1  PIC 9(2) COMP.
    02  data-name-2-2  PIC X(4).
```

General rules

1. If the FILE STATUS clause is specified, the runtime system copies the I-O status to data-name-1.
2. If specified, data-name-2 is assigned as follows:t
   a) If data-name-1 has the value 0, the contents of data-name-2 are undefined.
   b) If data-name-1 has a non-zero value, data-name-2 contains the additional error code. The value 64 in data-name-2 indicates that this code is the (BS2000) DMS code; the value 96 in data-name-2 indicates that the code is the (POSIX) SIS code. The command HELP DMS <contents-of-data-name-2-2> or HELP SIS <contents-of-data-name-2-2> supplies more detailed information on the corresponding error code.
3. The I-O status is copied during the execution of each OPEN, CLOSE, READ, WRITE, or REWRITE statement that references the specified file, and prior to the execution of each corresponding USE procedure (see "I-O status", page 356).

ORGANIZATION clause

Function

The ORGANIZATION clause defines the logical structure of a file.

Format

```
[ORGANIZATION IS] {SEQUENTIAL       }
                  {LINE SEQUENTIAL  }
```

General rules

1. File organization is defined at the time a file is created and cannot be changed subsequently.
2. If the ORGANIZATION clause is omitted, ORGANIZATION IS SEQUENTIAL is assumed.
3. If sequential or line sequential organization is defined for an *external* file, then sequential or line sequential organization must be defined in all programs that describe this external file.

PADDING CHARACTER clause

Function

The PADDING CHARACTER clause is used to specify a padding character.

Format

```
PADDING CHARACTER IS {data-name}
                     {literal  }
```

The PADDING CHARACTER clause is treated as a comment by the compiler.

RECORD DELIMITER clause

Function

The RECORD DELIMITER clause indicates the method of determining the length of a variable-length record on the external medium.

Format

```
RECORD DELIMITER IS {STANDARD-1}
                    {BS2000    }
```

The RECORD DELIMITER clause is treated as a comment by the compiler.

RESERVE clause

Function

The RESERVE clause allows the user to modify the number of input/output areas (buffers) allocated to the program by the compiler.

Format

```
RESERVE integer  [AREA ]
                 [AREAS]
```

The RESERVE clause is treated as a comment by the compiler.

I-O-CONTROL paragraph

Function

The I-O-CONTROL paragraph defines the events at whose occurrence restart points are to be established and specifies the memory area which is to be shared by different files. In addition, it indicates the location of files on multiple-file reels and defines special input/output conditions.

Format

```
A   B     Margin indication
|   |
|   |
|   |
I-O-CONTROL.

    [MULTIPLE FILE TAPE clause]...
    [RERUN clause]...
    [SAME AREA clause]...
```

Syntax rule

I-O-CONTROL must be written starting in area A. All subsequent entries must be written in area B.

MULTIPLE FILE TAPE clause

Function

The MULTIPLE FILE TAPE clause is required when more than one file shares the same physical reel of tape.

Format

```
MULTIPLE FILE TAPE CONTAINS {file-name-1 [POSITION integer-1}...
```

Syntax rule

integer can have any value between 1 and 3315.

General rules

1. When all file-names are supplied in the same order in which they appear on a reel, the POSITION phrase may be omitted.
2. If any of the files is not specified, then the positions must be supplied as relative to the beginning of the tape.
3. Irrespective of the number of files that share the reel of tape, only those used by the program need be specified.
4. Only one file may be opened on the same reel at any given time.
5. REWIND can be carried out when the last file of the tape has been processed.
6. If the MULTIPLE FILE TAPE clause is specified for an *external* file, then the MULTIPLE FILE TAPE clause must be specified in all programs that describe this external file. If position numbers are specified, these must be the same in all programs.

For further information see the "COBOL85 User Guide" [1].

RERUN clause

Function

A RERUN clause indicates where and when restart-point records are to be issued. A restart-point record describes the status of a program at a specified point during program execution. It is produced automatically by the operating system upon the request of the program and contains all information necessary to restart the program from that point. The RERUN clause controls such requests by the COBOL object. For further information on the RERUN clause, see "COBOL85 User Guide" [1].

Format

```
                                                  ⎧⎧           ⎧REEL⎫    ⎫                  ⎫
                                                  ⎪⎪[END OF] ⎨      ⎬    ⎪                  ⎪
                                                  ⎪⎪           ⎩UNIT⎭    ⎬ OF file-name-2   ⎪
RERUN ⎡ON ⎧file-name-1       ⎫⎤ EVERY ⎨⎨integer-1 RECORDS}⎭                  ⎬
      ⎣   ⎩implementor-name⎭⎦                ⎪                                  ⎪
                                                  ⎪ integer-2 CLOCK-UNITS                ⎪
                                                  ⎩ condition-name                       ⎭
```

Syntax rules

1. file-name-1 must be the name of a sequential tape file; it must be described in the FILE SECTION.
2. implementor-name has the format
   SYSnnn
   where 000 ≤ nnn ≤ 244
   nnn determines the processing mode.
   If nnn ≥ 200, restart points are written alternately to two restart files. This enables the restart to be made from the two most recent restart points.
3. If the same implementor-name is also used in the SELECT clause, it must be associated with a tape file.
4. The END OF REEL/UNIT phrase may be used only when file-name-2 describes a sequential file. REEL and UNIT are synonymous.
5. implementor-name must be specified in the RERUN clause when integer-1 RECORDS or integer-2 CLOCK-UNITS is used.
6. Subject to the following restrictions, more than one RERUN clause may be specified for the same file-name-2:
   a) When specifying more than one integer-1 RECORDS phrase, the same file-name-2 must appear only once in them.

b) When specifying more than one END OF REEL or END OF UNIT phrase, the same file-name-2 must appear only once in them.

7. file-name-1 or implementor-name specifies the file to which the restart points are to be output.
8. Restart points are written as follows:
   a) If file-name-1 is specified, the restart points are output to each reel or volume assigned to file-name-1.
   b) If implementor-name is specified, the restart points are written as follows:
      When implementor-name is assigned to a file-name from the SELECT clause, the file involved must be a tape file. The effect on restart point generation is the same as if the file-name had been specified directly in the RERUN clause (see 8a).

      When implementor-name is not assigned to a file from a SELECT clause, a disk storage file must be assigned at runtime; otherwise, a file with the name progid.RERUN.implementor-name is assigned by the runtime system. Only in this case will restart points be written to disk storage.
   c) If the same SYS numbers are specified in the ASSIGN clause and in the RERUN END OF REEL clause, the restart point is written to the end of the output tape.
9. There are four types of RERUN clauses, depending on the conditions under which restart points are requested.
   a) END OF REEL or END OF UNIT without specification of the ON clause: the restart points are written to file-name-2, which must specify an output file.
   b) END OF REEL or END OF UNIT with file-name-1 specified in the ON clause: the restart points are written to file-name-1, which must refer to an output file. Additionally, the usual end-of-reel handling is carried out for file-name-2. file-name-2 may be either an input or output file.
   c) END OF REEL or END OF UNIT with implementor-name in the ON clause: The restart points are written to a separate file (see 8b). File-name-2 may be an input or an output file.
   d) integer-1 RECORDS: The restart points are written to the file specified by file-name-1 or implementor-name, respectively, whenever integer-1 records have been processed. File-name-2 may be either an input or an output file with any organization or type of access; it must not be referenced in the USING/GIVING phrase or in an INPUT/OUTPUT procedure during sort (see chapter 10, "Sorting of records").
10. The CLOCK-UNITS phrase is treated by the compiler as a comment.
11. The condition-name phrase is also treated by the compiler as a comment.
12. If an *external* file is specified by file-name-1, the behavior is undefined.

SAME AREA clause

Function

The SAME AREA clause indicates which files are to share a specified input/output area during program execution.

Format 1 applies to all files except sort files, unless RECORD is specified.

Format 2 is suitable for sort files

(see chapter 10, "Sorting of records", page 650).

Format 1

```
SAME [RECORD] AREA FOR file-name-1 {file-name-2}...
```

Syntax rules

1. More than one SAME AREA clause may be included in a program. In this case, the following must be observed:

   A specific file-name must not appear in more than one SAME AREA clause. The same is true of the SAME RECORD AREA clause.

   A specific file-name may concurrently appear in a SAME RECORD AREA clause. In this case, all file-names appearing in the SAME AREA clause must also appear in the SAME RECORD AREA clause. The SAME RECORD AREA clause may also contain other file-names that do not appear in the SAME AREA clause.

2. The SAME AREA clause indicates that the specified files (no sort-files) are to share the input/output areas assigned to them.

3. The SAME RECORD AREA clause indicates that the specified files are to share the same storage areas for processing the current logical record.

   A logical record in the SAME RECORD AREA clause is considered a logical record of all files which are opened for OUTPUT and whose names are supplied in that SAME RECORD AREA clause. It is also a logical record of the file (in this clause) from which the most recent input occurred. This is equivalent to an implicit overwriting of all record areas, where the records are aligned on the leftmost character position.

General rules

1. If SAME AREA is used, only one file may be open at any given time.
2. If the RECORD phrase is used, all specified files may be open at the same time.
3. General rule 1 applies in the case of files that are specified in the SAME AREA clause as well as the SAME RECORD AREA clause.
4. The SAME [RECORD] AREA clause must not be specified for *external* files.

4.3 Language elements of the Data Division

FILE SECTION

Function

The FILE SECTION defines how the files are set up. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

Format

```
FILE SECTION.

[file-description-entry.
{record-description-entry}...]...
```

File description entries are discussed below. Record description entries are detailed in chapter 3 (page 141ff). Certain restrictions apply with respect to record sizes; these are listed in the clauses described below.

File description (FD) entry

Function

The file description (FD) entry specifies the physical structure and the record names for a given file.

A file description entry must be written for each file to be processed by the program. The information contained in this entry generally pertains to the physical aspects of the file, that is, the description of the data as it appears on the input or output medium.

Format

```
A    B      Margin indication
|    |
|    |
|    |
|    |
FD   file-name

     [BLOCK CONTAINS clause]
     [CODE-SET clause]
     [DATA RECORDS clause]
     [EXTERNAL clause]
     [GLOBAL clause]
     [LABEL RECORDS clause]
     [LINAGE clause]
     [RECORD clause]
     [RECORDING MODE clause]
     [VALUE OF clause].
```

Syntax rules

1. The level indicator FD identifies the beginning of a file description entry.
2. file-name must be identical to the file-name given in a SELECT clause.
3. Clauses that follow the file-name may appear in any order.
4. The file description entry must be followed by one or more record description entries.

General rules

1. The following table provides a summary of the functions of clauses used in file description entries.

Clause	Function
BLOCK CONTAINS	Specifies block size (physical record size).
CODE-SET	Defines character code set for data output to external devices.
DATA RECORDS	Specifies the names of the records in the file.
EXTERNAL	Declares a file as external
GLOBAL	Declares a file as global
LABEL RECORDS	Gives the names and values of the label records contained in a file.
LINAGE	Specifies the size of a logical page. Also permits definition of a page heading and a page footing.
RECORD	Specifies logical record size.
RECORDING MODE	Specifies the format of the logical records.
VALUE OF	Specifies the values of some data items of a label.

Table 4-1: Functions of file description clauses

2. The EXTERNAL and GLOBAL clauses are described in chapter 7, "Inter-program communication" (page 544 and page 547). The other file description clauses are described in alphabetical order below.

BLOCK CONTAINS clause

Function

The BLOCK CONTAINS clause specifies the maximum size of a physical block.

Format

```
BLOCK CONTAINS [integer-1 TO] integer-2 {CHARACTERS}
                                        {RECORDS   }
```

Syntax rules

1. A block must contain at least 20 characters and must not exceed a maximum length of 32763 characters.
2. The CHARACTERS or RECORDS phrase indicates whether block length is to be specified as a multiple of characters or logical records.
3. If neither CHARACTERS nor RECORDS is specified, CHARACTERS will be assumed.
4. integer-1 TO integer-2 indicates the number of characters or records in a given block, depending on the phrase used (either CHARACTERS or RECORDS).
5. When only integer-2 is specified, it refers to the maximum size of the block. When integer-1 and integer-2 are specified, they refer to the minimum and maximum size of the block, respectively. However, integer-1, when specified, is used only for documentation purposes and is treated as a comment by the compiler.

   The **maximum block size** specified by this clause has the following meaning:

   The blocks of the file must not be longer but can be shorter than the specified length. This is frequently the case with unblocked or blocked records whose length is variable.
6. When the CHARACTERS phrase is used, the block size is specified in terms of the number of characters contained within the block, regardless of the types of characters used to represent the data items within the block (see "USAGE clause", page 192). In this case, both integer-1 and integer-2 must include slack bytes and four characters for the record length field of each record of the block.
7. When the CHARACTERS phrase is used, and only integer-2 is specified, integer-2 indicates the exact length of the physical block. When both integer-1 and integer-2 are specified, they refer to the minimum or maximum physical block length, respectively.

8. When the RECORDS phrase is used, the block size is specified in terms of logical records. In this case, the compiler calculates the block size by multiplying the number of characters in the maximum size logical record by the value specified in integer-2. In the case of variable-length records, four characters are added for the record length field.
9. When the BLOCK CONTAINS clause is omitted, the compiler assumes that the records are not blocked, i.e. BLOCK CONTAINS 1 RECORDS is assumed. Consequently, the BLOCK CONTAINS clause may be omitted when **all** blocks of the file contain one, and only one, record.

General rules

1. The following table shows how the compiler calculates the block sizes in terms of characters and includes the appropriate phrases contained in the BLOCK CONTAINS clause.

   Legend for Table 4-2:

   F = fixed-length records
   V = variable-length records
   BL = block length
   RL = record length
   RL_{max} = maximum record length
   BLF = block length field (has the value 4)
   RLF = record length field (has the value 4)
   n = integer

Record format	BLOCK CONTAINS [integer-1 TO] integer-2	
	CHARACTERS	**RECORDS**
Fixed length	BL = integer-2 (integer-2 = n * RL)	BL = integer-2 *RL
Variable length	BL = integer-2 + BLF	BL = integer-2 * (RL_{max} + RLF) + BLF

Table 4-2: Calculation of the maximum block size

2. The CHARACTERS phrase should be used if the RECORDS phrase would produce a block size that is not accurate enough.
   Let us assume, for example, that a block contains 4 records consisting of one 50-character record and three 100-character records. If `BLOCK CONTAINS 4 RECORDS` is specified, and the maximum record length is defined as 100 characters, the compiler will calculate a block size of 4*(100+4)+4 = 420 characters. In this case, however, since each block actually requires only (50+4)+3*(100+4)+4 = 366 characters, the exact block size could be specified by using the following clause instead:
   `BLOCK CONTAINS 366 CHARACTERS.`
3. If the BLOCK CONTAINS clause is specified for an *external* file, the BLOCK CONTAINS clause must be specified in all programs that describe this external file; the block length calculated from the information specified in the BLOCK CONTAINS clause must be the same, regardless of whether it results from the number of "RECORDS" or the number of "CHARACTERS".

CODE-SET clause

Function

The CODE-SET clause specifies the character code set used to represent data on the external media.

Format

```
CODE-SET IS alphabet-name
```

Syntax rules

1. When the CODE-SET clause is specified for a file, all data in that file must be described as USAGE IS DISPLAY, and any signed numeric data must be described with the SIGN IS SEPARATE clause.
2. The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase (see "SPECIAL-NAMES paragraph", page 125).

General rules

1. If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from/to the native character codes. This code conversion occurs during the execution of an input or output operation (see "SPECIAL-NAMES paragraph", page 125).
2. If the CODE-SET clause is not specified, the native character code set (EBCDIC) is assumed for data on the external media.
3. If the CODE-SET clause refers to an *external* file, an identical CODE-SET clause must be specified in all programs that describe this external file.

DATA RECORDS clause

Function

The DATA RECORDS clause is used only for documentation. It specifies the names of the records in a file.

Format

```
DATA {RECORD  IS } {data-name-1}...
     {RECORDS ARE}
```

Syntax rules

1. data-name-1 is the name of a record. It must be preceded by level number 01 in the file description entry.
2. The presence of more than one data-name indicates that the file contains more than one type of record. These records may be of differing sizes, formats etc. The order in which they are listed is not significant.

LABEL RECORDS clause

Function

The LABEL RECORDS clause specifies whether labels are present, and identifies them if they are.

Format

```
LABEL { RECORD  IS } { OMITTED           }
      { RECORDS ARE} { STANDARD          }
                     { {data-name-1}...  }
```

Syntax rules

1. For data-name-1, record description entries must be present for the file concerned. data-name-1... must not appear as operands in the DATA RECORDS clause of this file. The data-name may optionally be used to specify, in the LINKAGE SECTION, a record used for label handling.
2. The OMITTED phrase specifies that there are either no unique labels for this file, or the existing labels are non-standard, and the user does not wish to use a USE procedure for label processing (e.g. he may want to process the labels as records).
3. The STANDARD phrase specifies that labels are present for the file and that these labels are in accordance with system conventions (see the "DMS" manual [9]).
4. In the following discussion, all references to data-name-1 also apply to data-name-2, etc.

   When data-name-1 is specified, this indicates either the presence of user labels in addition to standard labels, or the presence of nonstandard labels. data-name-1 defines the name of a user label record.

   When processing user labels, data-name-1 may be defined for any files except unit-record files.

General rules

1. OMITTED may not be specified for files which are assigned with "ASSIGN TO literal".
2. For the format of system labels, see the "DMS" manual [9].
3. User labels are formatted as follows:

   a) Each user label is 80 characters long.

b) Positions 1-3 of a user header label must contain the characters UHL.

c) Positions 1-3 of a user trailer label must contain the characters UTL.

d) Position 4 shows the relative location of the label in a sequence of header or trailer labels; in other words, this position must contain a digit from 1 to 9. If only one label (UHL and/or UTL) is present, position 4 must contain the character "1".

e) Positions 5-80 are formatted according to user specifications.

For further details, see the "DMS" manual [9].

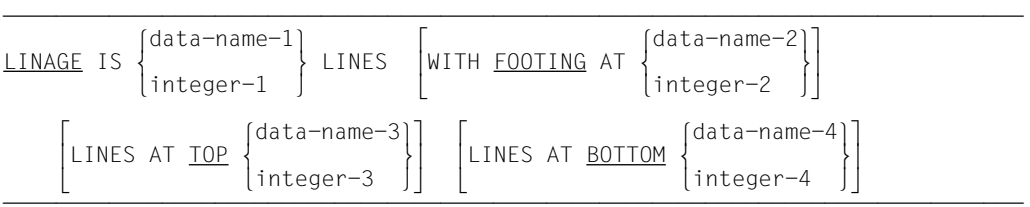
4. User header labels follow standard file header labels of the system, however, they precede the first record.
5. User trailer labels follow standard end-of-file labels of the system.
6. Nonstandard labels can be from 1 to 4095 characters long. Their format and contents are defined by the user.
7. If data-name-1 is specified, then all Procedure Division references to the specified data-names or to items subordinate to these data-names must appear within user declaratives (USE procedures).
8. If the LABEL RECORDS clause refers to an *external* file, an equivalent LABEL RECORDS clause must be specified in all programs that describe this external file.

LINAGE clause

Function

The LINAGE clause provides a means of specifying, for an output file, the size of a logical page in terms of the number of lines. It can also be used to specify the size of the top and bottom margins on the logical page and the line number, within the page body, at which the footing area is to begin.

Format



Syntax rules

1. data-name-1, data-name-2, data-name-3, and data-name-4 must be elementary unsigned numeric integer data items.
2. data-name-1, data-name-2, data-name-3, and data-name-4 may be qualified.
3. The value of integer-1 or of the data item referenced by data-name-1 must be greater than zero.
4. The value of integer-2 or of the data item referenced by data-name-2 must be greater than zero, but not greater than integer-1 or the value of the data item referenced by data-name-1.
5. The value of integer-3 and integer-4, or the data items referenced by data-name-3 and data-name-4, may be 0.
6. The LINAGE clause is not permitted for files which are opened with OPEN EXTEND.
7. The LINAGE clause is permitted only for files which are assigned to PRINTER literal-1 or literal-2.

8. The LINAGE clause provides a means of specifying the size of a logical page in terms of the number of lines. The logical page size is the sum of the values of each phrase in the LINAGE clause except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the value for this function is zero. If the FOOTING phrase is not specified, no footing area is defined (see Table 4-3).

top margin LINES AT TOP	data-name-3/ integer-3 > = 0 (default = 0)
page body LINAGE IS footing area WITH FOOTING AT	data-name-1/ integer-1 → 1, 2, 3, ., ., ., n → Number of print lines data-name-2/ (default = 0 footing lines) integer-2 . . . data-name-1/ integer-1
bottom margin LINES AT BOTTOM	data-name-4 / integer-4 > = 0 (default = 0)

Table 4-3 Setup of a logical page

9. The size of a logical page must not necessarily correspond to the size of a physical page.
10. The value of integer-1 or the data item referenced by data-name-1 specifies the number of lines that can be written and/or spaced on the logical page. This part of the logical page, in which these lines can be written and/or spaced, is called the page body.
11. The value of integer-3 or the data item referenced by data-name-3 specifies the number of lines forming the top margin of the logical page. This area is left blank.
12. The value of integer-4 or the data item referenced by data-name-4 specifies the number of lines forming the bottom margin of the logical page. This area is left blank.
13. The value of integer-2 or or the data item referenced by data-name-2 specifies the line number within the page body at which the footing area begins.
14. The footing area comprises the area of the logical page between the line specified by integer-2 or data-name-2 and the line represented by integer-1 or data-name-1.

15. The values of integer-1, integer-3 and integer-4 (or the values of data items data-name-1, data-name-3 and data-name-4, if specified) are used at object time by an OPEN statement (with the OUTPUT phrase) to specify the number of lines in each of the indicated parts of the first logical page. At the same time, the value of integer-2 or of data item referenced by data-name-2 (if specified) is used to define the footing area.

    If a page overflow occurs during execution of a WRITE statement with the ADVANCING phrase, the values of integer-1, integer-3 and integer-4 are used to specify the number of lines that comprise each of the indicated sections of the next logical page.

    The value of integer-2 or of data item referenced by data-name-2 (if specified) is then used to define the footing area of the next logical page.

General rules

1. The COBOL register LINAGE-COUNTER is generated whenever a LINAGE clause occurs. The LINAGE-COUNTER value at any given time represents the line number at which the printer is positioned within the current page body. The first printable line on each logical page has the number 1.

    A separate LINAGE-COUNTER is supplied for each file described in the FILE SECTION whose file description entry contains a LINAGE clause.

2. The LINAGE-COUNTER may be accessed, but must not be modified, by Procedure Division statements. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.

3. The LINAGE-COUNTER is automatically updated during the execution of a WRITE statement for the file:

    a) If the ADVANCING PAGE phrase is specified in the WRITE statement, the LINAGE-COUNTER is automatically reset to the value 1.

    b) If ADVANCING integer or ADVANCING identifier-2 is specified in the WRITE statement, the LINAGE-COUNTER is incremented by integer or by the value of the data item referenced by identifier-2.

    c) If the ADVANCING phrase is omitted in the WRITE statement, the LINAGE-COUNTER is automatically incremented by the value 1.

    d) The LINAGE-COUNTER is automatically reset to the value 1 when the printer is positioned to the first line to be written on the next logical page (see "WRITE statement", page 420).

    e) The LINAGE-COUNTER is automatically set to the value 1 at the time an OPEN statement is executed for the file.

4. If the LINAGE clause refers to an *external* file, an equivalent LINAGE clause must be specified in all programs that describe this external file. In contrast to the Standard, the compiler described here requires specifications only of the same type (i.e. either only data-name specifications or only integer specifications). The contents of the data items or the numerical values may be different.

RECORD clause

Function

The RECORD clause defines the length of the records in a file and controls the external record format (see "RECORDING MODE clause", page 394).

Format 1 Indicates fixed-length records by specifying the number of character positions in a record.

Format 2 Indicates variable-length records, whereby the size of the record must lie within a certain range.

Format 3 Indicates variable-length records, whereby the minimum and maximum number of character positions are specified.

Format 1

```
RECORD CONTAINS integer-1 CHARACTERS
```

Syntax rules

1. The length of each record is precisely defined by its record description entry. The length specification in the RECORD clause is ignored in this respect.
2. The number of character positions in each record description entry for the file must be equal to integer-1.
3. integer-1 must be at least 1 and can be as large as 32767. When the RECORD clause is used in a sort file description entry, the maximum permitted value for integer-1 is equal to 32755 minus the sort key length.

Format 2

```
RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]

    [DEPENDING ON file-name-1]
```

Syntax rules

1. In any record description entry, it is impermissible for the length specification to be less than integer-2 or more than integer-3.

2. integer-3 must be larger than integer-2.
3. integer-2 must be at least 1, while integer-3 can be up to 32763. When the RECORD clause is used in a sort file description entry, the maximum permitted value for integer-3 is equal to 32751 minus the sort key length.
4. data-name-1 must be described as an unsigned integer data item in the WORKING-STORAGE or LINKAGE SECTION.

General rules

1. If integer-2 is omitted, it is assumed that the length of the shortest record is 1. For each SORT or MERGE statement, the sort key must lie completely within the range of this minimal length.
2. If integer-3 is omitted, the length of the longest record in the record description entry for this file is assumed. This number also provides the block size.
3. If data-name-1 is specified, the number of character positions in the record must be moved to data-name-1 before a RELEASE, REWRITE or WRITE statement is executed for this file.
4. If data-name-1 is specified, its contents remain unchanged when a RELEASE, REWRITE or WRITE statement is executed or when a READ or RETURN statement terminates abnormally.
5. During execution of a RELEASE, REWRITE or WRITE statement, the record length is determined as follows:
   a) If data-name-1 is specified: by the contents of the data item referenced by data-name-1.
   b) If data-name-1 is omitted and the record description entry has no OCCURS clause with DEPENDING ON phrase: by the number of character positions in the record.
   c) If data-name-1 is not specified and the record description entry contains no OCCURS clause with DEPENDING ON phrase: by the fixed portion of the record description entry (i.e. all data description entries without the DEPENDING ON phrase) and by the number of occurrences of the table elements during execution of the statement.
6. If data-name-1 is specified and a READ statement has been successfully executed, data-name-1 contains the number of character positions of the record just read. This number is, however, not greater than integer-3, but it may be larger than the largest record description entry.
7. If INTO is specified in a READ or RETURN statement, the number of character positions in the current record, which is the source field in the implicit MOVE, is determined as follows:

a) If data-name-1 is specified: by the contents of the data item referenced by data-name-1.

b) If data-name-1 is not specified: by the value that would have been transferred had data-name-1 been specified.

Format 3

```
RECORD CONTAINS integer-4 TO integer-5 CHARACTERS
```

Syntax rules

1. integer-4 describes the number of character positions in the shortest record; integer-5 the number in the longest.
2. In no record description entry for the file may the length be less than specified in integer-4 or greater than specified in integer-5.
3. integer-5 must be greater than integer-4.
4. integer-4 must be at least 1; integer-5 must not exceed 32763. When the RECORD clause is used in a sort file description entry, the maximum permitted value for integer-5 is 32751 minus the sort key length.

General rules for all formats

1. The length of each record is precisely defined by its record description entry. If the RECORD clause is specified, the length is compared with the specifications given in the clause (see also "RECORDING MODE clause", page 394).
2. The length of a record is determined by the sum of the character positions of all its elementary items and the slack bytes generated by the compiler. If the record contains a table, the minimum or maximum number of table elements is taken into account when calculating the length. For further details see SYNCHRONIZED clause (page 189), USAGE clause (page 192) and "Data alignment" (page 73).
3. If the RECORD clause is not specified, the minimum or maximum record length results from the specifications in the corresponding record description entry. If a record description entry contains an OCCURS clause with the DEPENDING phrase, the value 1 is assumed as the minimum record length for this.
4. If the RECORD clause is specified for an *external* file, a RECORD clause of the same format must be specified in all programs that describe this external file; the mimimum and maximum record lengths calculated from the specifications in the RECORD clause must match those from the corresponding record description entries.

5. The following table shows which specifications in the formats of the RECORD clause are decisive in calculating the mimimum and maximum record lengths.

	Format 1	**Format 2**	**Format 3**	**No RECORD clause**
Minimum record length	Longest record of record description entry	integer-2; if not specified: 1	integer-4	Shortest record of record description entry; with OCCURS DEPENDING: 1
Maximum record length	as above	integer-3; if not specified: longest record of record description entry	integer-5	Longest record of record description entry

RECORDING MODE clause

Function

The RECORDING MODE clause specifies that the format of logical records in the file is "undefined".

Format

```
RECORDING MODE IS U
```

Syntax rule

Specifying U means that the file may contain any combination of fixed-length or variable-length records. Records in mode U cannot be blocked and have no preceding control data item (record length field, RLF). If RECORDING MODE IS U is specified, there is no need to specify the BLOCK CONTAINS clause.

General rules

1. The record formats "F" (fixed-length records) and "V" (variable-length recoreds) are defined by the RECORD clause:

   fixed record length by a RECORD clause in format 1, variable record length by a RECORD clause in format 2.

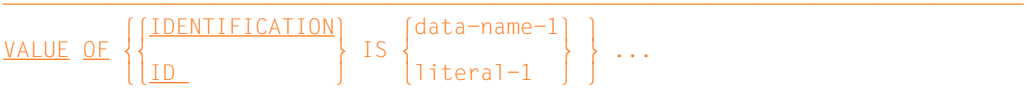
2. If the RECORDING MODE clause refers to an *external* file, an equivalent RECORDING MODE clause must be specified in all programs that describe this external file.

VALUE OF clause

Function

The VALUE OF clause particularizes the description of data items in a label record.

Format



The VALUE OF clause is treated as a comment by the compiler.

4.4 Language elements of the Procedure Division

4.4.1 Input/output statements

In COBOL, input and output are record-oriented. Thus the READ, WRITE and REWRITE statements process records. The COBOL user is therefore only concerned with the processing of individual records. The following operations are performed automatically:

– moving data into input/output areas (buffers) and/or internal storage

– validity checking

– correcting errors (where feasible)

– blocking/unblocking

– switching volumes.

Note

The description of input/output statements uses the expressions "volume" and "reel". Volume may be used for all input/output devices; reel is applicable to magnetic tape devices only. The handling of disk storage files with sequential access is logically equivalent to the handling of magnetic tape files.

Overview

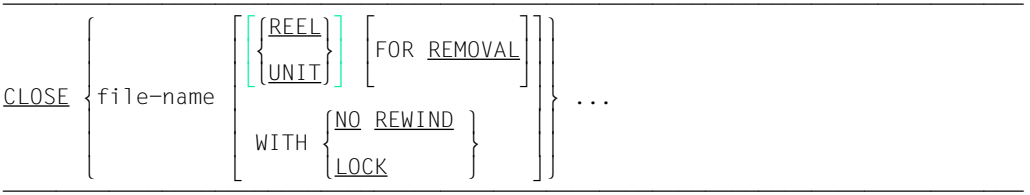
Statement	Function
CLOSE	Terminates processing of a reel, unit or file.
OPEN	Opens a file for processing.
READ	Reads a record.
REWRITE	Replaces a record on a disk storage file by a specified record.
USE	In addition to input/output statements, USE statements may be supplied to indicate label and error handling procedures (see "DECLARATIVES", page 213).
WRITE	Writes a record.

CLOSE statement

Function

The CLOSE statement terminates the processing of input/output reels, units and files, with optional rewind and/or lock where applicable.

Format



Syntax rule

1. The files referenced in the CLOSE statement need not have the same organization or access type.
2. In the case of line sequential files only the specification WITH LOCK is permitted.

General rules

1. A CLOSE statement may be issued only for a file which is open.
2. The meanings of the options in the CLOSE statement are given below:

REEL	The current tape reel is to be closed.
UNIT	The current mass-storage unit is to be closed.
NO REWIND	No repositioning to the beginning of the reel is to be performed after a tape file is closed.
LOCK	The file cannot be reopened in the same program run.
REMOVAL	The current reel of a magnetic tape file is to be unloaded.

3. If not specified otherwise, the terms REEL and UNIT are equivalent and are fully interchangeable within a CLOSE statement. The handling of sequential disk storage files is logically equivalent to the handling of files on magnetic tape or similar sequential media.

4. In order to show the effect of various CLOSE statements as applied to various storage media, all input/output files are divided into the following categories:

   a) **UNIT RECORD volume file (unit record file):**
   a file assigned to an input or output medium for which the concepts rewinding, units, and reels have no meaning.

   b) **Sequential single-volume file:**
   a sequential file that is entirely contained on one volume. There may be more than one file on this volume.

   c) **Sequential multivolume file:**
   a sequential file contained on more than one volume.

5. The results of executing each CLOSE statement for each type of file are summarized in the following table:

CLOS statement	**File type**		
	Unit record	Sequential single-volume	Sequential multivolume
CLOSE	C	C, G	C, G, A
CLOSE WITH LOCK	C, E	C, G, E	C, G, E, A
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

Table 4-4: Relationship between types of sequential files and the formats of the CLOSE statement

The definitions of the symbols used in the table are given below. When the definition of the symbol depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to files opened as INPUT, OUTPUT and I-O.

A **Previous volumes unaffected**

All volumes of the file processed prior to the current volume were processed according to standard volume switching procedures, except those volumes controlled by a prior CLOSE REEL/UNIT statement.

B **No rewind of current reel**

The current volume is positioned at the logical end of the file on the volume.

C **Standard close file**

For files opened with the INPUT or I-O phrase:

if the file is positioned at its end and a LABEL RECORDS clause was supplied, then (if a USE procedure is present) the standard trailer label procedure and user trailer label procedure are executed. The order in which these two routines are executed is specified by the USE procedure. The standard system closing procedures are then performed.

If the file is positioned at its end, but a LABEL RECORDS clause was not supplied, then only the standard system closing procedures are performed.

If the file is positioned other than at its end, only the standard system closing procedures are performed. Even if USE procedures are supplied, no label processing will take place in this case. (An INPUT or I-O file is considered to be at its end if the imperative statement in the AT END phrase of the READ statement, if entered, has been executed, and no CLOSE statement has been executed.)

For files opened with the OUTPUT phrase:

if a LABEL RECORDS clause was specified for this file, the standard ending label procedures and the user ending label procedures (provided such were specified by a USE procedure) are performed. The order in which these two procedures are executed is defined by the USE statement. The standard system closing procedures are then performed.

If label records are not specified for the file, only the standard system closing procedures are performed.

D **Unload current reel**

The REMOVAL option, when specified, causes the current reel of a magnetic tape file to be unloaded. Further processing of the file requires an appropriate continuation reel. After executing a CLOSE statement without the REEL/UNIT phrase specified, and an OPEN statement for this file, this reel may be processed again.

E **Standard file lock**

If the LOCK phrase is used, the file cannot be reopened in the same program run.

F **Standard close volume**

The following operations are performed for files opened with the INPUT or I-O phrase:

1. When the current volume is the last or only volume for the file:

   Disk storage files

   Processing of the current data block is terminated. The next READ statement makes the first record of the following data block available.

   Tape files

   Processing of the reel is terminated. The next READ statement leads to an at end condition.

2. When other volumes exist for the file:

   a) Volume swapping.

   b) USE procedures for processing standard and user header labels (if specified by a USE statement). The order in which these procedures are executed is defined by the USE statement.

   c) The next record on the new volume is made available for the subsequent read operation.

For files opened as OUTPUT, the following operations are carried out:

1. *Disk storage files*

Processing of the current data block is terminated. The next WRITE statement creates a new data block.

2. *Tape files*

   a) Standard and user trailer label procedures (if specified by a USE statement). The order in which these procedures are executed is defined by the USE statement.

   b) Volume swapping.

   c) Standard and user header label procedures (if specified by a USE statement). The order in which these procedures are executed is defined by the USE statement.

G **Rewind**

The current volume is positioned at its beginning (for a magnetic tape file, this is the beginning of the reel; for disk storage files, this is the beginning of the file concerned).

H I/O status 07 is set.

The optional phrases are ignored.

J I/O status 07 is set.

The CLOSE statement is ignored.

6. The execution of a CLOSE statement causes the contents of the data item specified in the FILE STATUS clause (if such a clause exists) to be updated (see also "FILE STATUS clause", page 365).
7. If an optional input file does not exist, no end-of-file processing and no reel/volume processing is executed.
8. If more than one file-name has been specified, the effect is the same as if a separate CLOSE statement had been written for each file-name.

OPEN statement

Function

The OPEN statement opens files for processing and performs label checking and output.

Format

```
      ⎧ INPUT {file-name-1  ⎡REVERSED        ⎤}...⎫
      ⎪       ⎩             ⎣WITH NO REWIND  ⎦⎭    ⎪
OPEN  ⎨                                            ⎬...
      ⎪OUTPUT {file-name-2 [WITH NO REWIND]}...    ⎪
      ⎪I-O {file-name-3}...                        ⎪
      ⎩EXTEND {file-name-4}...                     ⎭
```

Syntax rules

1. REVERSED may be specified only for sequential files.
2. The EXTEND phrase must not be used for multi-file volumes (see also "I-O-CONTROL paragraph", page 370).
3. EXTEND may be specified only for files for which no LINAGE clause has been specified.
4. A file-name must not appear more than once in an OPEN statement.
5. The I-O phrase is permitted for disk storage files only.
6. The files specified in a given OPEN statement may have different organizations or access modes.
7. In the case of line sequential files, the only permissible open modes are OPEN INPUT and OPEN OUTPUT without the specifications REVERSED and NO REWIND.

General rules

1. After successful execution of an OPEN statement, the availability of the file specified by file-name-1 is established, and the file is in the open mode.
2. After successful execution of an OPEN statement, the associated record area is available to the program. Only one record area exists for an external file; this area is available to all programs that describe this file.
3. Before the first successful execution of an OPEN statement for a file, no statement (other than a SORT/MERGE statement with the USING or GIVING phrase) must be executed that would either explicitly or implicitly reference that file.

4. The following table lists the statements permissible in OPEN mode (X indicates "permitted"):

Statement	**OPEN mode**			
	Input	Output	I-O	Extend
READ	X		X	
REWRITE			X	
WRITE		X		X

5. All of the phrases INPUT, OUTPUT, I-O or EXTEND may be used, within the same program, in different OPEN statements for a given file. Before executing another OPEN statement after the logically first one for the same file in a program, a CLOSE statement must have been performed; at this time, REEL/UNIT or LOCK must not be specified in the CLOSE statement.
6. INPUT means that the file is to be processed as an input file (input mode).
7. OUTPUT indicates that the file is to be processed as an output file (output mode).
8. I-O indicates that input and output operations (i.e. reading and updating operations) are to be performed on the file. Since this phrase implies the existence of the file, it cannot be used if the disk storage file is being initially created (update mode).
9. The EXTEND phrase, when specified, causes the file to be positioned immediately behind the last logical record of the file. Subsequent WRITE statements for that file cause records to be added to the file in the same manner as if it had been opened with the OUTPUT phrase specified (extend mode).
10. REVERSED indicates that records in the file are to be read in reversed order, i.e. starting with the last record of the file.

    The REVERSED phrase should not be specified for a file with nonstandard labels unless the last such label is followed by a tape marker. In all other cases, the system will read label records as if they were records.
11. NO REWIND may be specified for a tape file or for a disk storage file which was opened OUTPUT. The actions taken for tape and disk storage files are listed below:

    Tape files:

    NO REWIND indicates that the tape reel is not to be rewound when the file is opened.

    Disk storage files:

    NO REWIND indicates that no records existing on the file are to be overwritten. NO REWIND exists only for reasons of compatibility. OPEN OUTPUT WITH NO REWIND has the same function as OPEN EXTEND.

12. If the storage medium containing the file permits the rewind function, the following rules apply:

    a) If neither the REVERSED, nor the EXTEND, nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

    b) If the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; instead, the file is expected to be at its beginning already.

    c) If the REVERSED phrase is specified, the records of the file are made available in reversed order; that is, starting with the last record of the file.

13. If an optional file does not exist, successful execution of an OPEN statement with EXTEND or I-O specified causes the desired file to be created.

14. Label handling procedures are performed by the OPEN statement for all files, except those for which LABEL RECORDS ARE OMITTED is specified in the file description entry.

    If the INPUT phrase of the OPEN statement is specified, the following steps are taken:

    a) If system labels are present, they are checked.

    b) If user labels or nonstandard labels are present and an applicable USE procedure is declared, that USE procedure is executed.

    c) The file is positioned so that the first record can be read.

    If the OUTPUT phrase of the OPEN statement is supplied, execution takes place as follows:

    d) If standard labels are specified for the file, they are created and written.

    e) If an applicable USE procedure is declared, it is executed; and user or nonstandard labels are written.

    f) The record area is made available to the program in order to receive data.

    If the I-O phrase of the OPEN statement is specified, the following steps are taken:

    g) If system labels are present, they are checked.

    h) If user labels are present and an applicable USE procedure is declared, that USE procedure is executed.

    i) The file is positioned for data read or replacement operations.

    If the EXTEND phrase as well as the LABEL RECORDS STANDARD/data-name clause are specified for an OPEN statement, then execution of the OPEN statement involves the following steps:

j) The file header labels are processed only if it is a single-reel or single-volume file.

k) The reel/volume header labels on the last existing reel/volume are processed as in the case of the OPEN statement with INPUT phrase.

l) The existing file trailer labels are processed as in the case of the OPEN statement with INPUT phrase. These label records are then deleted.

m) Further processing is then the same as with OPEN OUTPUT.

15. After execution of an OPEN statement, the contents of the data item indicated in the FILE STATUS clause (if specified) will be updated (see also "FILE STATUS clause", page 365).
16. If more than one file-name is specified, the result is the same as if a separate OPEN statement had been written for each file-name.
17. The minimum and maximum record lengths for a file are defined when the file is created and must not be modified subsequently.

READ statement

Function

The READ statement makes the next logical record from a file available to the program.

Format

```
READ file-name [NEXT] RECORD [INTO identifier]

     [AT END imperative-statement-1]

     [NOT AT END imperative-statement-2]

     [END-READ]
```

Syntax rules

1. file-name and identifier must not reference the same storage area.
2. If no USE procedure is declared for the file, AT END must be specified in the READ statement.

General rules

1. The NEXT phrase is optional and has no effect on the execution of a READ statement.
2. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.
3. The execution of a READ statement causes the contents of the data item specified in the FILE STATUS clause of the related file description entry to be updated (see "FILE STATUS clause", page 365).
4. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current record are undefined after execution of the READ statement.
5. The INTO phrase may be specified:
   - if only one record description is subordinate to the file description entry, or
   - if all record-names associated with the file-name and the data item referenced by the identifier represent group items or alphanumeric elementary items.

6. The execution of a READ statement with the INTO phrase is equivalent to:

```
READ file-name
MOVE record-name TO identifier
```

   The MOVE operation takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the record is available both in the input area and in the area specified by the identifier. The length of the source field is determined by the length of the record that is read (see RECORD clause, page 390).
   If the file description entry contains a RECORD-IS-VARYING clause, the implicitly executed transfer operation is a group item transfer.
   If the execution of the READ statement was unsuccessful, the implied MOVE statement does not occur.
   The index for the identifier is calculated after execution of the READ statement and immediately before the data transfer.

7. If the input area is to be explicitly referenced following a READ statement without the INTO phrase, it is the user's responsibility to ensure that the correct record description entry (i.e. corresponding to the length of the read record) is used.

8. If there is no next logical record, or if an optional input file does not exist, the following occurs in the order specified:

   a) The I-O status (FILE STATUS) associated with file-name is set to indicate the at end condition.

   b) If the AT END phrase is specified, control is transferred to imperative-statement-1 in the AT END phrase. Any USE procedure declared for file-name is not executed.

   c) If the AT END phrase is not specified, a USE procedure must be declared. This procedure is then executed.

   If the at end condition occurs, execution of the READ statement is unsuccessful.

9. If an at end condition does not occur during the execution of a READ statement, the AT END phrase is ignored, if specified, and the following actions occur:

   a) The I-O status for file-name is updated.

   b) If some other exception condition occurs, control is transferred to the USE procedure.

   c) If no exception condition exists, the record is made available in the record area, and any implicit move as a result of the INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2 of the NOT AT END phrase, if specified. In the latter case, execution continues according to the rules for the specified imperative-statement. If a procedure branching or conditional statement that causes explicit transfer of control is executed, control is transferred in ac-

cordance with the rules for that statement; otherwise, upon completion of the execution of the imperative-statement-2 specified in the NOT AT END phrase, control is transferred to the end of the READ statement.

10. If the physical end of a reel or disk storage volume is encountered during execution of a READ statement for a multi-volume file, then the following operations take place:

    a) The standard volume trailer label routine and, if specified by an appropriate USE procedure, the user volume trailer label routine are executed. The order in which the two routines are performed is specified by the USE procedure.

    b) A volume switch is executed.

    c) The standard volume header label routine and, if specified, the user volume header label routine are executed. Again, the order in which these routines are executed is specified by the USE procedure.

11. Following an unsuccessful attempt to perform a READ statement, the contents of the input area associated with the file as well as the file position indicator are undefined.

12. If the number of character positions in a record is less than the minimum length specified in the record descriptions, the contents to the right of the last valid character will be unpredictable.
    If the number of character positions is greater than the maximum length specified in the record descriptions, the record will be truncated to the right of the maximum length.
    In either case the read operation is successful, but a FILE STATUS is set to indicate the occurrence of a record length conflict.

13. After the at end condition has occurred, no READ statement may be issued for the file until a successful CLOSE statement has been carried out, followed by a successful OPEN statement for that file.

14. END-READ delimits the scope of the READ statement.

REWRITE statement

Function

The REWRITE statement replaces a logical record on a disk storage file.

Format

```
REWRITE record-name [FROM identifier]  [END-REWRITE]
```

Syntax rules

1. record-name and identifier must not refer to the same storage area.
2. record-name must be associated with a file description (FD) entry in the Data Division of the program and may be qualified.
3. The REWRITE statement is not permitted for line sequential files.

General rules

1. record-name identifies the record to be replaced.
2. The file associated with record-name must be a disk storage file and must be open in the I-O mode at the time the REWRITE statement is executed.
3. The record addressed by the record name must be of the same length as the record to be replaced on the file.
4. If the number of character positions specified in the record referenced by record-name is not equal to the number of character positions in the record being replaced, execution of the REWRITE statement is unsuccessful, the updating operation does not take place, the content of the record area is unaffected and the I-O status of the file is set to a value indicating that the record limits have been exceeded.
5. Execution of a REWRITE statement with the FROM phrase is equivalent to execution of the following statements:

   ```
   MOVE identifier TO record-name
   REWRITE record-name.
   ```

   The content of the storage area described by record-name before the implicit MOVE statement is performed has no effect on the execution of the REWRITE statement.

   Thus, when the FROM phrase is used, data is transferred from identifier to record-name and then released to the appropriate file.

Identifier may be used to reference any data outside the current file description entry.

Transfer of data through the implicit MOVE statement takes place in accordance with the rules for a MOVE statement without the CORRESPONDING phrase. After the REWRITE statement is executed, the content of the record is still available in the area specified by identifier, although it is no longer available in the area specified by record-name.

6. A REWRITE statement must be preceded by a successful READ statement as the last input/output statement for the associated file.
7. Execution of the REWRITE statement causes the record accessed by the preceding READ statement to be replaced on that file.
8. The record rewritten by a successful REWRITE statement is no longer accessible in the record area; an exception to this rule is the use of the SAME RECORD AREA clause. In this case, the record is available to all other files specified in the SAME RECORD AREA clause as well as to the current file.
9. Rewriting a record does not cause the record contents on the associated disk storage file to be changed until the next block of the file is read or that file is closed.
10. Execution of a REWRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS clause", page 365).
11. END-REWRITE delimits the scope of the REWRITE statement.

USE statement

Function

The USE statement introduces declarative procedures and defines the conditions for their execution. The USE statement itself, however, is not executed.

Format 1 declares label handling routines.

Format 2 declares procedures to be run if an input/output error occurs for a file.

Format 1

Syntax rules for format 1

1. file-name-1 must be defined in a file description (FD) entry of the program's Data Division.
2. file-name-1 must not be the name of a sort file.
3. file-name-1 refers to the file description entry for which the specified label handling procedures are to be performed.
4. The procedures specified are executed in conjunction with OPEN and CLOSE statements for the file.
5. Once a USE procedure has been executed, program execution resumes with the calling routine.
6. BEFORE or AFTER indicates the types of labels to be processed.

   BEFORE indicates that nonstandard labels are to be processed (such labels may be specified only for tape files).

   AFTER indicates that user labels follow standard file labels and that these user labels are to be processed.
7. BEGINNING or ENDING indicates that header or trailer labels, respectively, are to be processed.

   If the BEGINNING and ENDING phrases are omitted, the declared procedures will be run for both header and trailer labels.
8. The phrases REEL, UNIT or FILE indicate that the declared procedures are to be run if volume, reel, or file labels are present.

The REEL phrase is not applicable to disk storage files. The UNIT phrase is not applicable to files in the random access mode, since only file labels are processed in that mode. The compiler treats the REEL and UNIT phrases as interchangeable.

9. If the above phrases are omitted, the declarative procedures are executed depending on the type of volume involved, either for reel labels and file labels, or for volume labels and file labels.
10. Format 1 of the USE statement is not permitted for line sequential files.

General rules for format 1

1. The labels to be processed for a file must be specified within the file description entry of that file as data-names in the LABEL RECORDS clause. These labels must be defined as level-01 data items within the file description entry or the LINKAGE SECTION.
2. The same file-name may appear in more than one variant of the format-1 USE statement. However, this must not cause two or more declarative procedures to be initiated at the same time.
3. If the file-name-1 specification is used, the file description for the file-name must not specify a LABEL RECORDS clause with the OMITTED phrase.
4. No user label routines can be declared for *external* files.
5. The standard system procedures are performed on all standard label records consisting of system and user labels.
   a) Labels on input or input/output files are checked in the following order:
      1. The I-O system checks standard header labels.
      2. The USE procedures (if any) check user header labels.
      3. The I-O system checks standard trailer labels.
      4. The USE procedures (if any) check user trailer labels.
   a) Labels on output files are created in the following order:
      1. The I-O system creates standard header labels.
      2. The USE procedures (if any) create user header labels.
      3. Before the user header labels are written, they are checked to see whether they begin with the string UHL.
      4. The I-O system creates standard trailer labels.
      5. The USE procedures (if any) create user trailer labels.
      6. Before the user trailer labels are written, they are checked to see whether they begin with the string UTL.

6. Within a USE procedure, there must be no reference to nondeclarative procedures except for the PERFORM statement.

   References to procedure names which are subordinate to a USE statement may be made from another procedure or from a nondeclarative procedure by using a PERFORM statement only.

7. The exit from a format-1 declarative procedure is generated by the compiler following the last statement in a given section. All logical paths within that section must lead to this exit point.

8. There is one exception to general rule 7:

   The GO TO statement with the MORE-LABELS phrase may be used as a special exit point. After this statement is executed, the runtime system will take one of the following actions:

   a) If labels are currently being created, the system will write the current header or trailer labels; the program will then continue at the beginning of the declarative section in order to create more labels. The user must make sure that the last statement in the section is executed after all labels are created. At this point, it should be noted that after execution of the GO TO statement with the MORE-LABELS phrase, a label will be written in any case; if the user did not supply a new label, the runtime system will generate a dummy label record in analogy to general rule 7.

   b) If labels are currently being checked, the system will read an additional header or trailer label; the program will then continue at the beginning of the declarative section in order to check more labels. However, when processing user labels, the system will reenter the section only when there is another user label to check. Thus, in this case, the programmer does not have to provide a program path that flows through the last statement in the section. On the other hand, when processing nonstandard labels, the system does not know how many labels exist. The last statement in the section must therefore be executed in this case in order to terminate nonstandard label processing.

Example 4-1

for format 1

In this example, one declarative section (ALPHA) handles header labels and another (BETA) handles trailer labels. The declarative procedures are executed both in input and output modes.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. USESEQ.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SAMPLE-FILE ASSIGN TO "TESTTAPE".
DATA DIVISION.
FILE SECTION.
FD  SAMPLE-FILE,
    RECORD CONTAINS 100 CHARACTERS,
    LABEL RECORD IS SAMPLE-LABEL,
    DATA RECORD IS SAMPLE-RECORD.
01  SAMPLE-LABEL.
    02  LABEL-ID                PICTURE X(4).
    02  LABEL-INFO              PICTURE X(76).
01  SAMPLE-RECORD               PICTURE X(100).
WORKING-STORAGE SECTION.
77  I-O-INDICATOR               PICTURE X.
    88  INPUT-MODE              VALUE "I".
    88  OUTPUT-MODE             VALUE "O".
77  LABEL-COUNTER               PICTURE 9.
PROCEDURE DIVISION.
DECLARATIVES.
ALPHA SECTION.
    USE AFTER STANDARD BEGINNING FILE
        LABEL PROCEDURE ON SAMPLE-FILE.
ALPHA-1.
    IF INPUT-MODE
    THEN
      DISPLAY "512010 THIS LABEL READ:-" SAMPLE-LABEL
              UPON T
      GO TO MORE-LABELS
    ELSE
      IF LABEL-COUNTER = 0
      THEN
        MOVE "UHL1" TO LABEL-ID
        MOVE "THIS WAS PRODUCED BY ALPHA SECTION."
             TO LABEL-INFO
        DISPLAY "511030 THIS LABEL CREATED:-" SAMPLE-LABEL
                UPON T
        MOVE 1 TO LABEL-COUNTER
        GO TO MORE-LABELS
      ELSE
        MOVE "UHL2" TO LABEL-ID
        MOVE "SECOND LABEL PRODUCED BY ALPHA SECTION" TO LABEL-INFO
        DISPLAY "511530 AND THIS LABEL TOO:-"
                SAMPLE-LABEL UPON T
        MOVE 2 TO LABEL-COUNTER
```

```
      END-IF
    END-IF.
ALPHA-END.
    EXIT.
BETA SECTION.
    USE AFTER STANDARD ENDING FILE
       LABEL PROCEDURE ON SAMPLE-FILE.
BETA-1.
    IF INPUT-MODE
    THEN
      DISPLAY "522010 THIS LABEL READ:-" SAMPLE-LABEL
               UPON T
      GO TO MORE-LABELS
    ELSE
      IF LABEL-COUNTER = 0
      THEN
        MOVE "UTL1" TO LABEL-ID
        MOVE "THIS WAS PRODUCED BY BETA SECTION" TO LABEL-INFO
        DISPLAY "521030 THIS LABEL CREATED:-" SAMPLE-LABEL
                 UPON T
        MOVE 1 TO LABEL-COUNTER
        GO TO MORE-LABELS
      ELSE
        MOVE "UTL2" TO LABEL-ID
        MOVE "SECOND LABEL PRODUCED BY BETA SECTION" TO LABEL-INFO
        DISPLAY "521530 AND THIS LABEL TOO: "
                 SAMPLE-LABEL UPON T
        MOVE 2 TO LABEL-COUNTER
      END-IF
    END-IF.
BETA-END.
    EXIT.
END DECLARATIVES.
GAMMA SECTION.
HERE-GOES.
    MOVE "O" TO I-O-INDICATOR.
    OPEN OUTPUT SAMPLE-FILE
    CLOSE SAMPLE-FILE
    STOP RUN.
```

Example 4-2

for format 1

In this example, UHL-FIELD, LABEL-NO and USER-INFO are common label items. Therefore, the unqualified reference to UHL-FIELD in the paragraph named L-1 is legitimate.

Data Division entries:

```
FD  FILE-1.
    ...
    LABEL RECORD IS LABEL-1
    ...
01  LABEL-1.
    02 UHL-FIELD      PIC X(3).
    02 LABEL-NO       PIC 9.
    02 USER-INFO      PIC X(76).
01  RECORD-1.
    ...
FD  FILE-2
    ...
    LABEL RECORD IS LABEL-2
    ...
01  LABEL-2.
    02 UHL-FIELD      PIC X(3).
    02 LABEL-NO       PIC 9.
    02 USER-INFO      PIC X(76).
01  RECORD-2.
```

Procedure Division statements:

```
L SECTION.
    USE AFTER STANDARD LABEL PROCEDURE ON INPUT.
L-1.
    IF UHL-FIELD NOT = "UHL" THEN STOP RUN.
    ...
    GO TO MORE-LABELS.
L-9.
    EXIT.
M SECTION.
    ...
HOUSEKEEPING SECTION.
FILE-OPEN.
    OPEN INPUT FILE-1, FILE-2.
```

Format 2

```
USE [GLOBAL] AFTER STANDARD {ERROR     } PROCEDURE ON {{file-name-1}...  }.
                            {EXCEPTION }               {INPUT          }
                                                       {OUTPUT         }
                                                       {I-O            }
                                                       {EXTEND         }
```

Syntax rules for format 2

1. The ERROR and EXCEPTION phrases are equivalent and may be used interchangeably.
2. file-name-1 must not appear in more than one USE statement. The INPUT, OUTPUT, I-O, and EXTEND phrases may each be specified only once.
3. Files referenced either implicitly (INPUT, OUTPUT, I-O and EXTEND) or explicitly (file-name-1, file-name-2, ...) in the USE statement need not have the same organization and access mode.
4. The USE statement with the GLOBAL phrase can only be specified in nested programs. It is described in detail in chapter 7, "Inter-program communication" (page 568).

General rules for format 2

1. The USE procedures are performed:
   a) when an at end condition occurs, provided that the input/output statement in which the at end condition appears does not contain an AT END phrase.
   b) when a severe error occurs (FILE STATUS CODE ≥ 30).

   If there is no corresponding USE procedure for the file when a) or b) occurs, the program will abort.
2. When file-name-1 is specified, the error handling procedures are executed only for the named files. No other USE procedures are performed for these files.
3. Before execution of the user error routine, the standard system error routines for input/output error handling are executed.
4. After a USE procedure is executed, control passes to the calling routine.
5. INPUT indicates that the specified procedures are executed only for files opened in input mode (OPEN statement with the INPUT phrase.)
6. OUTPUT indicates that the specified procedures are executed only for files opened in output mode (OPEN statement with the OUTPUT phrase).

7. I-O indicates that the specified procedures are executed only for files opened in I-O mode (OPEN statement with the I-O phrase).
8. EXTEND indicates that the specified procedures are executed only for files opened in extend mode (OPEN statement with the EXTEND phrase).
9. Within a USE procedure, there must be no reference to nondeclarative procedures, except for the PERFORM statement.
10. Reference to procedure names which are subordinate to a USE statement may be made from another procedure or from a nondeclarative procedure by using a PERFORM statement only.

Example 4-3

for format 2

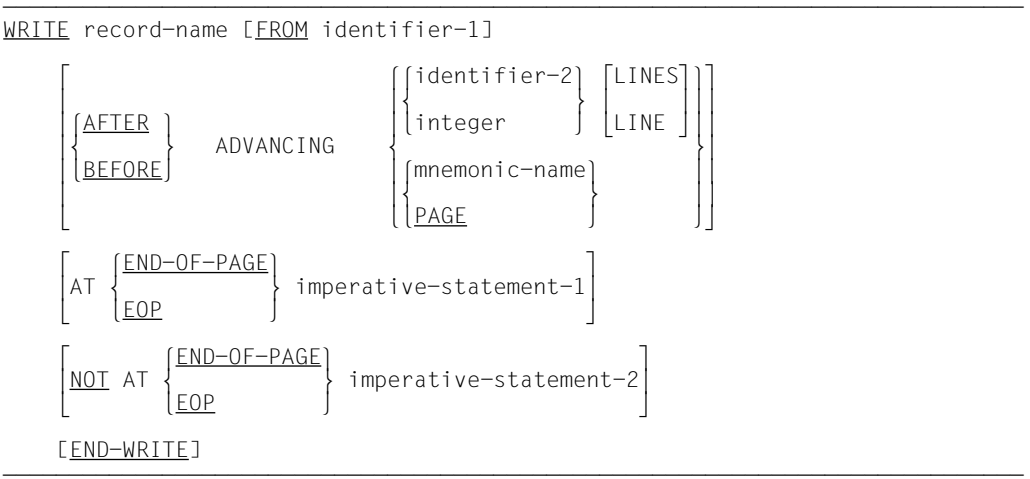
```
IDENTIFICATION DIVISION.
PROGRAM-ID. USESEQ2.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE ASSIGN "MASTER-FILE"
      FILE STATUS INDICATOR.
FILE SECTION.
FD  MASTER-FILE.
01  REC                     PIC X(80).
WORKING-STORAGE SECTION.
01  INDICATOR               PIC 99.
01  CLOSE-INDICATOR         PIC X VALUE "0".
    88  FILE-CLOSED         VALUE "0".
    88  FILE-OPEN           VALUE "1".
PROCEDURE DIVISION.
DECLARATIVES.
INPUT-ERROR SECTION.
    USE AFTER ERROR PROCEDURE ON MASTER-FILE.
STATUS-QUERY.
    IF INDICATOR = 10
       DISPLAY "End of MASTER-FILE encountered" UPON T
    ELSE
       DISPLAY "Unrecoverable error (" Indicator
               ") for MASTER-FILE" UPON T
       IF FILE-OPEN
          CLOSE MASTER-FILE
       END-IF
       DISPLAY "Program terminated abnormally" UPON T
       STOP RUN
    END-IF.
END DECLARATIVES.
BEGIN SECTION.
WORK.
    OPEN INPUT MASTER-FILE.
    SET FILE-OPEN TO TRUE.
    READ MASTER-FILE.
    CLOSE MASTER-FILE WITH LOCK.
    SET FILE-CLOSED TO TRUE.
    OPEN INPUT MASTER-FILE.
    CLOSE MASTER-FILE.
    STOP RUN.
```

WRITE statement

Function

The WRITE statement causes a record to be released to an output file. It can also be used to control form feeds in print files.

Format



Syntax rules

1. record-name and identifier-1 must not refer to the same storage area.
2. record-name must be associated with a file description (FD) entry in the Data Division of the program and may be qualified.
3. record-name must not be part of a sort file.
4. If identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

   However, the compiler also permits the use of nonnumeric elementary items for identifier-2.
5. The value of integer or the data item referenced by identifier-2 must be greater than or equal to zero, but not greater than 15.

   For files with the ASSIGN entry PRINTER literal-1, integer must be greater than 15 and less than 100.

6. If END-OF-PAGE or NOT END-OF-PAGE is used, a LINAGE clause must be present as part of the file description entry for the corresponding file (see "LINAGE clause", page 386).
7. The EOP and END-OF-PAGE phrases are equivalent.
8. mnemonic-name, if specified, must have an entry associated with it in the SPECIAL-NAMES paragraph.
9. mnemonic-name must not be supplied in the ADVANCING phrase if a LINAGE clause was specified for the file referenced in the WRITE statement.
10. ADVANCING PAGE and END-OF-PAGE may not be specified together in a single WRITE statement.

General rules

1. The record supplied by a WRITE statement is no longer available in the record area, unless the file associated with the record was specified in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful. The record is also available to any other files which may have been referenced in any SAME RECORD AREA clause together with the specified file.
2. Execution of a WRITE statement with the FROM phrase is equivalent to the execution of the following statements:

   ```
   MOVE identifier-1 TO record-name
   WRITE record-name
   ```

   Data is transferred according to the rules for a MOVE statement without the CORRESPONDING phrase.

   The content of the record area before execution of the implicit MOVE statement has no effect on the execution of the WRITE statement.

   After the WRITE statement is successfully executed, the information is still available in the area referenced by identifier-1; however, as pointed out in general rule 1, this is not necessarily true for the record area.
3. If the end-of-volume condition is encountered on a multi-volume output file for tape or disk storage and sequential access mode, the following steps will be performed during execution of the WRITE statement:

   a) The standard volume header label procedures and the user volume trailer label procedures specified via a USE procedure are performed. The order in which these procedures are executed depends on the BEFORE/AFTER phrases supplied in the USE procedure, if any.

   b) A volume swap is effected.

c) The standard volume header label procedures and the user volume header label procedures specified via a USE procedure are performed. The order in which these procedures are executed depends on the BEFORE/AFTER phrases supplied in the USE procedure, if any.

4. After execution of a WRITE statement, the value of the data item of any FILE STATUS clause existing for that file is updated (see also "FILE STATUS clause", page 365).
5. Execution of a WRITE statement does not affect the file position.
6. If a file is open in extend mode, the WRITE statement causes records to be added to the end of the file as if the file were open in output mode. The first record written after execution of the OPEN EXTEND statement is the successor of the last record in the file.
7. If the VARYING phrase has been specified for the file associated with record-name, the number of character positions in the record referenced by record-name must not be larger than integer-3 or smaller than integer-2 (see "RECORD clause", format 2, page 390). If no RECORD clause is specified with the VARYING phrase, the number of character positions must not be longer than the largest record (as determined by the record description entry) of the associated file. If either of these cases occur, the WRITE statement is unsuccessful, and the WRITE operation does not take place. The content of the record area remains unchanged. The I-O status of the file associated with record-name is set to a value indicating the cause of the condition (see "I-O status", page 356).
8. Both the ADVANCING and the END-OF-PAGE phrases allow control of the vertical positioning of each individual line on a representation of a printed page.
9. When the BEFORE/AFTER ADVANCING phrase is used, the corresponding record is printed before or after the page is advanced according to the rules described below.
10. If the end-of-page condition does not occur during the execution of a WRITE statement with the NOT END-OF-PAGE phrase, control is transferred to imperative-statement-2 if the WRITE statement is executed successfully. The transfer of control takes place after the record is written and after updating of the I/O status of the file from which the record originates.

    If the execution of the WRITE statement is unsuccessful, the I/O status of the file is updated, then the USE procedure specified for this file is executed, and finally control is transferred to the imperative-statement-2.
11. The ADVANCING phrase causes the representation of the printed page to be advanced according to the following rules:

    a) If identifier-2 (nonnumeric or numeric integer elementary data item), integer or mnemonic-name is present, the page is advanced in accordance with the rules specified in Table 4-7.

b) If PAGE is present, the record is printed either before or after positioning to the beginning of the first line on a new page. The beginning of a new page is defined either by the physical properties of the printer (advance to channel 1) or, if a LINAGE clause is present, by the beginning of the next logical page (see also "LINAGE clause", page 386).

12. If the ADVANCING phrase is omitted, a WRITE...AFTER ADVANCING 1 LINE will be executed for files with the ASSIGN specification PRINTER or PRINTER literal-1 (see also Table 4-7, page 426).

13. If the end of a logical page is encountered during execution of a WRITE statement with the END-OF-PAGE phrase, control is transferred to imperative-statement-1. The logical end of a page is defined by the LINAGE clause in the file description entry of the file.

14. An END-OF-PAGE condition occurs when the execution of a WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the page footing area. This is the case whenever, during the execution of such a WRITE statement, the value of the special register LINAGE-COUNTER becomes equal to or greater than the value of integer-2 of the LINAGE clause data item referenced by data-name-2. In this case, the WRITE statement is executed, and control is then transferred to the imperative statement which follows the END-OF-PAGE phrase.

15. An automatic page overflow condition occurs when the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

    This is the case whenever, during the execution of such a WRITE statement, the value of the special register LINAGE-COUNTER becomes equal to or greater than the value of integer-1 or the LINAGE clause data item referenced by data-name-1. In this case, the record is printed either before or after positioning to the first line of the next logical page (depending on the BEFORE/AFTER phrase). This line is defined by specification of the LINAGE clause. The imperative statement specified in the END-OF-PAGE phrase is executed after the record has been written and the device repositioned.

    If the integer-2 or data-name-2 phrases of the LINAGE clause are not used, the END-OF-PAGE condition will occur whenever a WRITE statement is executed (with EOP phrase specified) and the value in the special LINAGE-COUNTER register is greater than or equal to the value of integer-1 or the LINAGE clause data item referenced by data-name-1.

    If integer-2 or data-name-2 is specified in the LINAGE clause but execution of a WRITE statement would result in the value of the LINAGE-COUNTER being both

    a) equal to or greater than the value of data-name-2 or integer-2, and

    b) equal to or greater than the value of data-name-1 or integer-1

    processing proceeds as if integer-2 or data-name-2 had not been specified.

16. When the ADVANCING phrase is used together with the LINAGE clause for a device which does not represent a physical printer, any blank lines at the top and bottom margins as well as within the page body are represented on the appropriate storage medium as records containing blank characters.
17. Execution of a WRITE statement which attempts to write outside the physical boundaries of a file results in an exception condition. The following steps are performed:
    a) The value of the FILE STATUS data item of the file is set to indicate violation of the boundaries of the file (see "FILE STATUS clause", page 365).
    b) Any USE ERROR/EXCEPTION procedure that is explicitly or implicitly specified for this file will be executed.
    c) If there is neither an explicit nor an implicit USE procedure for this file, the program will terminate abnormally.
18. When a WRITE statement is executed, the file must have been opened with an OPEN statement containing the OUTPUT or EXTEND phrase.
19. Table 4-7 gives precise detail of the use of the first character in the record, depending on the symbolic device names of the ASSIGN clause and the WRITE statement phrases.
20. When a WRITE statement is used with AFTER-ADVANCING phrase, the printer is positioned to the line requested and is advanced one additional line after printing.

    If PRINTER literal-1 is specified in the ASSIGN phrase, a WRITE AFTER phrase is performed as follows:

    a) An empty record is printed with corresponding form-feed specification (WRITE BEFORE).

    The record is written.

    For all devices, form-feed suppression is only possible when the WRITE BEFORE phrase is specified.
21. END-WRITE delimits the scope of the WRITE statement.

Example 4-4

The file LOADFILE is read and written to the print file called AFILE.

Each current record is written before advancing 5 blank lines.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WP1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT AFILE ASSIGN TO PRINTER "PRINTOUT".
    SELECT LOADFILE ASSIGN TO "MASTER-FILE".
DATA DIVISION.
FILE SECTION.
FD   AFILE

     LINAGE IS 24 LINES
     LINES AT TOP 24
     LINES AT BOTTOM 24.
01  CUSTMRREC  PIC X(95).
FD   LOADFILE
     LABEL RECORD IS STANDARD.
01  CUSTREC.
    05 CNA PIC X(30).
    05 STR PIC X(20).
    05 LOC PIC X(20).
    05 ZIP PIC X(5).
    05 TYP PIC X(20).
WORKING-STORAGE SECTION.
01  FILE-SWITCH PIC X VALUE LOW-VALUE.
   88  EOF VALUE HIGH-VALUE.
PROCEDURE DIVISION.
MAIN SECTION.
FILE-OPEN.
    OPEN OUTPUT AFILE.
    OPEN INPUT LOADFILE.
READS.
    PERFORM UNTIL EOF
      READ LOADFILE INTO CUSTMRREC
      AT END
        CLOSE AFILE LOADFILE
        SET EOF TO TRUE
      NOT AT END
        WRITE CUSTMRREC BEFORE ADVANCING 5.
      END-READ
    END-PERFORM
    STOP RUN.
```

Indicator	Format	Action	
identifier-2	Numeric integer with a value in the range from 01 to 15	n-line spacing	
identifier-2	Alphanumeric data item, length 1 (PIC X). Following values are valid: blank 0 - + 1-8 A B	 Single-spacing Double-spacing Triple-spacing Suppress spacing Skip to channel 1-8 Skip to channel 10 Skip to channel 11	
integer	Numeric literal in the range 01 to 15	n-line spacing	
mnemonic-name	A name related to an implementor-name in a SPECIAL-NAMES paragraph.	implementor-name	Action
		C01 thru C08 C10 OR C11	Skip to channel 1-8 Skip to channel 10-11

Table 4-6 Meaning and result of spacing indicators in a WRITE ADVANCING statement

Symbolic device name	WRITE statement without ADVANCING phrase	WRITE statement with ADVANCING phrase	Comments
PRINTER literal	Standard spacing when ADVANCING is omitted as if AFTER 1 LINE had been specified; the first character of the record is available for user data.	The first character of the record is available for user data.	The place for the carriage control character is reserved by the compiler and is not accessible to the user. This type of printer supports specification of the LINAGE clause in the file description entry. WRITEstatements both with and without the ADVANCING phrase specified are allowed for a given file.

Table 4-8: Use of symbolic device names for printers in connection with the WRITE statement

Symbolic device name	WRITE statement without ADVANCING phrase	WRITE statement with ADVANCING phrase	Comments
PRINTER PRINTER01- PRINTER99	As above.	As above.	The place for the carriage control character is reserved by the compiler and is not accessible to the user. The LINAGE clause is not permitted for this file. Use of WRITE statements with and without the ADVANCING phrase for the same file is not permitted. If this does occur, a WRITE AFTER ADVANCING is executed implicitly for the records without the ADVANCING phrase.
literal	Spacing is controlled by the first character each logical record; the user must therefore supply the appropriate control character before every execution of such WRITE statement.	The user must reserve the first character of a logical record; the runtime system inserts the feed control characof the record. program execution. Any user data there will be overwritten.	Mixed use of WRITE statements both with and without specifications of the ADVANCING phrase is permitted. In either case, however, the user information of the printer record begins only with the second character of the record.

Table 4-8: Use of symbolic device names for printers in connection with the WRITE statement

5 Relative file organization

5.1 File concepts

A file is a collection of records that can be transferred to, or read from, a volume. The user defines the organization of the file as well as the mode and order in which the records are processed.

The organization of a file describes its logical structure. There are sequential, indexed, and relative types of file organization. The file organization which is defined at the time a file is created cannot be changed later on. This same rule applies to the record size of a particular file, as defined in the file and record description entries.

5.1.1 Relative organization

When using relative file organization, the location of each record in a relative file is determined by means of a relative record number, i.e. an integer value greater than zero, which specifies the position of that record within the logical sequence of the file. The record number is predefined by the user in a relative key field. The file may be seen as a serial sequence of areas, each of which contains one logical record. Each of these areas is identified by a relative record number. Storage and retrieval of the records is accomplished on the basis of that number. For example, the tenth record is referenced through the relative record number 10 and is located in the tenth record area, whether or not records have been written in any of the record areas 1 to 9. Relative file organization is permitted for disk storage files only.

5.1.2 Sequential access to records

Records in a relative file may be sequentially created, read, and updated.

A relative file may be created sequentially, in which case the RELATIVE KEY need not be specified, as the records are written to the output file in physically consecutive order.

In sequential reading of records from a relative file, the records are processed in the order of their relative record numbers, i.e. a READ statement will make available the next existing logical record of the file.

The first record that is read is either the first record made available in the file, or a record whose position was specified in a START statement. In the latter case, RELATIVE KEY must be specified.

If the RELATIVE KEY is specified for a relative file, execution of a READ or WRITE statement causes the RELATIVE KEY item to be set to the relative record number of the record which is made available.

An already existing relative file may be updated with the aid of READ, REWRITE, or DELETE statements. The last record read may be updated and then rewritten to its original location in the file, or it may be logically deleted.

5.1.3 Random access to records

Records of a relative file may be randomly created, read, and updated.

With this access method, the RELATIVE KEY phrase is always required. Before each execution of an input statement or output statement, the data item indicated in RELATIVE KEY phrase must be provided with the value of the required relative record number.

A relative file may be created randomly; in this case, the record placed on the file occupies the record area of the relative record number as specified in the RELATIVE KEY data item.

When a relative file is read randomly, the retrieved record is the one whose relative record number is contained in the data item of the RELATIVE KEY phrase associated with that file.

An already existing relative file may be updated with the aid of REWRITE or DELETE statements. An explicit random READ statement may optionally be issued prior to the execution of a REWRITE or DELETE statement for the record to be updated, as these statements will modify the record that is denoted by the relative record number in the RELATIVE KEY data item.

5.1.4 Dynamic access to records

Combination of sequential and random access modes.

5.1.5 I-O status

The I-O status is a value that can be used in a COBOL program to check the status of an input/output operation. In order to do this, the FILE STATUS clause must be specified in the FILE CONTROL paragraph of the Environment Division.
The I-O status value is transferred to a two-character data item

– during the execution of a CLOSE, OPEN, READ, REWRITE, or WRITE statement,

– prior to the execution of any associated imperative statement, and

– prior to the execution of any corresponding USE AFTER STANDARD EXCEPTION procedure.

The table below shows the I-O status values and their meanings:

I-O status	Meaning
	Execution successful
00	The I-O statement terminated normally. No further information regarding the I-O operation is available.
04	Record length conflict: A READ statement was executed success- fully, but the length of the record which was read does not lie within the limits specified in the record description for the file. Successful OPEN INPUT/I-O/EXTEND for a file with the OPTIONAL phrase in the SELECT clause that was not present at hte time of execution of the OPEN statement.
	Execution unsuccessful: at end condition
10	An attempt was made to execute a READ statement. However, no next logical record was available, since the end-of-file was encountered (sequential READ). A first attempt was made to execute a READ statement for a non-existent file which is specified as OPTIONAL.
14	An attempt was made to execute a READ statement. However, the data item described by RELATIVE KEY is too small to accommodate the relative record number. (sequential READ).
	Execution unsuccessful: invalid key condition
22	Duplicate key An attempt was made to execute a WRITE statement with a key for which there is already a record in the file.
23	Record not located or zero record key An attempt was made (using a READ, START, DELETE or REWRITE statement with a key) to access a record not contained in the file, or the access was effected with a zero record key.

I-O status	Meaning
24	Boundary values exceeded (see "COBOL85 User Guide" [1]). An attempt was made to execute a WRITE statement beyond the system-defined boundaries of a relative file (insufficient secondary allocation in the FILE command), or a WRITE statement is attempted in sequential access mode with a relative record number so large that it does not fit in the data item defined with the RELATIVE KEY phrase.
	Execution unsuccessful: permanent error
30	No further information regarding the I-O operation is available.
35	An attempt was made to execute an OPEN INPUT/I-O statement for a nonexistent file.
37	An OPEN statement is attempted on a file that cannot be opened due to the following conditions: 1. OPEN OUTPUT/I-O/EXTEND on a write-protected file (password, RETPD in catalog, ACCESS=READ in catalog) 2. OPEN INPUT on a read-protected file (password)
38	An attempt was made to execute an OPEN statement for a file previously closed with the LOCK phrase.
39	The OPEN statement was unsuccessful as a result of one of the following conditions: 1. One or more of the operands ACCESS-METHOD, RECORD-FORMAT or RECORD-SIZE were specified in the SET-FILE-LINK command with values which conflict with the corresponding explicit or implicit program specifications. 2. The catalog entry of the FCBTYPE operand for an input file does not match the corresponding explicit or implicit program specification or the specification in the SET-FILE-LINK command. 3. Variable record length has been defined for a file that is to be processed using the UPAM access method of the DMS.
	Execution unsuccessful: logical error
41	An attempt was made to execute an OPEN statement for a file which was already open.
42	An attempt was made to execute a CLOSE statement for a file which was not open.
43	For ACCESS MODE IS SEQUENTIAL: The most recent I-O statement executed prior to a DELETE or REWRITE statement was not a successfully executed READ statement.
44	Record length limits exceeded: An attempt was made to execute a WRITE or REWRITE statement, but the length of the record does not lie within the limits defined for the file.

I-O status	Meaning
46	An attempt was made to execute a sequential READ statement for a file in INPUT or I-O mode. However, there is no valid next record since: 1. the preceding START statement terminated abnormally, or 2. the preceding READ statement terminated abnormally without causing an at end condition. 3. the preceding READ statement caused an AT END condition.
47	An attempt was made to execute a READ or START statement for a file not in INPUT or I-O mode.
48	An attempt was made to execute a WRITE statement for a file – not in OUTPUT or EXTEND mode (for sequential access) – not in OUTPUT or I-O mode (for random or dynamic access)
49	An attempt was made to execute a DELETE or REWRITE statement for a file not in I-O mode.
	Other conditions with unsuccessful execution
90	System error; no further information regarding the cause is available.
91	System error; OPEN error
93	For simultaneous processing only (see "Shared updating of files" in the "COBOL85 User Guide" [1]): The I-O statement could not terminate normally because a different task is accessing the same file, and the access operations are incompatible.
94	For shared update processing only (see "Shared updating of files" in the "COBOL85 User Guide" [1]): deviation from call sequence READ - REWRITE/DELETE.
95	The specifications in the BLOCK-CONTROL-INFO or BUFFER-LENGTH operands of the SET-FILE-LINK command are not consistent with the file format, the block length, or the format of the volume being used.

5.2 Language elements of the Environment Division

INPUT-OUTPUT SECTION

Function

The INPUT-OUTPUT SECTION deals with the following:

– the definition of each file used in the program,

– the assignment of these files to external devices, and

– the transmission and handling of data between external devices and the object program.

This section is divided into two paragraphs:

– the FILE-CONTROL paragraph, which names the files used in the program and assigns them to external devices, and

– the I-O-CONTROL paragraph, which defines special input/output techniques.

Format

```
 A       Margin indication
 |
 |
 |
 v

 INPUT-OUTPUT SECTION.

 FILE-CONTROL. {file-control-entry}...
[I-O-CONTROL.  [input-output-control-entry]]
```

General rules

1. All sections and paragraphs must begin in area A.
2. The INPUT-OUTPUT SECTION is optional.

   If the INPUT-OUTPUT SECTION is defined, the FILE-CONTROL paragraph must also be specified.

FILE-CONTROL paragraph

Function

In the FILE-CONTROL paragraph each file is given a name and is then assigned to one or more external devices. The relevant file attributes required for processing are also provided. This information specifies how the data is organized and how it is to be accessed.

Format

```
FILE-CONTROL.

    SELECT clause
    ASSIGN clause
   [ACCESS MODE clause]
   [FILE STATUS clause]
    ORGANIZATION clause
   [RESERVE clause].
```

Syntax rules

1. The FILE-CONTROL heading must be written in area A; all subsequent entries must be written in area B.
2. The SELECT clause must be the first entry in the FILE-CONTROL paragraph. All other clauses may appear in any order.

In the pages that follow, the SELECT and ASSIGN clauses are described first, followed by the remaining clauses in alphabetical order.

SELECT clause

Function

The SELECT clause is used to give a name to each file in the program.

Format 1 applies to all files except sort files.

Format 2 is suitable for sort files

(see chapter 10, "Sorting of records", page 652).

Format 1

```
SELECT [OPTIONAL] file-name
```

Syntax rules

1. Each file-name used in a program may appear only once in the SELECT clause. File-name refers to the name by which a file is referenced in the source program (internal file-name).
2. Each file specified in a SELECT clause must have a file description (FD) entry in the Data Division of the source program.
3. The OPTIONAL phrase is required for files which are not necessarily present each time the program is executed. If such a file is not present at run time, the first READ statement for this file branches to the related at end condition.
   The OPTIONAL phrase is permitted only for files opened in INPUT, I-O or EXTEND mode.

General rule

If the OPTIONAL phrase refers to an *external* file, OPTIONAL must be specified in all programs that describe this external file.

ASSIGN clause

Function

The ASSIGN clause assigns an external device to a file of the COBOL program. One ASSIGN clause is required for each file in the program.

Format

```
ASSIGN TO {literal-1   }...
          {data-name-1}
```

Syntax rules

1. literal-1 or the contents of data-name-1 specifies the link name for the file. The name must be alphanumeric, must be specified in uppercase letters, and must not be a figurative constant. The link name is formed from the first 8 characters of this literal, and must therefore be unique within the program. If the 8th character is a hyphen (-), it is replaced by a # character.
2. data-name-1 must not be qualified.
3. data-name-1 must be defined as an alphanumeric data item or as a group item in the WORKING-STORAGE SECTION or LINKAGE SECTION.

General rules

1. The type of file organization should be specified in the ORGANIZATION clause (see page 441).
2. If a number of literals are specified, only the first will be evaluated; the remaining literals are ignored.
3. If the ASSIGN clause refers to an *external* file, the ASSIGN clause must be used in the same form in all programs that describe this external file. The contents of the literal or data-name may, however, be different.

ACCESS MODE clause

Function

The ACCESS MODE clause determines the manner in which the records of a file are to be accessed.

Format

```
                  ⎧SEQUENTIAL [RELATIVE KEY IS data-name-1]⎫
ACCESS MODE IS    ⎪⎧RANDOM    ⎫                           ⎪
                  ⎨⎨           ⎬ RELATIVE KEY IS data-name-1⎬ .
                  ⎪⎩DYNAMIC   ⎭                           ⎪
                  ⎩                                       ⎭
```

Syntax rules

1. The RELATIVE KEY phrase specifies the key field whose content is used for retrieving or placing a logical record in a file.
2. data-name-1 must not be subscripted or indexed.
3. The data item referenced by data-name-1 must be defined as an unsigned integer.
4. data-name-1 must not be specified within the record description entry of the associated file.

General rules

1. SEQUENTIAL means that records are read or written sequentially, i.e. the next logical record of the file is made available when a READ statement is executed, or the next logical record is placed on that file when a WRITE statement is executed.
2. RANDOM means that records are randomly read or written on the basis of a key, i.e. access is made using the RELATIVE KEY.
3. DYNAMIC means that random and/or sequential access is possible.
4. If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is assumed.
5. The data item referenced by data-name-1 is used for communicating a relative record number between the user and the input/output system.

   All records which exist on a file are uniquely identified by their relative record numbers. The relative record number of a given record specifies the logical position of that record within the file record sequence. The first logical record is relative record number one (1), and the following logical records have the relative record numbers 2, 3, 4... .

6. The RELATIVE KEY must not be zero.
7. If the ACCESS MODE clause refers to an *external* file, an equivalent ACCESS MODE clause must be specified in all other programs that describe this external file; in particular, data-name-1 must be an external data item identical in all programs.

FILE STATUS clause

Function

The FILE STATUS clause specifies a data item that indicates the status of input/output operations during processing. In addition, by specifying a further item, an additional error code is made available.

Format

```
FILE STATUS IS data-name-1 [, data-name-2]
```

Syntax rules

1. data-name-1 and data-name-2 must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION of the Data Division.
2. data-name-1 must be a two-byte numeric (USAGE DISPLAY only) or alphanumeric item.
3. data-name-2 must be a 6-character group item with the following format:

```
01  data-name-2.
    02  data-name-2-1  PIC 9(2) COMP.
    02  data-name-2-2  PIC X(4).
```

General rules

1. If the FILE STATUS clause is specified, the runtime system copies the I-O status to data-name-1.
2. If specified, data-name-2 is assigned as follows:t
   a) If data-name-1 has the value 0, the contents of data-name-2 are undefined.
   b) If data-name-1 has a non-zero value, data-name-2 contains the additional error code. The value 64 in data-name-2 indicates that this code is the (BS2000) DMS code; the value 96 in data-name-2 indicates that the code is the (POSIX) SIS code. The command HELP DMS <contents-of-data-name-2-2> or HELP SIS <contents-of-data-name-2-2> supplies more detailed information on the corresponding error code.
3. The I-O status is copied during the execution of each OPEN, CLOSE, READ, WRITE, or REWRITE statement that references the specified file, and prior to the execution of each corresponding USE procedure (see "I-O status", page 431).

ORGANIZATION clause

Function

The ORGANIZATION clause defines the logical structure of a file.

Format

```
[ORGANIZATION IS] RELATIVE
```

General rules

1. File organization is defined at the time a file is created and cannot be changed subsequently.
2. If ORGANIZATION IS RELATIVE is specified for an *external* file, ORGANIZATION IS RELATIVE must be specified in all programs that describe this external file.

RESERVE clause

Function

The RESERVE clause allows the user to modify the number of input/output areas (buffers) to be allocated to the object program by the compiler.

Format

```
RESERVE integer  [AREA ]
                 [AREAS]
```

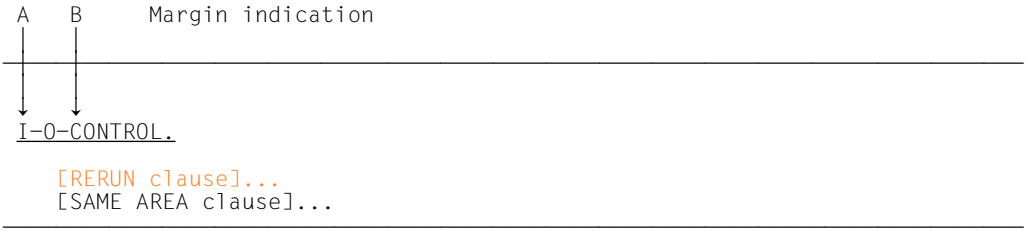
The RESERVE clause is treated as a comment by the compiler.

I-O-CONTROL paragraph

Function

The I-O-CONTROL paragraph defines the events at whose occurrence restart points are to be established and specifies the memory area which is to be shared by different files. In addition, it indicates the location of files on multiple-file reels and defines special input/output conditions.

Format



Syntax rule

I-O-CONTROL must be written starting in area A. All subsequent entries must be written in area B.

RERUN clause

Function

The RERUN clause indicates where and when restart-point records are to be issued. A restart-point record describes the status of an object program at a specified point during program execution. It is produced automatically by the operating system upon the request of the object program and contains all information necessary to restart the program from that point. The RERUN clause controls such requests by the COBOL object program.
For further information on the RERUN clause, see "COBOL85 User Guide" [1].

Format

```
RERUN [ ON { implementor-name } ]
       [   { file-name-1     } ]

      EVERY { integer-1 RECORDS OF file-name-2 }
            { integer-2 CLOCK-UNITS            }
            { condition-name                   }
```

Syntax rules and general rules

See RERUN clause for sequential files, page 372.

SAME AREA clause

Function

The SAME AREA clause indicates which files are to share a specified input/output area during program execution.

Format 1 applies to all files except sort files, unless RECORD is specified.

Format 2 is suitable for sort files

(see chapter 10, "Sorting of records", page 650.

Format 1

```
SAME [RECORD] AREA FOR file-name-1 {file-name-2}...
```

Syntax rules and general rules

See SAME AREA clause for sequential files, page 374.

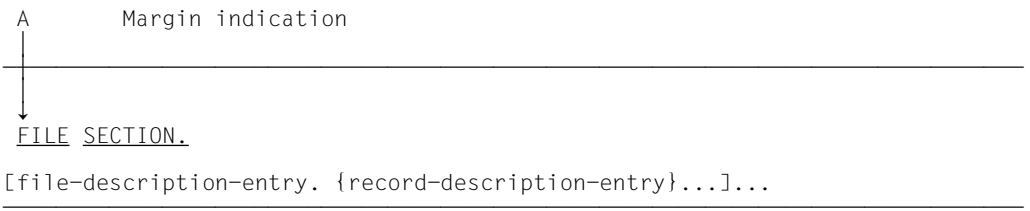
5.3 Language elements of the data division

FILE SECTION

Function

The FILE SECTION is used to define the structure of files. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

Format



File description entries are discussed below. Record description entries are discussed in chapter 3 (page 141ff) and on the following pages.

File description (FD) entry

Function

The file description (FD) entry specifies the physical setup and the record names of a given file.

A file description entry must be written for each file to be processed by the program. The information contained in this entry generally pertains to the physical aspects of the file, i.e. the description of data as it appears on the input or output medium.

Format

```
A   B     Margin indication
|   |
|   |
|   |
FD  file-name

    [BLOCK CONTAINS clause]
    [DATA RECORDS clause]
    [EXTERNAL clause]
    [GLOBAL clause]
    [LABEL RECORDS clause]
    [RECORD clause]
    [VALUE OF clause].
```

Syntax rules

1. The level indicator FD identifies the beginning of a file description entry.
2. file-name must be identical to the file-name given in a SELECT clause.
3. Clauses that follow the file-name may appear in any order.
4. The file description entry must be followed by one or more record description entries.

General rules

1. Table 5-1 provides a summary of the functions of clauses used in file description entries.

Clause	Function
BLOCK CONTAINS	Specifies physical block length
DATA RECORDS	Specifies the names of the records in the file
EXTERNAL	Declares a file as external
GLOBAL	Declares a file as global
LABEL RECORDS	Gives the names and values of the label records contained in the file.
RECORD	Specifies logical record size
VALUE OF	Specifies the values of some data items of a label record.

Table 5-1: Functions of file description clauses

2. The EXTERNAL and GLOBAL clauses are described in chapter 7 "Inter-program communication" (page 544 and page 547). The other file description clauses are described below in alphabetical order.

BLOCKS CONTAINS clause

Function

The BLOCK CONTAINS clause specifies the maximum size of a physical block.

Format

```
BLOCK CONTAINS [integer-1 TO] integer-2 {CHARACTERS}
                                        {RECORDS   }
```

Syntax rules and general rules

See BLOCK CONTAINS clause for sequential files, page 379.

DATA RECORDS clause

Function

The DATA RECORDS clause is used only for documentation. It specifies the names of the records in a file.

Format

```
DATA {RECORD  IS } {data-name-1}...
     {RECORDS ARE}
```

Syntax rules and general rules

See DATA RECORDS clause for sequential files, page 383.

LABEL RECORDS clause

Function

The LABEL RECORDS clause specifies whether labels are present and, if so, indicates whether they are conforming labels.

Format

Syntax rules and general rules

See LABEL RECORDS clause for sequential files, page 390.

RECORD clause

Function

The RECORD clause specifies the size of the records in a file.

Format 1 indicates fixed-length records by specifying the number of character positions in a record.

Format 2 indicates variable-length records by specifying a range of values for the record length.

Format 3 indicates variable-length records by specifying the minimum and maximum number of character positions in a record.

The length of each record is defined precisely by its record description entry. If the RECORD clause is specified, the record length is compared with the specifications in the clause.

Note

For relative files that are to be processed using the UPAM access method of the DMS, only fixed-length records (format 1) may be defined (see "COBOL85 User Guide" [1]).

Format 1

```
RECORD CONTAINS integer-1 CHARACTERS
```

Syntax rules and general rules

See RECORD clause for sequential files, page 390.

Format 2

```
RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]

     DEPENDING ON data-name-1
```

Syntax rules and general rules

See RECORD clause for sequential files, page 390.

Format 3

```
RECORD CONTAINS integer-4 TO integer-5 CHARACTERS
```

Syntax rules and general rules

See RECORD clause for sequential files, page 392.

Note:

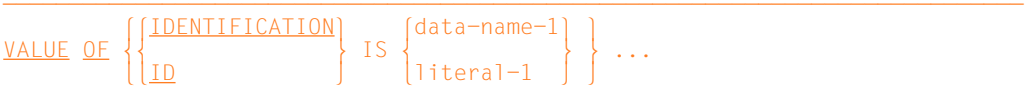
The maximum limits specified for sequential files are reduced by eight bytes for files organized relatively.

VALUE OF clause

Function

The VALUE OF clause particularizes the description of items in a label record.

Format



The VALUE OF clause is treated as a comment by the compiler.

5.4 Language elements of the Procedure Division

5.4.1 Invalid key condition

The invalid key condition may occur after the execution of a START, READ, WRITE, REWRITE, or DELETE statement. Details concerning the occurrence of this condition are described under each of these statements.

If an invalid key condition was encountered, the File Control Processor performs the following actions in the specified order.

1. If a FILE STATUS clause exists for the file, a corresponding value is entered into the FILE STATUS data item in order to indicate the invalid key condition (see "FILE STATUS clause", page 440).
2. If the INVALID KEY phrase is specified in the input/output statement, any USE procedure associated with the file connector is not executed; control is passed to the imperative-statement specified in the INVALID KEY phrase instead. Execution then continues according to the rules for that statement. Depending on the type of statement specified, control is transferred to some other branch of the program or to the end of the input/output statement, and the NOT INVALID KEY phrase, if specified, is ignored.
3. If the INVALID KEY phrase is not specified, a USE procedure must be declared. The specified procedure is then executed. The NOT INVALID KEY is ignored, if specified.

If the invalid key condition does not exist after the execution of an input/output statement, the INVALID KEY phrase is ignored, if specified. The I-O status is updated, and the following actions occur:

1. If an exception condition which is not an invalid key condition exists, control is transferred to the USE procedure.
2. If no exception condition exists, control is transferred to the end of the input/output statement or to the imperative-statement of the NOT INVALID KEY phrase, if it is specified. In the latter case, execution continues according to the rules for the specified imperative-statement, and control is transferred to some other branch in the program or to the end of the input/output statement.
3. If neither the INVALID KEY phrase has been specified, nor a USE procedure declared, the occurrence of an invalid key or exception condition causes the program to abort with the standard error handling routine.

5.4.2 Input/output statements

In COBOL, input and output is record-oriented. Thus, the READ, WRITE, DELETE and REWRITE statements process records. The COBOL user is therefore only concerned with the processing of single records. The following operations are performed automatically: moving data into input/output areas (buffers) and/or internal storage, validity checking, error correction (where feasible).

Summary

Statement	Function
CLOSE	Terminates processing of a file
DELETE	Removes a record
OPEN	Opens a file for processing
READ	Reads a record
REWRITE	Replaces a record
START	Positions within a file
USE	In addition to input/output statements, USE statements may be specified to indicate label and error handling procedures (see "Declaratives subdivision of the Procedure Division", page 213).
WRITE	Writes a record

CLOSE statement

Function

The CLOSE statement terminates the processing of files, with optional lock where applicable.

Format

```
CLOSE {file-name-1 [WITH LOCK]}...
```

Syntax rule

The files referenced in the CLOSE statement need not all have the same organization or access.

General rules

1. A CLOSE statement may only be executed for a file in an open mode.
2. After execution of a CLOSE statement, the contents of the data item specified in the FILE STATUS clause will be updated (see also "FILE STATUS clause", page 440).
3. After successful execution of a CLOSE statement, the record area assigned to the file is no longer accessible. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.
4. All files that are still open on completion of a task are closed.
5. If more than one file-name is specified, the result is the same as if a separate CLOSE statement had been written for each file-name.
6. The LOCK phrase implies that the file cannot be opened again during the current execution of this run unit.

DELETE statement

Function

A DELETE statement logically removes a record from a disk storage file.

Format

```
DELETE file-name RECORD

    [INVALID KEY imperative-statement-1]

    [NOT INVALID KEY imperative-statement-2]

    [END-DELETE]
```

Syntax rules

1. The INVALID KEY phrase must not be specified for a DELETE statement which references a file in sequential access mode.
2. If a file is not in sequential access mode and no applicable USE procedure has been specified, the INVALID KEY phrase is mandatory.

General rules

1. The file referenced in the DELETE statement must be open in I-O mode during the execution of this statement (see also "OPEN statement", page 460).
2. After successful execution of the DELETE statement, the identified record is deleted from the file.
3. Execution of a DELETE statement does not affect the contents of the record area associated with the file or the content of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause.
4. For a file in sequential access mode, the last input/output statement entered prior to the DELETE statement must be a successfully completed READ statement. The RECORD KEY must not be changed between reading and deletion. Execution of the DELETE statement causes the record read by the preceding READ statement to be logically removed from that file.

5. For a file in random or dynamic access mode, the File Control Processor deletes the record identified by the contents of the data item specified in the RECORD KEY clause of the file. If the record referenced by the key does not exist on the file, an invalid key condition occurs (see "Invalid key condition", page 455).
6. After execution of the DELETE statement, the contents of the data item specified in the FILE STATUS clause for the file are updated (see "FILE STATUS clause", page 440).
7. Transfer of control following the execution of the DELETE statement depends on whether the INVALID KEY or NOT INVALID KEY phrase is specified (see "Invalid key condition", page 455).
8. END-DELETE delimits the scope of the DELETE statement.

OPEN statement

Function

The OPEN statement opens files for processing.

Format

```
      ⎧INPUT {file-name-1}...  ⎫
      ⎪OUTPUT {file-name-2}... ⎪
OPEN  ⎨                        ⎬...
      ⎪I-O {file-name-3}...    ⎪
      ⎩EXTEND {file-name-4}... ⎭
```

Syntax rules

1. The same file-name must not appear more than once in an OPEN statement.
2. The files specified in a given OPEN statement need not all have the same organization or access mode.
3. The EXTEND phrase may be specified only for files with sequential access.

General rules

1. After successful execution of an OPEN statement, the file specified by file-name is available in the open mode.
2. After successful execution of an OPEN statement, the associated record area is available to the program. Only one record area exists for an external file; this area is available to all programs that describe this file.
3. Before successful execution of an OPEN statement for a file, no statement (except for a SORT or MERGE statement with the USING or GIVING phrase) may be executed that would either explicitly or implicitly reference that file.
4. INPUT means that the file is to be processed as an input file (input mode).
5. OUTPUT indicates that the file is to be processed as an output file (output mode).
6. I-O indicates that input and output operations (i.e. read, write and update operations) are to be performed on the file (update mode). Since this phrase implies the existence of the file, it cannot be used if the file is being initially created.
7. The EXTEND phrase positions the file immediately after the last logical record in the file. Subsequent WRITE statements for this file causes records to be appended to the file exactly as if it had been opened in OUTPUT mode (extend mode).

8. If the FILE STATUS clause is specified, the contents of the data-name given in the FILE STATUS clause are updated following execution of an OPEN statement (see also FILE STATUS clause, page 440).
9. All of the INPUT, OUTPUT or I-O phrases may be used, within the same program, in different OPEN statements for a given file. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement for the same file must be preceded by the execution of a CLOSE statement. The LOCK phrase must not be specified in this CLOSE statement.
10. If more than one file-name is specified in an OPEN statement, the result is the same as if a separate OPEN statement had been written for each file name.
11. If an optional file for which OPEN INPUT is specified does not exist, the I/O status is set to the appropriate value.
12. When a file is opened with INPUT or I-O, it is positioned to the first record to be processed.
13. The EXTEND phrase positions the file immediately after the last logical record of the file. For a relative file, the last logical record is the record with the highest existing relative record number.
14. If an optional file does not exist, successful execution of an OPEN statement with I-O or EXTEND phrase causes the file to be created.
    The same applies to an OPEN statement with OUTPUT phrase. The newly created file is empty.
15. Table 5-2 lists the statements permissible in the OPEN mode (X = "permitted").

ACCESS clause	Statement	Open mode			
		INPUT	OUTPUT	I-O	EXTEND
SEQUENTIAL	READ	X		X	
	WRITE		X		X
	REWRITE			X	
	START	X		X	
	DELETE			X	
RANDOM	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
DYNAMIC	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

Table 5-2: Permissible input/output statements in each OPEN mode

READ statement

Function

The READ statement makes the next logical record from a file (sequential access) or a specified record from a disk file (random access) available to the program.

Format 1 is used for the sequential reading of records from all files in sequential or dynamic access mode.

Format 2 is used for the random (RECORD KEY) reading of records from all files in random or dynamic access mode.

The extension WITH NO LOCK in both formats is effective during shared updating of files; it is described in the "COBOL85 User Guide" [1].

Format 1

```
READ file-name [WITH NO LOCK] [NEXT] RECORD [INTO identifier]

   [AT END imperative-statement-1]

   [NOT AT END imperative-statement-2]

   [END-READ]
```

Syntax rules for format 1

1. file-name and identifier must not reference the same storage area.
2. If no USE procedure is declared for the file, AT END must be specified in the READ statement.
3. NEXT must be specified if records of a file in dynamic access mode are to be read sequentially and neither AT END nor NOT AT END is specified.

General rules

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.
2. The NEXT phrase is optional in the sequential access mode and has no significance.

3. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current record are undefined after execution of the READ statement.
4. The INTO phrase may be specified in a READ statement:
   - if only one record description is subordinate to the file description entry, or
   - if all record-names associated with the file-name and the data item referenced by the identifier represent group items or alphanumeric elementary items.
5. The execution of a READ statement with the INTO phrase is equivalent to:

   ```
   READ file-name
   MOVE record-name TO identifier
   ```

   The MOVE operation takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the record is available both in the input area and in the area specified by the identifier. The length of the source field is determined by the length of the record that is read (see "RECORD clause", page 452).
   If the execution of the READ statement was unsuccessful, the implied MOVE statement does not occur.
   The index for the identifier is calculated after execution of the READ statement and immediately before the implicit MOVE.
6. If, after a READ statement without the INTO phrase, the user wishes to explicitly address the input area, then he is responsible for using the correct record description (i.e. the one which matches the length of the record which was read).
7. If the file position indicator indicates that no next logical record exists, or if a file specified as OPTIONAL in the select clause does not exist, the following occurs in the order specified:
   a) The I-O status (FILE STATUS) associated with file-name is set to indicate the at end condition.
   b) If the AT END phrase is specified, control is transferred to imperative-statement-1 in the AT END phrase. Any USE procedure declared for file-name is not executed.
   c) If the AT END phrase is not specified, a USE procedure must be declared. This procedure is then executed. On returning from this procedure, control is passed to the next executable statement following the end of the READ statement.

   When the at end condition occurs, execution of the READ statement is unsuccessful.
8. If an at end condition does not occur during the execution of a READ statement, the AT END phrase is ignored, if specified, and the following actions occur:
   a) The I-O status for file-name is updated.

b) If some other exception condition occurs, control is transferred to the USE procedure.

c) If no exception condition exists, the record is made available in the record area, and any implicit move as a result of the INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2 of the NOT AT END phrase, if specified. In the latter case, execution continues according to the rules for the specified imperative-statement, and control is transferred to some other branch in the program or to the end of the READ statement.

9. If the RELATIVE KEY phrase is specified, the execution of the READ statement moves the relative record number of the record made available to the relative key data item according to the rules for the MOVE statement.
10. After unsuccessful execution of a READ statement, the content of the input area belonging to the file is undefined and the I/O status indicates that no valid next record has been read.
11. If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for length, the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for length, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred.
12. After the at end condition has occurred, no further READ statement may be issued for the file before a successful CLOSE statement followed by a successful OPEN statement has been executed.
13. END-READ delimits the scope of the READ statement.

Format 2

```
READ file-name [WITH NO LOCK] RECORD [INTO identifier]

  [INVALID KEY imperative-statement-1]

  [NOT INVALID KEY imperative-statement-2]

  [END-READ]
```

Syntax rules for format 2

1. file-name and identifier must not reference the same storage area.

2. If no USE procedure is present for the file, INVALID KEY must be specified.

General rules

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.
2. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area.
3. The INTO phrase may be specified in the following cases:
   – if only one record description is subordinate to the file description entry, or
   – if all record-names associated with the file-name and the data item referenced by the identifier represent group items or alphanumeric elementary items.
4. The execution of a READ statement with the INTO phrase is equivalent to:

```
READ file-name
MOVE record-name TO identifier
```

   The MOVE operation takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the record is available both in the input area and in the area specified by the identifier. The length of the source field is determined by the length of the record that is read (see "RECORD clause", page 452).
   If the execution of the READ statement was unsuccessful, the implied MOVE statement does not occur.
   The index for the identifier is calculated after execution of the READ statement and immediately before the implicit MOVE.
5. If the input area is to be explicitly referenced following a READ statement without the INTO phrase, it is the user's responsibility to ensure that the correct record description entry (i.e. corresponding to the length of the read record) is used.
6. If an invalid key condition does not occur during the execution of a READ statement, the INVALID KEY phrase is ignored, if specified, and the following actions occur:
   a) The I-O status for file-name is updated.
   b) If some other exception condition occurs, control is transferred to the USE procedure.
   c) If no exception condition exists, the record is made available in the record area, and any implicit move as a result of the INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2 of the NOT INVALID

KEY phrase, if specified. In the latter case, execution continues according to the rules for the specified imperative-statement, and control is transferred to some other branch in the program or to the end of the READ statement.

7. Following an unsuccessful attempt to perform a READ statement, the contents of the record area associated with the file and the file position indicator are undefined.
8. If a file is read randomly, a search is executed for a record whose relative record number is equal to the value of the RELATIVE KEY field for this file. If there is no such record in the file, an invalid key condition occurs and execution of the READ statement is unsuccessful (see "Invalid key condition", page 455).
9. If an optional file does not exist, the invalid key condition occurs and execution of the READ statement is unsuccessful.
10. If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for length, the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for length, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred.
11. END-READ delimits the scope of the READ statement.

REWRITE statement

Function

The REWRITE statement replaces a logical record on a disk storage file.

Format

```
REWRITE record-name [FROM identifier]

      [INVALID KEY imperative-statement-1]

      [NOT INVALID KEY imperative-statement-2]

      [END-REWRITE]
```

Syntax rules

1. record-name and identifier must not refer to the same storage area.
2. record-name must be associated with a file description (FD) entry in the Data Division of the program and may be qualified.
3. The INVALID KEY and the NOT INVALID KEY phrases must not be specified for a REWRITE statement which references a relative file in sequential access mode.
4. For a REWRITE statement which does not reference a file in sequential access mode the INVALID KEY phrase must be specified unless an appropriate USE procedure has been declared.

General rules

1. record-name identifies the record to be replaced.
2. The file associated with record-name must be a disk file and be open in the I-O mode at the time the REWRITE statement is executed.
3. The length of the record to be replaced may be modified.
4. Execution of a REWRITE statement with the FROM phrase is equivalent to execution of the following statements:

```
MOVE identifier TO record-name
REWRITE record-name.
```

The content of the storage area described by record-name before the implicit MOVE statement is performed has no effect on the execution of the REWRITE statement.

Thus, when the FROM phrase is used, data is transferred from identifier to record-name and then released to the appropriate file. The identifier may be used to reference any data outside the current file description entry.

Transfer of data through the implicit MOVE statement takes place in accordance with the rules for a MOVE statement without the CORRESPONDING phrase. After the REWRITE statement is executed, the content of the record is still available in the area specified by identifier, although it is no longer available in the area specified by record-name.

5. The following rules apply to all files whose access mode is sequential:
   a) A REWRITE statement must be preceded by a successful READ statement as the last input/output statement for the associated file.
   b) Execution of the REWRITE statement causes the record accessed by the preceding READ statement to be replaced on that file.
   c) The content of the data item specified by means of the RELATIVE KEY phrase must not be modified between the READ and REWRITE statements.
6. In the case of files whose access mode is random or dynamic, the RECORD KEY must be supplied with an appropriate value prior to the execution of the REWRITE statement. In such cases, the REWRITE statement replaces the record according to the contents of the data item specified by RECORD KEY.
7. If the file does not contain a data item corresponding to the contents of the RELATIVE KEY item, the invalid key condition occurs (see "Invalid key condition", page 455). In this case, execution of the REWRITE statement is unsuccessful, no updating is executed, the contents of the record area remain unchanged, and the I-O status is set to a value which indicates an invalid key condition.
8. The record rewritten by a successful REWRITE statement is no longer accessible in the record area; an exception to this rule is the use of the SAME RECORD AREA clause. In this case, the record is available to all other files specified in the SAME RECORD AREA clause as well as to the current file.
9. Execution of a REWRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS clause", page 440).
10. The continuation of processing after successful or unsuccessful execution of a REWRITE statement depends on whether INVALID KEY or NOT INVALID KEY is specified (see "Invalid key condition", page 455).
11. When rewriting a record, the user should ensure that its first byte does not contain the value X’FF’, since this will cause logical deletion of the record.

12. The number of character positions in the record indicated by record-name must not be greater than the largest number of character positions or less than the smallest number of character positions permitted by the associated RECORD IS VARYING clause. Otherwise, the REWRITE statement will be unsuccessful, the update operation will not take place, the content of the record area remains unchanged and the I-O status of the file associated with record-name is set to a value indicating a record length conflict (see "I-O status", page 431).
13. END-REWRITE delimits the scope of the REWRITE statement.

START statement

Function

The START statement defines the logical starting point within a file for subsequent sequential read operations.

Format

```
START file-name [WITH NO LOCK]  [ KEY { IS EQUAL TO                          } file-name ]
                                      { 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 >                             }

      [INVALID KEY imperative-statement-1]

      [NOT INVALID KEY imperative-statement-2]

      [END-START]
```

The format extension WITH NO LOCK is effective during shared updating of files; it is described in the "COBOL85 User Guide" [1].

Syntax rules

1. Relational operators are mandatory separators. To avoid possible misinterpretations, they have not been underlined in the above format.
2. file-name must refer to a file in sequential or dynamic access mode.
3. data-name may be qualified.

   It must be the name of the RELATIVE KEY data item declared for this file.
4. The INVALID KEY phrase must be present if no applicable USE procedure has been specified for that file.

General rules

1. The file specified by file-name must be open in the INPUT or I-O mode at the time the START statement is executed (see "OPEN statement", page 460).
2. If the KEY phrase is omitted from the START statement, the relational operator EQUAL is assumed.
3. Execution of the START statement alters neither the content of the record area of the file nor the content of the data item in the file referenced by RELATIVE KEY.
4. The type of comparison is specified by the relational operator in the KEY phrase. The position of each record in the file specified by file-name is compared with the contents of the data item referenced by data-name (RELATIVE KEY).
   a) With the relational operators EQUAL, GREATER, GREATER OR EQUAL, NOT LESS and their equivalents, positioning takes place in the file to the first record that satisfies the relation condition.
   b) For the relational operators LESS, LESS OR EQUAL, NOT GREATER and their equivalents, positioning takes place in the file to the last record that satisfies the relation condition.
   c) If no record in the file satisfies the relation condition, an INVALID KEY condition occurs, and the START statement is unsuccessful.
5. Transfer of control following the successful or unsuccessful execution of the START statement depends on whether INVALID KEY or NOT INVALID KEY has been specified (see "Invalid key condition", page 455).
6. After execution of a START statement, the contents of the FILE STATUS data items (if specified) of that file are updated (see also "FILE STATUS clause", page 440).
7. The END-START phrase delimits the scope of the START statement.

USE statement

Function

The USE statement introduces declarative procedures which are to be executed if an I/O error occurs for a file.
The USE statement itself is not an executable statement.

Format

```
USE AFTER STANDARD {ERROR    } PROCEDURE ON {{file-name-1}...}
                   {          }            {INPUT          }
                   {EXCEPTION }            {               }.
                                           {OUTPUT         }
                                           {I-O            }
                                           {EXTEND         }
```

Syntax rules

1. The ERROR and EXCEPTION phrases are equivalent and may be used interchangeably.
2. Files referenced either implicitly (INPUT, OUTPUT, I-O and EXTEND) or explicitly (file-name-1) in the USE statement need not have the same organization and access mode.
3. file-name-1 may only be specified in one USE statement. Similarly, the INPUT, OUTPUT, I-O and EXTEND phrases may only be specified once.

General rules

1. When file-name-1 is specified, the error handling procedures are executed for the named files only. No further USE procedures are run for these files.
2. Before execution of the user error handling routine, the standard system routines for input/output error handling are executed.
3. INPUT indicates that the specified procedures are only executed for files opened in input mode (OPEN statement with the INPUT option).
4. OUTPUT indicates that the specified procedures are only executed for files opened in output mode (OPEN statement with the OUTPUT phrase).
5. I-O indicates that the specified procedures are executed only for files opened in update mode (OPEN statement with the I-O option).
6. EXTEND means that the specified procedures are to be executed only for files which are in extend mode (OPEN statement with the EXTEND phrase).

7. The USE procedures are executed when:

   a) an invalid key or at end condition occurs, if the input/output which encountered either of these conditions does not include an INVALID KEY or AT END phrase.

   b) a severe error occurs (FILE STATUS CODE ≥ 30).

   If no corresponding USE procedure exists for the file when a) or b) occurs, the program will abort.

8. After a USE procedure has been executed, control is returned to the calling routine.

9. Within a USE procedure, there must be no reference to nondeclarative procedures, with the exception of the PERFORM statement.

10. Reference to procedure-names which are subordinate to a USE statement may be made from another procedure or from a nondeclarative procedure by using a PERFORM statement only.

Example 5-1

```
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 RELKEY
    FILE STATUS INDICATOR.
DATA DIVISION.
FILE SECTION.
FD  WORKFILE.
01 REC.
    03 INHOLD PIC X(100).
WORKING-STORAGE SECTION.
01 INDICATOR PIC 99.
01  RELKEY PIC 9(4).
01  CLOSE-INDICATOR           PIC X VALUE "0".
    88 FILE-CLOSED VALUE "0".
    88 FILE-OPEN VALUE "1".
PROCEDURE DIVISION.

DECLARATIVES.
INPUT-ERROR SECTION.
    USE AFTER ERROR PROCEDURE ON WORKFILE.
STATUS-QUERY.
    EVALUATE INDICATOR
    WHEN    10
      DISPLAY "End of WORKFILE encountered" UPON T
    WHEN    22
      DISPLAY "Record with key" RELKEY "ALREADY EXISTS" UPON T
    WHEN    23
      DISPLAY "Record with key" RELKEY "NON-EXISTENT" UPON T
    WHEN OTHER
      DISPLAY "Unrecoverable error "(" INDICATOR")"
              "FOR FILE-INPUT" UPON T
      IF FILE-OPEN
      THEN
        CLOSE WORKFILE
      END-IF
      DISPLAY "Program terminated abnormally" UPON T
      STOP RUN
    END-EVALUATE.
END-DECLARATIVES.
MAIN SECTION.
OPEN-CLOSE.
    ...
    STOP RUN.
```

WRITE statement

Function

The WRITE statement initiates output of a record to an output or input/output file.

Format

```
WRITE record-name [FROM identifier-1]

     [INVALID KEY   imperative-statement-1]

     [NOT INVALID KEY imperative-statement-2]

     [END-WRITE]
```

Syntax rules

1. record-name and identifier-1 must not reference the same storage area.
2. record-name must be associated with a file description (FD) entry in the Data Division of the program and may be qualified.
3. The INVALID KEY phrase must be specified for a file unless an applicable USE procedure was declared.

General rules

1. The file whose record is referenced by the WRITE statement must be open in the OUTPUT, I-O or EXTEND mode.
2. When a WRITE statement is executed, the position within the file of the record to be output is located by means of the contents of the RECORD KEY data item. Before the WRITE statement is executed, the content of the associated key field must be set accordingly (see "ACCESS MODE clause", RELATIVE KEY phrase, page 438).
3. The record released by a WRITE statement is no longer available in the record area, unless the file associated with the record was specified in a SAME RECORD AREA clause, or the execution of the WRITE statement was abnormally terminated as unsuccessful because of the occurrence of an invalid key condition. The record is also available to those files which were referenced in a SAME RECORD AREA clause together with the specified file.

4. Execution of a WRITE statement with the FROM phrase is equivalent to execution of the following statements:

   ```
   MOVE identifier-1 TO record-name
   WRITE record-name
   ```

   Data is transferred according to the rules for a MOVE statement without the CORRESPONDING phrase.

   The contents of the record area before execution of the implicit MOVE statement has no effect on the execution of the WRITE statement.

5. Execution of a WRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS clause", page 440).

6. If, during the execution of a WRITE statement with the NOT INVALID KEY phrase, the invalid key condition does not occur, control is transferred to imperative-statement-2 as follows:

   a) If the execution of the WRITE statement is successful: after the record is written and after updating the I/O status of the file from which the record originates.

   b) If the execution of the WRITE statement is unsuccessful: after the I/O status of the file has been updated and after executing the USE procedure that was specified for the file from which the record originates.

7. If a file is opened in output mode (OPEN statement with OUTPUT phrase), the following should be noted:

   a) In sequential access mode, the WRITE statement causes output of a record for creation of a new file. The first record is assigned a relative record number of 1 (one) while the subsequent records are numbered 2, 3, 4,... If RELATIVE KEY was specified, the File Control Processor will enter the relative record number in the RELATIVE KEY phrase during the execution of the write statement.

   b) In random or dynamic access modes (which are equivalent for OUTPUT), the value of the data item in the (here mandatory) RELATIVE KEY phrase must be set by the user to equal the relative record number which is to be assigned to the record contained in the record area. That record is then output as the nth record of the file, where "n" is the value of the relative record number.

8. If a file is in extend mode (OPEN statement with EXTEND phrase), a WRITE statement causes a record to be appended to the file. The first record released in this manner receives a relative record number which is 1 higher than the highest existing relative record number in the file. The relative record number of each subsequent record is incremented by 1 (with respect to the previous record). If the RELATIVE KEY phrase is specified, the relative record number of the MOVE statement is transferred to the record key field with each WRITE statement.

9. If a file is opened in update mode (OPEN statement with I-O phrase), and its access mode is either random or dynamic (which means the same in this case), the WRITE statement inserts records in the associated existing file. The value of the data item of the (here mandatory) RELATIVE KEY phrase must be set by the user to equal the relative record number which is to be assigned to the record contained in the record area. As the WRITE statement is executed, the contents of the record with the appropriate record number are transferred to the File Control Processor.
10. The invalid key condition is caused by the events listed in Table 5-3.

ACCESS MODE clause	**OPEN mode and action taken**	**Reason for INVALID KEY condition**
SEQUENTIAL	OUTPUT or EXTEND A record is appended to an existing or newly created file.	There is no room in the file for the new record.
RANDOM or DYNAMIC	OUTPUT or I-O A record is appended to an existing file.	The content of the RELATIVE KEY field specifies a record which already exists in the file or there is no room in the file for the new record.

Table 5-3: WRITE statement - Causes of invalid key conditions

11. Occurrence of an invalid key condition indicates that the WRITE statement was unsuccessful; the contents of the record area are still available, and any existing FILE STATUS data item in that file is set to a value indicating the cause of the invalid key condition. The program resumes execution in accordance with the rules for the invalid key condition.
12. The number of character positions in the record referenced by record-name must not be larger than the largest or smaller than the smallest number of character positions allowed by the associated RECORD IS VARYING clause. Otherwise, the WRITE statement is unsuccessful, the WRITE operation does not take place, the content of the record area is unaffected and the I-O status of the file associated with record-name is set to a value indicating a record length conflict (see "I-O status", page 431).
13. END-WRITE delimits the scope of the WRITE statement.

6 Indexed file organization

6.1 File concepts

A file is a collection of records that can be transferred to, or read from, a volume. The user defines the organization of the file as well as the mode and order in which the records are processed.

The organization of a file describes its logical structure. There are sequential, indexed, and relative types of file organization. The file organization which is defined at the time a file is created cannot be changed later on.

6.1.1 Indexed organization

When indexed file organization is used, the position of each logical record in the file is determined by indices which are generated with the file and are maintained by the system. These indices are based on keys which must be supplied by the user in the records. Indexed files must be assigned to disk storage devices.

The RECORD KEY clause must be specified when creating an indexed file. This clause defines which data item within the record is to be used as the primary key.

The ALTERNATE RECORD KEY clause can be used to define one or more alternate keys (secondary keys) in addition to the primary key.

The START statement can be used to define a starting point, within an indexed file, for a series of subsequent sequential access operations.

6.1.2 Sequential access to records

Records in an indexed file may be sequentially created, read and updated.

The records are read in ascending order of their keys.

6.1.3 Random access to records

Records of indexed files may be randomly created, read, and updated.

The data item defined as the key is the data-name specified in the RECORD KEY clause.

When a new record is created, the value of the RECORD KEY should not be identical to the value of a key field that is already in existence.

6.1.4 Dynamic access to records

In dynamic access mode, the user may, by using the appropriate I/O statements, switch as required between sequential and random access.

6.1.5 I-O status

The I-O status is a value that can be used in a COBOL program to check the status of an input/output operation. In order to do this, the FILE STATUS clause must be specified in the FILE CONTROL paragraph of the Environment Division.
The I-O status value is transferred to a two-character data item

– during the execution of a CLOSE, OPEN, READ, REWRITE, or WRITE statement,
– prior to the execution of any associated imperative statement, and
– prior to the execution of any corresponding USE AFTER STANDARD EXCEPTION procedure.

The table below shows I-O status values and their meanings:

I-O status	Meaning
	Execution successful
00	The I-O statement terminated normally. No further information regarding the I-O operation is available.
02	A record was read with ALTERNATE KEY and subsequent sequential reading with the same key has found at least one record with an identical key. A record was written with ALTERNATE KEY WITH DUPLICATES and there is already a record with an identical key value for at least one alternate key.
04	Record length conflict: A READ statement terminated normally. However, the length of the record read lies outside the limits defined in the record description entry for the given file.
05	An OPEN statement was executed for an OPTIONAL file which does not exist.

I-O status	Meaning
	Execution unsuccessful: at end condition
10	An attempt was made to execute a sequential READ operation. However, no next logical record was available, as the end-of-file was encountered.
	Execution unsuccessful: invalid key condition
21	File sequence error in conjunction with ACCESS MODE IS SEQUENTIAL: 1. The record key value was changed between the successful execution of a READ statement and the execution of the next REWRITE statement for a file, or 2. the ascending sequence of record keys was violated in successive WRITE statements.
22	Duplicate key An attempt was made to execute a WRITE statement with a primary key for which there is already a record in the indexed file. An attempt was made to create a record with ALTERNATE KEY, but without WITH DUPLICATES, and there is already an alternate key with the same value in the file.
23	Record not located An attempt was made (using a READ, START, DELETE or REWRITE statement with key) to access a record not containing in the file.
24	Boundary values exceeded An attempt was made to execute a WRITE statement beyond the system-defined boundaries of an indexed file (see "COBOL85 User Guide" [1]).
	Execution unsuccessful: unrecoverable error
30	No further information regarding the I-O operation is available (the DMS code provides further information).
35	An OPEN statement with the INPUT, I-O or EXTEND phrase was issued for a non-optional file which does not exist.
37	OPEN statement on a file that cannot be opened due to the following violations: 1. OPEN OUTPUT/I-O/EXTEND on a write-protected file (password, RETPD in catalog, ACCESS=READ in catalog) 2. OPEN INPUT on a read-protected file (password)
38	An attempt was made to execute an OPEN statement for a file previously closed with the LOCK phrase.
39	The OPEN statement was unsuccessful as a result of one of the following conditions: 1. One or more of the operands ACCESS-METHOD, RECORD-FORMAT, RECORD-SIZE or KEY-LENGTH were specified in the SET-FILE-LINK command with values that conflict with the corresponding explicit or implicit program specifications. 2. Record length error occurred for an input file (catalog check, if RECFORM=F). 3. The record size is greater than the BLKSIZE entry in the catalog of an input file.

I-O status	Meaning
	4. The catalog entry of one of the FCBTYPE, RECFORM, RECSIZE (if RECFORM=F), KEYPOS, or KEYLEN operands for an input file is in conflict with the corresponding explicit or implicit program specifications or with the corresponding specifications in the FILE command. 5. An attempt was made to open a file whose alternate key does not match the key values specified in the ALTERNATE RECORD KEY clause in the program.
	Execution unsuccessful: logical error
41	An attempt was made to execute an OPEN statement for a file which was already open.
42	An attempt was made to execute a CLOSE statement for a file which was not open.
43	For ACCESS MODE IS SEQUENTIAL: The most recent I-O statement executed prior to a DELETE or REWRITE statement was not a successfully executed READ statement.
44	Record length limits exceeded: An attempt was made to execute a WRITE or REWRITE statement. However, the length of the record is outside the range allowed for this file.
46	An attempt was made to execute a sequential READ statement for a file in INPUT or I-O mode. However, no valid next record is available since: 1. the preceding START statement was unsuccessful, or 2. the preceding READ statement was unsuccessful without leading to an at end condition, or 3. an attempt was made to execute a READ statement after the at end condition was encountered.
47	An attempt was made to execute a READ or START statement for a file that is not open in INPUT or I-O mode.
48	An attempt was made to execute a WRITE statement for a file that is not in OUTPUT, I-O or EXTEND mode.
49	An attempt was made to execute a DELETE or REWRITE statement for a file that is not in I-O mode.
	Other conditions with unsuccessful execution
90	System error; no further information regarding the cause is available.
91	OPEN error: the actual cause is evident from the DMS code (see "FILE STATUS clause" specifying data-name-2), page 491).
93	For shared update processing only (see "COBOL85 User Guide" [1], "Shared updating of files"): The I-O statement could not terminate normally because a different task is accessing the same file, and the access operations are incompatible:

I-O status	Meaning
94	1. For shared update processing only (see "COBOL85 User Guide" [1], "Shared updating of files"): deviation from call sequence READ - REWRITE/DELETE. 2. The record size is greater than the block size.
95	The specifications in the BLOCK-CONTROL-INFO or BUFFER-LENGTH operands of the SET-FILE-LINK command are not consistent with the file format, the block length, or the format of the volume being used.

6.2 Language elements of the Environment Division

INPUT-OUTPUT SECTION

Function

The INPUT-OUTPUT SECTION deals with the following:

– the definition of each file used in the program,

– the assignment of these files to external devices, and

– the transmission and handling of data between external devices and the object program.

This section is divided into two paragraphs:

– the FILE-CONTROL paragraph, which names the files used in the program and assigns them to external devices, and

– the I-O-CONTROL paragraph, which defines special input/output techniques.

Format

```
 A       Margin indication
 |
-|-------------------------------------------------------------------------------
 |
 ↓
 INPUT-OUTPUT SECTION.

 FILE-CONTROL. {file-control-entry}...
[I-O-CONTROL.  [input-output-control-entry]]
-------------------------------------------------------------------------------
```

General rules

1. All sections and paragraphs must begin in area A.
2. The INPUT-OUTPUT SECTION is optional. If the INPUT-OUTPUT SECTION is defined, the FILE-CONTROL paragraph must also be specified.

FILE-CONTROL paragraph

Function

The FILE-CONTROL paragraph is used to give each file a name. The files are assigned to one or more external devices, and the information required for file processing is made available. This information indicates how the data is organized and how it is to be accessed.

Format

```
 A   B     Margin indication
 |   |
 |   |
 ↓   ↓
 FILE-CONTROL.

     SELECT clause
     ASSIGN clause
    [ACCESS MODE clause]
    [ALTERNATE RECORD KEY clause]
    [FILE STATUS clause]
     ORGANIZATION clause
     RECORD KEY clause
    [RESERVE clause] .
```

Syntax rules

1. The FILE-CONTROL heading must be written in area A; all subsequent entries must be written in area B.
2. The SELECT clause must be the first entry in the FILE-CONTROL paragraph. Clauses that follow the SELECT clause may appear in any order.

In the pages that follow, the SELECT and ASSIGN clauses are described first, followed by the remaining clauses in alphabetical order.

SELECT clause

Function

The SELECT clause is used to give a name to each file in the program.

Format 1 applies to all files except sort files.

Format 2 is suitable for sort files

(see chapter 10, "Sorting of records", page 652).

Format 1

```
SELECT [OPTIONAL] file-name
```

Syntax rules and general rules

See SELECT clause for sequential files, page 361.

ASSIGN clause

Function

The ASSIGN clause assigns an external device to a file of the COBOL program. One ASSIGN clause is required for each file in the program.

Format

```
ASSIGN TO {literal-1   }...
          {data-name-1}
```

Syntax rules

1. literal-1 or the contents of data-name-1 specifies the link name for the file. The name must be alphanumeric, must be specified in uppercase letters, and must not be a figurative constant. The link name is formed from the first 8 characters of this literal, and must therefore be unique within the program. If the 8th character is a hyphen (-), it is replaced by a # character.
2. data-name-1 must not be qualified.
3. data-name-1 must be defined as an alphanumeric data item or as a group item in the WORKING-STORAGE SECTION or LINKAGE SECTION.

General rules

1. The type of file organization must be specified in the ORGANIZATION clause (see page 492).
2. If a number of literals are specified, only the first will be evaluated; the remaining literals are ignored.
3. If the ASSIGN clause refers to an *external* file, the ASSIGN clause must be used in the same form in all programs that describe this external file. The contents of the literal or data-name may, however, be different.

ACCESS MODE clause

Function

The ACCESS MODE clause determines the manner in which the records of a file are to be accessed.

Format

```
ACCESS MODE IS {SEQUENTIAL}
               {RANDOM    }
               {DYNAMIC   }
```

General rules

1. SEQUENTIAL means that records are read or written sequentially, i.e. the next logical record of the file is made available when a READ statement is executed, or the next logical record is placed on that file when a WRITE statement is executed.
2. RANDOM means that records are randomly read or written on the basis of a key, i.e. access is made using the RELATIVE KEY.
3. DYNAMIC means that random and/or sequential access is possible.
4. If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is assumed.
5. If the ACCESS MODE clause is specified for an *external* file, an equivalent ACCESS MODE clause must be specified in all other programs that describe this external file.

ALTERNATE RECORD KEY clause

Function

The ALTERNATE RECORD KEY clause defines a further, alternative key, in addition to the primary key, which can be used to access the records of an indexed file.

Format

```
ALTERNATE RECORD KEY IS data-name [WITH DUPLICATES]
```

Syntax rules

1. data-name must specify an alphanumeric data item in a record description entry, and this record description entry must be associated with the file name to which the ALTERNATE RECORD KEY clause is subordinate.
2. data-name may be any fixed-length field, with a length of 1 to 127 bytes, within the record. data-name may be qualified, but not subscripted or indexed.
3. data-name may not reference a group item which contains an element with an OCCURS clause with DEPENDING ON phrase.
4. If the file contains variable-length records, the alternate key must be contained within the first x character positions of the record, where x equals the minimum record size specified for the file (see "RECORD clause", page 390), i.e. the length of the alternate key must fit into the shortest possible record.
5. Several (up to 30) alternate record keys may be specified for one file.
6. data-name must not refer to a data item whose start address (the leftmost character position) is identical with the start address of the primary record key or of any other alternate record key. Apart from this restriction, the primary record key and any alternate record key(s) may overlap.

General rules

1. The data description and the relative position within a record of data-name must be the same as those specified when the file was created. The number of alternate record keys must match the number specified when the file was created.
2. WITH DUPLICATES specifies that the value of the related alternate record key may occur in more than one record. Records with identical key values are also called "duplicates". If WITH DUPLICATES is not specified, the value of the related alternate key must be unique, i.e. it may not occur more than once in the file.

3. If there are several record description entries in a file, data-name needs to be described in only one of these entries. The position of the key field in the record defined by data-name is used implicitly for all keys in the other record description entries.
4. If the ALTERNATE RECORD KEY clause(s) refers/refer to an *external* file, the same number of ALTERNATE RECORD KEY clauses must be specified in any program that uses this external file; the length and position of the key fields in the record and, where applicable, the DUPLICATES phrase, must also be defined so as to be consistent.

FILE STATUS clause

Function

The FILE STATUS clause specifies a data item that indicates the status of input/output operations during processing. In addition, by specifying a further item, an additional error code is made available.

Format

```
FILE STATUS IS data-name-1 [, data-name-2]
```

Syntax rules

1. data-name-1 and data-name-2 must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION of the Data Division.
2. data-name-1 must be a two-byte numeric (USAGE DISPLAY only) or alphanumeric item.
3. data-name-2 must be a 6-byte group item with the following format:

```
01  data-name-2.
    02  data-name-2-1  PIC 9(2) COMP.
    02  data-name-2-2  PIC X(4).
```

General rules

1. If the FILE STATUS clause is specified, the runtime system copies the I-O status to data-name-1.
2. If specified, data-name-2 is assigned as follows:t
   a) If data-name-1 has the value 0, the contents of data-name-2 are undefined.
   b) If data-name-1 has a non-zero value, data-name-2 contains the additional error code. The value 64 in data-name-2 indicates that this code is the (BS2000) DMS code; the value 96 in data-name-2 indicates that the code is the (POSIX) SIS code. The command HELP DMS <contents-of-data-name-2-2> or HELP SIS <contents-of-data-name-2-2> supplies more detailed information on the corresponding error code.
3. The I-O status is copied during the execution of each OPEN, CLOSE, READ, WRITE, or REWRITE statement that references the specified file, and prior to the execution of each corresponding USE procedure (see "I-O status", page 480).

ORGANIZATION clause

Function

The ORGANIZATION clause defines the logical structure of a file.

Format

```
[ORGANIZATION IS] INDEXED
```

General rule

The file organization is defined at the time a file is created and cannot be changed subsequently.

RECORD KEY clause

Function

The RECORD KEY clause defines the primary record key used for accessing the records in an indexed file.

Format

```
RECORD KEY IS data-name
```

Syntax rules

1. data-name must be an alphanumeric data item defined within a record description entry. This record description entry must be assigned to the file-name to which the RECORD KEY clause is subordinate.
2. data-name may be any fixed-length item within the record. The item may be 1 through 255 bytes in length. The data-name may be qualified, but not indexed or subscripted.
3. data-name may not reference a group item which contains an element with an OCCURS clause with DEPENDING ON phrase.
4. If the file contains variable-length records, the record key must be contained in the first x character positions in the record, where x is the minimum record length for this file (see "RECORD clause", page 390). This means that the length of the record key must be fully included within the shortest record.

General rules

1. Record key values must be unique for all records within the file.
2. The data description and the relative position within a record of data-name must be the same as those specified when the file was created. The number of alternate record keys must match the number specified when the file was created.
3. If there are several record description entries in a file, data-name needs to be described in only one of these entries. The position of the key field in the record defined by data-name is used implicitly for all keys in the other record description entries.
4. If the RECORD KEY clause refers to an *external* file, the equivalent RECORD KEY clause must be specified in any program that uses this external file; in particular, the length and position of the key fields in the record must be defined so as to be consistent.

RESERVE clause

Function

The RESERVE clause allows the user to modify the number of input/output areas (buffers) to be allocated to the program by the compiler.

Format

```
RESERVE integer  [AREA ]
                 [AREAS]
```

The RESERVE clause is treated as a comment by the compiler.

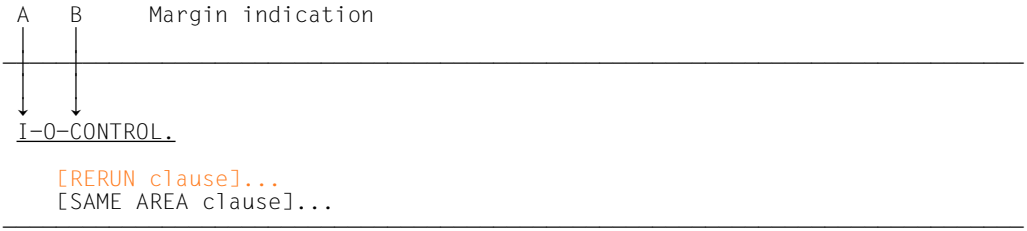
I-O-CONTROL paragraph

Function

The I-O-CONTROL paragraph defines the events which, if they occur, cause restart points to be established. It may also specify a memory area which is to be shared by different files.

In addition, it defines special input/output conditions.

Format



Syntax rule

I-O-CONTROL must be written starting in area A. All subsequent entries must be written in area B.

RERUN clause

Function

A RERUN clause indicates where and when restart-point records are to be issued. A restart-point record describes the status of a program at a specified point during program execution. It is produced automatically by the operating system upon the request of the program and contains all information necessary to restart the program from that point. The RERUN clause controls such requests by the COBOL object program.

Format

```
RERUN [ ON { implementor-name }  ]
       [   { file-name-1      }  ]

      EVERY { integer-1 RECORDS OF file-name-2 }
            { integer-2 CLOCK-UNITS            }
            { condition-name                   }
```

Syntax rules and general rules

See RERUN clause for sequential files, page 372.

SAME AREA clause

Function

The SAME AREA clause indicates that a number of files are to share a specified input/output area during program execution.

Format 1 applies to all files except sort files, unless RECORD is specified

Format 2 is suitable for sort files

(see chapter 10, "Sorting of records", page 650).

Format 1

```
SAME [RECORD] AREA FOR file-name-1 {file-name-2}...
```

Syntax rules and general rules

See SAME-AREA clause for sequential files, page 374.

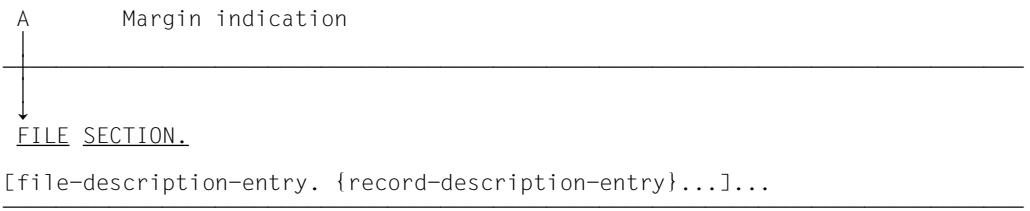
6.3 Language elements of the Data Division

FILE SECTION

Function

The FILE SECTION is used to define the structure of files. Each file is defined by a file description entry and one or more record description entries. Record description entries are written immediately following the file description entry.

Format



File description entries are discussed below. Record description entries are discussed in chapter 3 (page 141ff) and on the following pages.

File description (FD) entry

Function

The file description (FD) entry specifies the physical structure. and the record names of a given file.

A file description entry must be written for each file to be processed by the program. The information contained in this entry generally pertains to the physical aspects of the file, i.e. the description of data as it appears on the input or output medium.

Format

```
A   B     Margin indication
|   |
|   |
|   |
|   |
FD  file-name

    [BLOCK CONTAINS clause]
    [DATA RECORDS clause]
    [EXTERNAL clause]
    [GLOBAL clause]
    [LABEL RECORDS clause]
    [RECORD clause]
```

Syntax rules

1. The level indicator FD identifies the beginning of a file description entry.
2. file-name must be identical to the file-name given in a SELECT clause.
3. Clauses that follow the file-name may appear in any order.
4. The file description entry must be followed by one or more record description entries.

General rules

1. Table 6-1 provides a summary of the functions of clauses used in file description entries.

Clause	Function
BLOCK CONTAINS	Specifies physical block length
DATA RECORDS	Specifies the names of the records in the file
EXTERNAL	Declares a file as external
GLOBAL	Declares a file as global
LABEL RECORDS	Gives the names and values of the label records contained in the file
RECORD	Specifies logical record size

Table 6-1: Functions of file description clauses

2. The EXTERNAL and GLOBAL clauses are described in chapter 7, "Inter-program communication" (page 544 and page 547). The other file description clauses are described below in alphabetical order.

BLOCKS CONTAINS clause

Function

The BLOCK CONTAINS clause specifies the maximum size of a physical block.

Format

```
BLOCK CONTAINS [integer-1 TO] integer-2 {CHARACTERS}
                                        {RECORDS   }
```

Syntax rules and general rules

See BLOCK CONTAINS clause for sequential files, page 379.

DATA RECORDS clause

Function

The DATA RECORDS clause is used only for documentation. It specifies the names of the records in a file.

Format

```
DATA {RECORD  IS } {data-name-1}...
     {RECORDS ARE}
```

Syntax rules and general rules

See DATA RECORDS clause for sequential files, page 383

LABEL RECORDS clause

Function

The LABEL RECORDS clause specifies whether labels are present and, if so, indicates whether they are conforming labels.

Format

Syntax rules and general rules

See LABEL RECORDS clause for sequential files, page 384.

RECORD clause

Function

The RECORD clause defines the length of the records in a file.

Format 1 Indicates fixed-length records by specifying the number of character positions in a record.

Format 2 Indicates variable-length records whose record size must lie within a certain range.

Format 3 Indicates variable-length records, specifying the minimum and maximum number of character positions.

The length of each record is defined by its record description entry. If the RECORD clause is specified, the record length is compared with the entries in this clause (see also "RECORDING MODE clause", page 394).

The following applies to all formats:

If the RECORD clause is specified for an *external* file, a RECORD clause must be specified in all programs that describe this external file; the minimum and maximum record lengths calculated from the specifications in the RECORD clause or the corresponding record description entries must be consistent.

Format 1

```
RECORD CONTAINS integer-1 CHARACTERS
```

Syntax rules and general rules

See RECORD clause for sequential files, page 390.

Format 2

```
RECORD IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]

    [DEPENDING ON data-name-1]
```

Syntax rules and general rules

See RECORD clause for sequential files, page 390.

Format 3

```
RECORD CONTAINS integer-4 TO integer-5 CHARACTERS
```

Syntax rules and general rules

See RECORD clause for sequential files, page 392.

6.4 Language elements of the Procedure Division

6.4.1 Invalid key condition

The invalid key condition may occur after the execution of a START, READ, WRITE, REWRITE or DELETE statement. Details concerning the occurrence of this condition are described under each of these statements.

If an invalid key condition was encountered, the File Control Processor performs the following actions in the specified order.

1. If a FILE STATUS clause exists for the file, a corresponding value is entered into the FILE STATUS data item in order to indicate the invalid key condition (see "FILE STATUS clause", page 491).
2. If the INVALID KEY phrase is specified in the input/output statement, any USE procedure associated with the file connector is not executed; control is passed to the imperative-statement specified in the INVALID KEY phrase instead. Execution then continues according to the rules for that statement. Depending on the type of statement specified, control is transferred to some other branch of the program or to the end of the input/output statement, and the NOT INVALID KEY phrase, if specified, is ignored.
3. If the INVALID KEY phrase is not specified, a USE procedure must be declared. The specified procedure is then executed. The NOT INVALID KEY is ignored, if specified.

If the invalid key condition does not exist after the execution of an input/output statement, the INVALID KEY phrase is ignored, if specified. The I-O status is updated, and the following actions occur:

1. If an exception condition which is not an invalid key condition exists, control is transferred to the USE procedure.
2. If no exception condition exists, control is transferred to the end of the input/output statement or to the imperative-statement of the NOT INVALID KEY phrase, if it is specified. In the latter case, execution continues according to the rules for the specified imperative-statement, and control is transferred to some other branch in the program or to the end of the input/output statement.
3. If neither INVALID KEY is specified nor a USE procedure declared, the program is aborted with the standard error handling if an invalid key condition or an exception condition occurs.

6.4.2 Input-output statements

In COBOL, input and output is record-oriented. Thus, the READ, WRITE, DELETE and REWRITE statements process records. The COBOL user is therefore only concerned with the processing of single records. The following operations are performed automatically: moving data into input/output areas (buffers) and/or internal storage, validity checking, error correction (where feasible), blocking, and deblocking.

Summary

Statement	Function
CLOSE	Terminates processing of a file
DELETE	Removes a record
OPEN	Opens a file for processing
READ	Reads a record
REWRITE	Replaces a record
START	Positions within a file
USE	In addition to input/output statements, USE statements may be specified to indicate label and error handling procedures (see "Declaratives subdivision of the Procedure Division", page 213)
WRITE	Writes a record

CLOSE statement

Function

The CLOSE statement terminates the processing of files, with optional lock where applicable.

Format

```
CLOSE {file-name-1 [WITH LOCK]}...
```

Syntax rules and general rules

See the CLOSE statement for relative file organization, page 457.

DELETE statement

Function

A DELETE statement logically removes a record from a disk storage file.

Format

```
DELETE file-name RECORD

    [INVALID KEY imperative-statement-1]

    [NOT INVALID KEY imperative-statement-2]

    [END-DELETE]
```

Syntax rules

1. The INVALID KEY phrase must not be specified for a DELETE statement which references a file in sequential access mode.
2. If a file is not in sequential access mode and no applicable USE procedure has been specified, the INVALID KEY phrase is mandatory.

General rules

1. The file referenced in the DELETE statement must be open in I-O mode during the execution of this statement (see also "OPEN statement", page 511).
2. After successful execution of the DELETE statement, the identified record is deleted from the file.
3. Execution of a DELETE statement does not affect the contents of the record area associated with the file or the contents of the data item referenced by the data-name specified in the DEPENDING ON phrase of the RECORD clause.
4. For a file in sequential access mode, the last input/output statement entered prior to the DELETE statement must be a successfully completed READ statement. The RECORD KEY must not be changed between reading and deletion. Execution of the DELETE statement causes the record read by the preceding READ statement to be logically removed from that file.

5. For a file in random or dynamic access mode, the File Control Processor deletes the record identified by the contents of the data item specified in the RECORD KEY clause of the file. If the record referenced by the key does not exist on the file, an invalid key condition occurs (see "Invalid key condition", page 506).
6. After execution of the DELETE statement, the contents of the data item specified in the FILE STATUS clause for the file are updated (see "FILE STATUS clause", page 491).
7. Transfer of control following the execution of the DELETE statement depends on whether the INVALID KEY or NOT INVALID KEY phrase is specified (see "Invalid key condition", page 506).
8. END-DELETE delimits the scope of the DELETE statement.

OPEN statement

Function

The OPEN statement opens files for processing.

Format

```
      ⎧INPUT {file-name-1}...  ⎫
      ⎪OUTPUT {file-name-2}... ⎪
OPEN  ⎨                        ⎬...
      ⎪I-O {file-name-3}...    ⎪
      ⎩EXTEND {file-name-4}... ⎭
```

Syntax rules and general rules

See the OPEN statement for relative file organization, page 460.

READ statement

Function

The READ statement makes the following available to the program:

– the next record in the file when access is sequential

– a particular record in a disk file (specified by the key in the record) when access is random.

Format 1 is used for the sequential reading of records from all files in sequential or dynamic access mode.

Format 2 is used for the random (RECORD KEY or ALTERNATE RECORD KEY) reading of records from all files in random or dynamic access mode.

The extension WITH NO LOCK in both formats is effective during shared updating of files; it is described in the "COBOL85 User Guide" [1] (see under "SHARED-UPDATE").

Format 1

```
READ file-name [WITH NO LOCK] [NEXT] RECORD [INTO identifier]

   [AT END imperative-statement-1]

   [NOT AT END imperative-statement-2]

   [END-READ]
```

Syntax rules for format 1

1. file-name and identifier must not reference the same storage area.
2. If no USE procedure is declared for the file, AT END must be specified in the READ statement.
3. NEXT must be specified if records of a file in dynamic access mode are to be read sequentially and neither AT END nor NOT AT END is specified.

General rules

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.
2. The NEXT phrase is optional in the sequential access mode and has no significance.

3. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current record are undefined after execution of the READ statement.
4. The INTO phrase may be specified in a READ statement:
   – if only one record description is subordinate to the file description entry, or
   – if all record-names associated with the file-name and the data item referenced by the identifier represent group items or alphanumeric elementary items.
5. The execution of a READ statement with the INTO phrase is equivalent to:

   ```
   READ file-name
   MOVE record-name TO identifier
   ```

   The MOVE operation takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the record is available both in the input area and in the area specified by the identifier. The length of the source field is determined by the length of the record that is read (see "RECORD clause", page 390).
   If the execution of the READ statement was unsuccessful, the implied MOVE statement does not occur.
   The index for the identifier is calculated after execution of the READ statement and immediately before the implicit MOVE.
6. If, after a READ statement without the INTO phrase, the user wishes to explicitly address the input area, then he is responsible for using the correct record description (i.e. the one which matches the length of the record which was read).
7. If the file position indicator indicates that no next logical record exists, or if a file specified as OPTIONAL in the select clause does not exist, the following occurs in the order specified:
   a) The I-O status (FILE STATUS) associated with file-name is set to indicate the at end condition.
   b) If the AT END phrase is specified, control is transferred to imperative-statement-1 in the AT END phrase. Any USE procedure declared for file-name is not executed.
   c) If the AT END phrase is not specified, a USE procedure must be declared. This procedure is then executed. On returning from this procedure, control is passed to the next executable statement following the end of the READ statement.

   When the at end condition occurs, execution of the READ statement is unsuccessful.
8. If an at end condition does not occur during the execution of a READ statement, the AT END phrase is ignored, if specified, and the following actions occur:
   a) The I-O status for file-name is updated.

b) If some other exception condition occurs, control is transferred to the USE procedure.

c) If no exception condition exists, the record is made available in the record area, and any implicit move as a result of the INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2 of the NOT AT END phrase, if specified. In the latter case, execution continues according to the rules for the specified imperative-statement, and control is transferred to some other branch in the program or to the end of the READ statement.

9. Following the unsuccessful execution of a READ statement, the content of the record area associated with the file is undefined, and the I-O status indicates that no valid next record has been established.
10. If the number of character positions in the record that is read is less than the minimum size specified by the record description entries for length, the portion of the record area which is to the right of the last valid character read is undefined. If the number of character positions in the record that is read is greater than the maximum size specified by the record description entries for length, the record is truncated on the right to the maximum size. In either of these cases, the READ statement is successful and an I-O status is set indicating a record length conflict has occurred.
11. After the at end condition has occurred, no further READ statement may be issued for the file until either repositioning has been effected with START or the file has been successfully closed and reopened.
12. END-READ delimits the scope of the READ statement.

Format 2

```
READ file-name [WITH NO LOCK] RECORD [INTO identifier]

  [KEY IS data-name]

  [INVALID KEY imperative-statement-1]

  [NOT INVALID KEY imperative-statement-2]

  [END-READ]
```

Syntax rules for format 2

1. file-name and identifier must not reference the same storage area.
2. If no USE procedure is present for the file, INVALID KEY must be specified.

3. data-name must be the name of a RECORD KEY or ALTERNATE RECORD KEY field declared for this file.
4. data-name may be qualified.

General rules

1. An OPEN statement with the INPUT or I-O phrase must be executed for a file before the READ statement can be executed.
2. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area.
3. The INTO phrase may be specified in a READ statement:
   - if only one record description is subordinate to the file description entry, or
   - if all record-names associated with the file-name and the data item referenced by the identifier represent group items or alphanumeric elementary items.
4. The execution of a READ statement with the INTO phrase is equivalent to:

   ```
   READ file-name
   MOVE record-name TO identifier
   ```

   The MOVE operation takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. After the READ statement with INTO phrase has been successfully executed, the record is available both in the input area and in the area specified by the identifier. The length of the source field is determined by the length of the record that is read. If the execution of the READ statement was unsuccessful, the implied MOVE statement does not occur.

   The index for the identifier is calculated after execution of the READ statement and immediately before the implicit MOVE.
5. If the input area is to be explicitly referenced following a READ statement without the INTO phrase, it is the user's responsibility to ensure that the correct record description entry (i.e. corresponding to the length of the read record) is used.
6. Execution of the READ statement sets the file position indicator to value in the key of reference. This value is compared with the value contained in the corresponding data item of the stored records in the file until the first record having an equal value is found. In the case an alternate key is used as key of reference and if duplicates are available for this alternate key, the first record written is made available with this very key value. The record thus found is made available in the record area associated with file-name. If no record can be identified in this way, the invalid key condition occurs, and execution of the READ statement is unsuccessful (see "Invalid key condition", page 506).

7. If an invalid key condition does not occur during the execution of a READ statement, the INVALID KEY phrase is ignored, if specified, and the following actions occur:

   a) The I-O status for file-name is updated.

   b) If some other exception condition occurs, control is transferred to the USE procedure.

   c) If no exception condition exists, the record is made available in the record area, and any implicit move as a result of the INTO phrase is executed. Control is transferred to the end of the READ statement or to imperative-statement-2 of the NOT INVALID KEY phrase, if specified. In the latter case, execution continues according to the rules for the specified imperative-statement, and control is transferred to some other branch in the program or to the end of the READ statement.

8. Following an unsuccessful READ statement, the contents of the record area associated with the file, the record key, and the file position indicator are undefined.

9. If KEY is specified, data-name is established as the key of reference for the current read operation. In the case of dynamic access, this key of reference is also used for all subsequent format 1 READ statements until a different key of reference is established for the file.

10. If KEY is not specified, the primary key is established as the key of reference. In the case of dynamic access, this key of reference is also used for all subsequent format 1 READ statements until a different key of reference is established for the file.

11. If the number of character positions in a record that is read is greater than the maximum size specified by a record description, the record will be truncated accordingly; if the number is less than the minimum size specified by a record description, the contents to the right of the last valid character will be undefined (for the length of a record description see "RECORD clause", page 390). In either case the read operation is successful, but a FILE STATUS is set to indicate the occurrence of a record length conflict.

12. If the file position indicator was established by a previous READ statement, and the current alternate key permits duplicates, the record selected is the one

    – whose key value is equal to the file position indicator and which is logically located, in the set of duplicates, immediately after the record made available by that previous READ statement, or

    – whose key value is greater than the file position indicator.

13. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they were written into the file with WRITE or REWRITE statements which create such duplicate values.

14. END-READ delimits the scope of the READ statement.

REWRITE statement

Function

The REWRITE statement replaces a logical record on a disk storage file.

Format

```
REWRITE record-name [FROM identifier]

      [INVALID KEY imperative-statement-1]

      [NOT INVALID KEY imperative-statement-2]

      [END-REWRITE]
```

Syntax rules

1. record-name and identifier must not refer to the same storage area.
2. record-name must be associated with a file description (FD) entry in the Data Division of the program and may be qualified.
3. The INVALID KEY phrase must be specified, unless an applicable USE procedure was declared.

General rules

1. record-name identifies the record to be replaced.
2. The file associated with record-name must be a disk file and be open in I-O mode at the time the REWRITE statement is executed.
3. The length of the record to be replaced on the file may be modified.
4. Execution of a REWRITE statement with the FROM phrase is equivalent to execution of the following statements:

   ```
   MOVE identifier TO record-name
   REWRITE record-name.
   ```

   The content of the storage area described by record-name before the implicit MOVE statement is performed has no effect on the execution of the REWRITE statement.

Thus, when the FROM phrase is used, data is transferred from identifier to record-name and then released to the appropriate file. The identifier may be used to reference any data outside the current file description entry.

Transfer of data through the implicit MOVE statement takes place in accordance with the rules for a MOVE statement without the CORRESPONDING phrase. After the REWRITE statement is executed, the content of the record is still available in the area specified by identifier, although it is no longer available in the area specified by record-name.

5. The following rules apply to all files whose access mode is sequential:
   a) A REWRITE statement must be preceded by a successful READ statement as the last input/output statement for the associated file.
   b) The content of the data item specified by means of the RECORD KEY clause must not be modified between the READ and REWRITE statements.
   c) Execution of the REWRITE statement causes the record accessed by the preceding READ statement to be replaced on that file.
6. In the case of files whose access mode is random or dynamic, the RECORD KEY must be supplied with an appropriate value prior to the execution of the REWRITE statement. In such cases, the REWRITE statement replaces the record according to the contents of the data item specified by RECORD KEY.
7. The invalid key condition occurs in one of the following situations:
   a) in sequential access mode: if the primary key value of the record to be rewritten does not match the primary key value of the last record read from the file;
   b) in dynamic or random access mode: if the primary key value of the record to be rewritten does not match the primary key value of any record in the file;
   c) if the alternate record key value of the record to be rewritten is the same as the alternate record key value of a record which already exists in the file and duplicate keys are not permitted.
8. The record rewritten by a successful REWRITE statement is no longer accessible in the record area; an exception to this rule is the use of the SAME RECORD AREA clause. In this case, the record is available to all other files specified in the SAME RECORD AREA clause as well as to the current file.
9. The file position indicator is not affected by the execution of a REWRITE statement.
10. Execution of a REWRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated (see also "FILE STATUS clause", page 491).

11. The number of character positions in the record referenced by record-name must not be greater than the largest number of character positions or less than the smallest number of character positions permitted by the associated RECORD IS VARYING clause. Otherwise, the REWRITE statement will be unsuccessful, the update operation will not take place, the content of the record area remains unchanged and the I-O status of the file associated with record-name is set to a value indicating a record length conflict (see "I-O status", page 480).
12. For a record with an alternate key, the REWRITE statement is executed as follows:
    a) If the value of a specific alternate key is not changed, the order in which records are retrieved remains unchanged when that key is the key of reference.
    b) If the value of a specific alternate key is changed, the order in which records are retrieved may be changed when that key is the key of reference. If duplicates are permitted, the record becomes the last of the set of records which have the same alternate key value as the record to be rewritten.
13. END-REWRITE delimits the scope of the REWRITE statement.

START statement

Function

The START statement defines the logical starting point within a file for subsequent sequential read operations.

Format

```
START file-name [WITH NO LOCK]  [ KEY { IS EQUAL TO                          } file-name ]
                                      { 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 >                             }

      [INVALID KEY imperative-statement-1]

      [NOT INVALID KEY imperative-statement-2]

      [END-START]
```

The format extension WITH NO LOCK is effective during shared updating of files; it is described in the "COBOL85 User Guide" [1].

Syntax rules

1. Relational operators are mandatory separators. To avoid possible misinterpretations, they have not been underlined in the above format.
2. file-name must refer to a file in sequential or dynamic access mode.
3. data-name may be qualified.
4. The INVALID KEY phrase must be present if no applicable USE procedure has been specified for that file.
5. data-name must be the name of the RECORD KEY or ALTERNATE RECORD KEY data item declared for that file or an alphanumeric data item that is subordinate to the RECORD KEY or ALTERNATE RECORD KEY item. In this case, the first character of both items must be identical.

General rules

1. The type of comparison is specified by the relational operator in the KEY phrase. The logical key of each record in the file specified by data-name is compared with the contents of the data item referenced by data-name (RECORD KEY or ALTERNATE RECORD KEY). If the length of data-name differs from that of the RECORD KEY or ALTERNATE KEY item, the comparison is performed as though the larger data item had been truncated from the right to the length of the smaller item. All remaining rules governing the comparison of nonnumeric operands (of equal size) then apply, with any specified PROGRAM COLLATING SEQUENCE being ignored. All key comparisons thus take place on the basis of the EBCDIC collating sequence, i.e. as though no PROGRAM COLLATING SEQUENCE or PROGRAM COLLATING SEQUENCE IS NATIVE had been specified.

   a) For the relational operators EQUAL, GREATER, GREATER OR EQUAL, NOT LESS and their equivalents, the current record pointer is positioned to the first record currently existing in the file whose key satisfies the comparison.

   b) For the relational operators LESS, LESS OR EQUAL, NOT GREATER and their equivalents, the record pointer is positioned to the last record whose key satisfies the comparison.

   c) If the relation condition is not satisfied for any record in the file, an invalid key condition occurs, and the START statement is terminated unsuccessfully (see "Invalid key condition", page 506).

2. If the KEY phrase is omitted from the START statement, the relational operator EQUAL is assumed.

3. After execution of a START statement, the contents of the FILE STATUS data items (if specified) of that file are updated (see FILE STATUS clause, page 491).

4. The file specified by file-name must be open in the INPUT or I-O mode at the time the START statement is executed (see OPEN statement, page 511).

5. Execution of the START statement alters neither the content of the record area of the file nor the content of any data item referenced in a DEPENDING ON phrase of the RECORD clause associated with the file.

6. If an alternate key is specified in the KEY phrase of the START statement, that key becomes the key of reference for subsequent READ statements (format 1).

7. The END-START phrase delimits the scope of the START statement.

USE statement

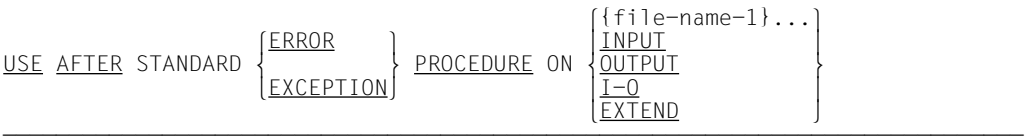
Function

The USE statement introduces declarative procedures and defines the conditions for their execution. However, the USE statement itself is not an executable statement.

Format 1 see "Sequential file organization" (page 411).

Format 2 introduces procedures to be executed if an input/output error occurs in a file.

Format 2



Syntax rules and general rules

See the USE statement for relative file organization, page 473.

Example 6-1

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. USEIND.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     TERMINAL IS T.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT MASTER-FILE ASSIGN "IN-OUT"
            ORGANIZATION IS INDEXED
            RECORD KEY ISAMKEY
            ACCESS MODE IS DYNAMIC
            FILE STATUS INDICATOR.
 DATA DIVISION.
 FILE SECTION.
 FD  FILE1.
 01  RECORD-FORMAT.
     03 ISAMKEY                PIC 9(8).
     03 CONTENTS               PIC X(72).
 WORKING-STORAGE SECTION.
 01  INDICATOR                 PIC 99.
 01  CLOSE-INDICATOR           PIC X VALUE "0".
   88  FILE-CLOSED           VALUE "0".
   88  FILE-OPEN             VALUE "1".
 PROCEDURE DIVISION.
 DECLARATIVES.
 FILE-ERROR SECTION.
     USE AFTER ERROR PROCEDURE ON FILE1.
 STATUS-QUERY.
     EVALUATE INDICATOR
     WHEN    10
       DISPLAY "End of FILE1 encountered" UPON T
     WHEN    21
       DISPLAY "Record with key" ISAMKEY
               "ALREADY EXISTS OR NOT IN"
               "ASCENDING ORDER" UPON T
     WHEN    22
       DISPLAY "Record with key" ISAMKEY
               "ALREADY EXISTS" UPON T
     WHEN    23
       DISPLAY "Record with key" ISAMKEY
               "NON-EXISTENT" UPON T
     WHEN OTHER
       DISPLAY "Unrecoverable error"
               "(" INDICATOR") FOR MASTER FILE" UPON T
       IF FILE-OPEN
       THEN
         CLOSE FILE1
       END-IF
       DISPLAY "Program terminated abnormally" UPON T
       STOP RUN
     END-EVALUATE.
END DECLARATIVES.
BEGIN SECTION.
OPEN-CLOSE.
    . . .
     STOP RUN.
```

WRITE statement

Function

The WRITE statement causes a record to be released to an output or input/output file.

Format 1 see "Sequential file organization" (page 420).

Format 2 of the WRITE statement must be used for disk storage files that are not sequentially organized.

Format 2

```
WRITE record-name [FROM identifier-1]
     [INVALID KEY   imperative-statement-1]
     [NOT INVALID KEY imperative-statement-2]
     [END-WRITE]
```

Syntax rules

1. record-name and identifier-1 must not reference the same storage area.
2. record-name must be associated with a file description (FD) entry in the Data Division of the program and may be qualified.
3. The INVALID KEY phrase must be specified for a file unless an applicable USE procedure was declared.

General rules

1. The file whose record is referenced by the WRITE statement must be open in the OUTPUT, I-O or EXTEND mode.
2. When a WRITE statement is executed, the position within the file of the record to be output is located by means of the contents of the RECORD KEY data item.
3. Before the WRITE statement is executed, the content of the associated key field must be set accordingly (see RECORD KEY clause, page 493).
4. The record released by a WRITE statement is no longer available in the record area, unless the file associated with the record was specified in a SAME RECORD AREA clause, or the execution of the WRITE statement was abnormally terminated as unsuccessful because of the occurrence of an invalid key condition. The record is also available to those files which were referenced in a SAME RECORD AREA clause together with the specified file.

5. Execution of a WRITE statement with the FROM phrase is equivalent to execution of the following statements:

   ```
   MOVE identifier TO record-name
   WRITE record-name
   ```

   Data is transferred according to the rules for a MOVE statement without the CORRESPONDING phrase.

   The contents of the record area before execution of the implicit MOVE statement has no effect on the execution of the WRITE statement.

   After the WRITE statement is successfully executed, the information is still available in the area referred to by the identifier; however, as pointed out in general rule 4, this is not necessarily true for the record area.

6. Execution of the WRITE statement causes the contents of the data item that was specified in the FILE STATUS clause of the related file description entry to be updated.

7. If, during the execution of a WRITE statement with the NOT INVALID KEY phrase, the invalid key condition does not occur, control is transferred to imperative-statement-2 as follows:

   a) If the execution of the WRITE statement is successful: after the record is written and after updating the I/O status of the file from which the record originates.

   b) If the execution of the WRITE statement is unsuccessful: after the I/O status of the file has been updated and after executing the USE procedure that was specified for the file from which the record originates.

8. If a file is opened in output mode (OPEN statement with OUTPUT phrase), the following should be noted:

   a) In sequential access mode, the WRITE statement releases a record for the creation of a new file. In this context, the records must be transferred in ascending order of RECORD KEY values. Before execution of the WRITE statement, the RECORD KEY data item must be set to the required value.

   b) In random or dynamic access modes (which are equivalent for OUTPUT), records may be released to the File Control Processor in any program-specified order.

9. The invalid key condition is caused by the events listed in Table 6-2.

Access mode	OPEN mode and action taken	Cause of invalid key condition
SEQUENTIAL	OUTPUT / EXTEND A record is added to a file to be newly created. ("load mode").	Either the value of the primary key is not greater than that of the preceding record (the primary record keys must be sorted in ascending alphanumeric order), or no more space is available in the file for writing the record.
RANDOM or DYNAMIC	OUTPUT/ I-O / EXTEND A record is added to an existing file	The record has the same primary key value as a record already existing in the file, or no more space is available on the file for writing the record, or an alternate key value for which duplicates are not permitted already exists in a record in the file.

Table 6-2: WRITE statement - Causes of invalid key conditions

10. Occurrence of an invalid key condition indicates that the WRITE statement was unsuccessful; the contents of the record area are still available, and any existing FILE STATUS data item in that file is set to a value indicating the cause of the invalid key condition. The program resumes execution in accordance with the rules for the invalid key condition.
11. The number of character positions in the record indicated by record-name cannot be greater than the largest number of character positions or less than the smallest number of character positions permitted by the associated RECORD IS VARYING clause. Otherwise, the REWRITE statement will be unsuccessful, the update operation will not take place, the content of the record area remains unchanged and the I-O status of the file associated with record-name is set to a value indicating a record length conflict (see "I-O status", page 480).
12. If the file was opened in extend mode, the first record passed to DMS must have a primary key whose value is higher than the highest existing primary key value in the file.
13. If ALTERNATE RECORD KEY WITH DUPLICATES is specified, the alternate key value of the record does not need to be unique.
14. END-WRITE delimits the scope of the WRITE statement.

7 Inter-program communication

The inter-program communication module of the COBOL85 system permits communication between the programs of a run unit.

Inter-program communication is understood to comprise the following:

– Transfer of control from one program to another, with the facility to pass between the individual programs parameters which allow data from the calling program to be made available to a called program.

– The use of common data and files by the different programs of a run unit.

7.1 Concepts

Separately compiled program

A complete COBOL program that was compiled in a separate compiler run is referred to as a separately compiled program. It can be either an individual program or the outermost containing program of a nested program.

Nested program

A COBOL program that comprises a number of complete programs nested within one another is referred to as a "nested program".
A program that contains further programs is referred to as a "containing program".
A program that is contained in another program is referred to as a "contained program".

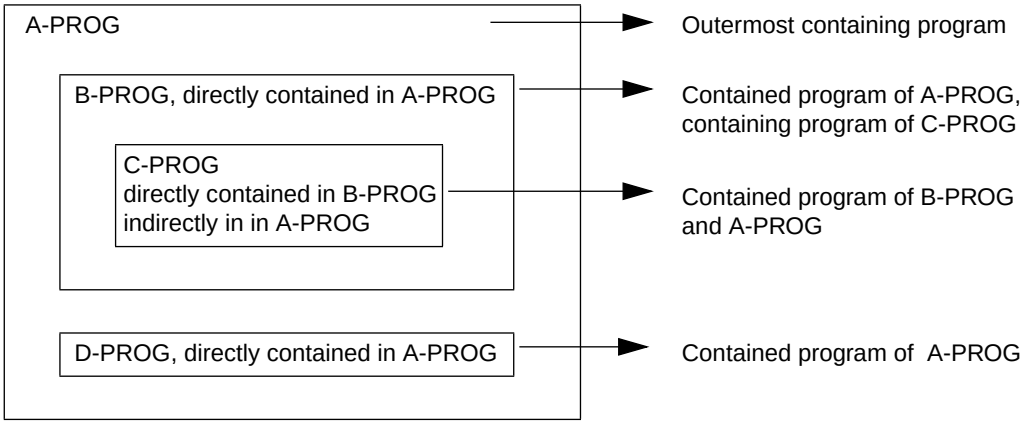
The programs of a nested program can be both containing programs and contained programs, apart from the "outermost" containing program that is treated in exactly the same manner as a separately compiled program within the run unit.

A contained program can be contained directly or indirectly in another program.

With respect to the immediately superordinate nesting level, a contained program is *directly* contained; with respect to other superordinate nesting levels it is *indirectly* contained.

Example 7-1

of the structure of a nested program



"Sibling program"

The programs comprising a nested program that are contained on the same level of nesting in a program are referred to as "sibling programs".

"Descendant"

Each program contained directly or indirectly in a "sibling program" is referred to as a "descendant" of this "sibling program".

Run unit

A run unit is a particular number of executable programs that act as a logical unit at run time.

A run unit may consist of

– one or more separately compiled programs,
– one or more nested programs,
– a combination of separately compiled programs and nested programs.

The program started on the system level is referred to as "main program" and all other programs of the run unit are known as "subprograms".

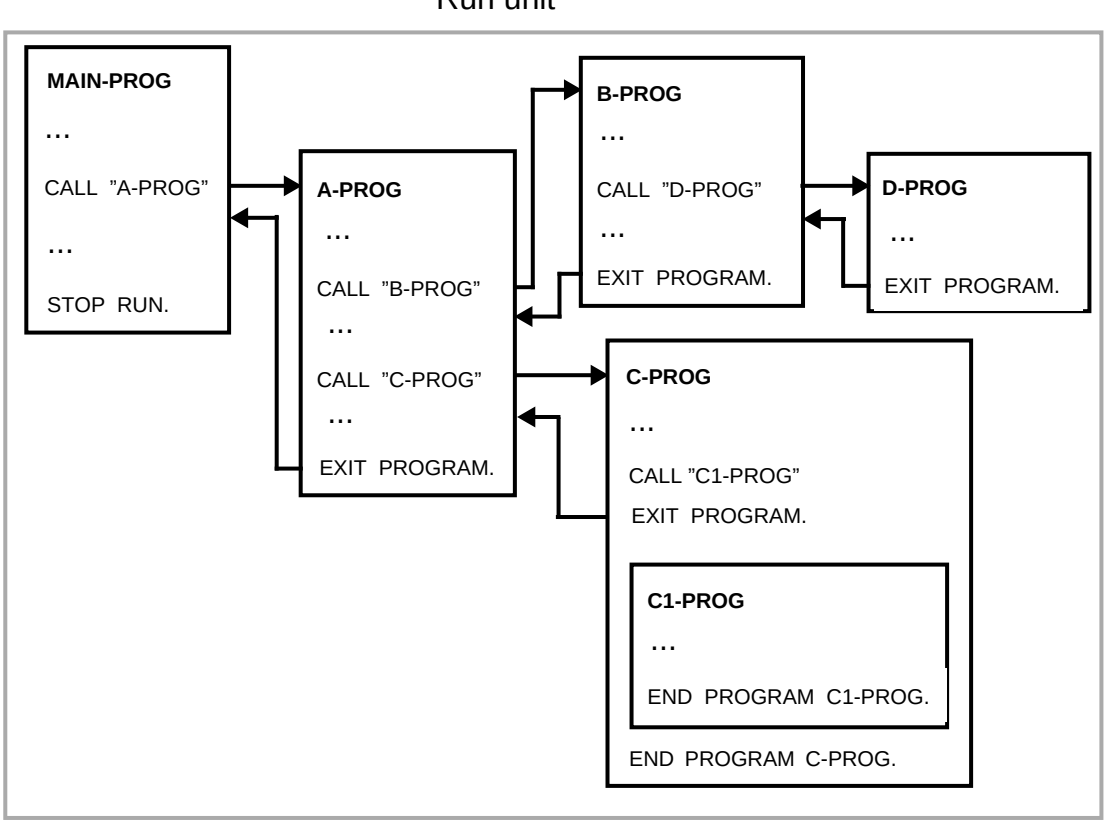
7.2 Control of inter-program communication

7.2.1 Runtime control

Control of a run unit begins with the program that is called on the system level. Each further program of the run unit is called by means of the CALL statement.
The CALL statement transfers program control to the called program. From there, control is subsequently returned to the calling program by means of the EXIT PROGRAM statement. The program is continued at the statement following the CALL statement.

The following example illustrates the logical structure of a run unit comprising five separately compiled programs, including one nested program:

Example 7-2



7.2.2 Rules for program names

A program is called by way of the name of the program declared in the PROGRAM-ID paragraph of the Identification Division.

The following basic rules apply to program names:

1. A program name may be referenced only by the CALL statement, the CANCEL statement and the end program header.
2. All separately compiled programs of a run unit must have different program names.
3. All programs of a nested program must have different names.
4. The contained programs of a nested program are "invisible" to all the separately compiled programs of a run unit; i.e. a separately compiled program cannot call any contained program of another separately compiled program.
5. The contained programs of a nested program may have the same name as the separately compiled programs of the run unit. The procedure provided for determining the valid program name in this instance is discussed in the following section ("Selecting the valid program name").
6. Within a nested program, one program can normally only call another program that is contained directly in it.
7. The call facilities in a nested program can be expanded with respect to the normal case when the COMMON attribute is applied to a contained program by means of the COMMON clause in the PROGRAM-ID paragraph.
   A program provided with the COMMON attribute can be called not only by the directly superordinate program but also by any "sibling program" and its "descendants" (see "COMMON clause", page 539).

Selecting the valid program name

When a program is called in a nested program, the relevant valid program name is then selected from the sum total of all the program names present in the run unit in accordance with the following rules of precedence:

1. The call applies to the name of a program that is directly contained in the calling program.
2. If 1) does not apply, the call applies to a COMMON program, i.e. to a program that can be called by its "sibling programs" and their "descendants".
3. If neither 1) nor 2) applies, the call applies to a separately compiled program of the run unit.

Example 7-3

```
PROGRAM-ID. A-PROG.
...
    CALL "B-PROG".                    B-PROG in A-PROG              (rule 1)
    CALL "C-PROG".                    Separately compiled C-PROG    (rule 3)
    CALL "D-PROG".                    D-PROG in A-PROG              (rule 1)
   PROGRAM-ID. B-PROG COMMON.
   ...
       CALL "D-PROG".                 Separately compiled D-PROG    (rule 3)
       CALL "C-PROG".                 C-PROG in B-PROG              (rule 1)
      PROGRAM-ID. C-PROG.
      ...
          CALL "B-PROG".              Separately compiled B-PROG    (rule 3)
      END PROGRAM C-PROG.
   END PROGRAM B-PROG.

   PROGRAM-ID. D-PROG.
   ...
       CALL "B-PROG".                 B-PROG in A-PROG              (rule 2)
       CALL "C-PROG".                 Separately compiled C-PROG    (rule 3)
   END PROGRAM D-PROG.
END PROGRAM A-PROG.
```

This example illustrates selection of the relevant valid program name.

A CALL statement for a program "A" within this nested program would be inadmissible since this call would apply to a further separately compiled program called "A" that may not be present in the run unit.

7.2.3 Initial state

The INITIAL attribute is created by specifying the INITIAL clause in the PROGRAM-ID paragraph of the program (see "INITIAL clause", page 539).

Initial state means:

– The program's internal data defined in the WORKING-STORAGE SECTION is initialized. If the VALUE clause is specified for a data item, the item is set to the defined value. If no VALUE clause is specified, the initial value of the data item is undefined.

– Files (contained files) belonging to the program are closed.

– The control mechanisms for all the PERFORM statements contained in the program are set to their initial state.

– A GO TO statement which refers to an ALTER statement in the same program is set to its initial state.

A program is in the initial state:

1. when it is called for the first time in a run unit,
2. when it is called for the first time since either it or a program in which it is directly or indirectly contained was the object of a CANCEL statement,
3. whenever it is called if it has the INITIAL attribute,
4. when it is contained directly or indirectly in a program that has the INITIAL attribute and when it called for the first time after this program has been called.

Example 7-4

```
PROGRAM-ID. A-PROG.
...
    CALL "B-PROG".                    Statement 1
    CALL "B-PROG".                    Statement 2
...
  PROGRAM-ID. B-PROG INITIAL.
  ...
      CALL "C-PROG".                  Statement 3
      CALL "D-PROG".                  Statement 4
  ...
    PROGRAM-ID. C-PROG COMMON.
    ...
    END PROGRAM C-PROG.

    PROGRAM-ID. D-PROG.
    ...
        CALL "C-PROG".                Statement 5
        CANCEL "C-PROG".
        CALL "C-PROG".                Statement 6
    ...
    END PROGRAM D-PROG.
  END PROGRAM B-PROG.
END PROGRAM A-PROG.
```

Statement 1: Program B-PROG is in the initial state (rules 1 and 3).

Statement 2: Program B-PROG is in the initial state (rule 3).

Statement 3: Program C-PROG is in the initial state (rules 1 and 4).

Statement 4: Program D-PROG is in the initial state (rules 1 and 4)

Statement 5: Program C-PROG is not in the initial state.

Statement 6: Program C-PROG is in the initial state (rule 2).

7.3 Using common data

7.3.1 External and internal data

Data items and files defined in a program may be internal or external data.

Internal data can only be accessed within the program in which this data is defined. External data can be accessed by any program in the run unit.

Internal data occupies a storage area associated exclusively with the particular program in which the data was defined. External data, on the other hand, occupies a storage area associated with the run unit. Data defined as external in the programs of a run unit can be accessed by all programs of the run unit.

External files and data items are created through specification of the EXTERNAL clause in the FILE SECTION or WORKING-STORAGE SECTION (see "EXTERNAL clause", page 544).

Example 7-5

A run unit comprises three programs in which certain data is defined as external. The resulting storage occupancy is illustrated by the following diagram:

Storage area Program A
Storage area Program B
Storage area Program C
Storage area External data

All internal data, i.e. data not defined as being external, is contained only in the storage area of the program in which it is defined. All data defined as external, on the other hand, is contained only in the storage area for external data.

7.3.2 Local and global names

The names used in *nested* programs may be classified as "local" and "global" names.

Local names

Local names are valid only within the program in which the associated data is described. Most of the names that are local as standard can be declared as global. The following names are always local:

– paragraph-names
– section-names

Global names

Global names are valid not only in the program in which they are defined but also in that program's contained programs.
The following name types are always global:

– Names defined in the CONFIGURATION SECTION:

  computer-name
  alphabet-names
  class-names
  condition-names (SPECIAL NAMES paragraph)
  mnemonic-names
  symbolic characters
  CURRENCY SIGN character
  DECIMAL-POINT

– COBOL special registers:

  TALLY
  PRINT-SWITCH
  SORT registers
  RETURN-CODE

The following name types are local as standard and can be explicitly defined as global in nested programs:

– condition-names
– data-names
– file-names
– index-names
– record-names
– report-names

Definition of a name as global is performed using the GLOBAL clause (see "GLOBAL clause", page 547).

Determining the valid name

When a name is referenced, the valid name is selected from the sum total of all the names defined in the nested program, in accordance with the following rules of precedence:

1) The referenced name is defined in the same program.

2) If 1) does not apply, the referenced name is defined as a *global* name in the *directly* superordinate (next outer) program.

3) If neither 1) nor 2) applies, the referenced name is defined as a *global* name in the *indirectly* superordinate program.
Condition 3) is checked until the data description entry of the referenced name is found in one of the other indirectly superordinate programs, possibly in the outermost containing program.

Example 7-6

```
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 program B-PROG (rule 1)
      MOVE  "B"  TO  B1.          B1 in program B-PROG (rule 1)
      MOVE  "C"  TO  C1.          C1 in program A-PROG (rule 2)
  ...
    PROGRAM-ID. C-PROG.
    ...
        MOVE  "X"  TO  A1.        A1 in program B-PROG (rule 2)
        MOVE  "Y"  TO  B1.        B1 in program A-PROG (rule 3)
        MOVE  "Z"  TO  C1.        C1 in program A-PROG (rule 3)
    ...
    END PROGRAM C-PROG.
  END PROGRAM B-PROG.
END PROGRAM A-PROG.
```

7.4 Language elements for inter-program communication

7.4.1 Overview

The language elements relevant to inter-program communication are summarized in the following table:

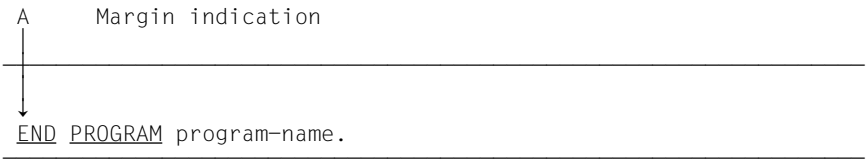
Language element	Function
END PROGRAM entry	Denotes the end of the named source program
INITIAL clause	The program is set to its initial state each time it is called
COMMON clause	The program can also be called by its sibling programs and their descendants
LINKAGE SECTION	In the called program: to define data passed from the calling program
EXTERNAL clause	Declaration of files and data items as external
GLOBAL clause	Declaration of names as global
PROCEDURE DIVISION	In the called program: definition of the standard entry point
USING phrase	List of data-names; indicates that data is transferred from the calling program
CALL statement	In the calling program: call for a subprogram or a contained program
USING phrase	List of data-names; indicates that data is transferred to the called program
CANCEL statement	The program is set to its initial state the next time it is called
ENTRY statement	Only in the subprogram of a run unit: definition of a non-standard entry point
USING phrase	List of data-names (defined in the LINKAGE SECTION); indicates that data is transferred from the calling program
EXIT PROGRAM statement	Return of control to the calling program
GOBACK statement	Identifies the logical end of a program
USE statement with GLOBAL attribute	In nested programs: definition of global USE procedures

7.4.2 End program header

Function

The end program header indicates the end of the named COBOL source program.

Format



Syntax rules

1. program-name must conform to the rules for formation of a user-defined word.
2. program-name must match the program name that was defined in the PROGRAM-ID paragraph of the program to be terminated (see PROGRAM-ID paragraph).

General rules

1. The end program header denotes the end of the COBOL source program specified in the entry.
2. Each program of a nested program must be terminated by means of an end program header.
3. The program line following the end program header in a contained program is either the Identification Division header of another contained program or another end program header.
4. If an end program header terminates an individual program, then either no further program line follows or the Identification Division header for the next individual program follows (sequence of programs).

7.4.3 Language elements of the Identification Division

PROGRAM-ID paragraph

Function

The PROGRAM-ID paragraph specifies the name by which a program is identified. Appropriate attributes can be assigned to the program by means of the INITIAL and COMMON clauses.

Format

```
PROGRAM-ID. program-name [IS { INITIAL          } PROGRAM].
                             { COMMON           }
                             { INITIAL COMMON   }
                             { COMMON INITIAL   }
```

Syntax rules

1. program-name must conform to the rules for formation of a user-defined word.
2. The programs of a nested program must have different names.
3. The COMMON clause may only be specified for the contained programs of a nested program. It must not be specified in the outermost containing program.

General rules

1. The effect of the INITIAL clause is that the program is set to its initial state each time it is called (see "Initial state", page 532).
2. The COMMON clause causes the program to have the COMMON attribute. Such a program can be called not only by the directly superordinate program but also by its "sibling programs" and their "descendants".

Example 7-7

of the effect of the INITIAL clause

Calling program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
...
DATA DIVISION.
WORKING-STORAGE SECTION.
...
PROCEDURE DIVISION.
...
    CALL "UPROG1" USING...  ————————————————————— (1)
...
...
    CALL "UPROG1" USING...  ————————————————————— (2)
```

Called program:

```
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.
```

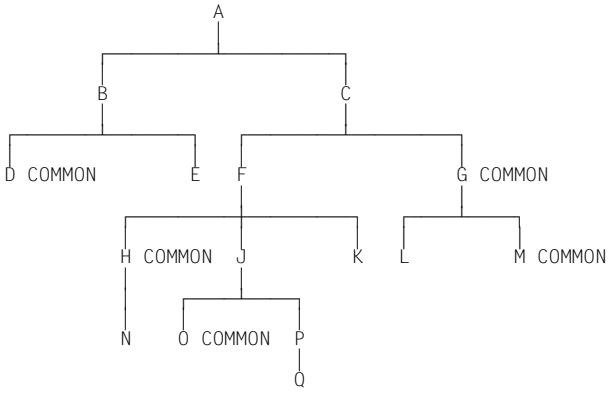
(1) UPROG1 is called for the first time by MAIN.

(2) UPROG1 is called for the second time by MAIN. UPROG1 is again in its initial state: the content of USER-DATE, for example, is again SPACE, even if it was changed during the first call of UPROG1.

Example 7-8

of the effect of the COMMON clause

Structure of a nested program A:



Valid call options within the nested program shown above:

Called program	Calling program															
	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 = permitted call

7.4.4 Language elements of the Data Division

LINKAGE SECTION

Function

The LINKAGE SECTION is located in a called program. It describes data items in the calling program, which may reference the called program.

Format

```
 A       Margin indication
 |
 |
 |
 |
 LINKAGE SECTION.
[77-level-description-entry ]
[                           ] ...
[record-description-entry   ]
```

Syntax rules

1. The LINKAGE SECTION is meaningful only in program which is referenced by a CALL statement containing the USING phrase. The Procedure Division header must also contain a USING phrase.
2. Data items that bear no hierarchical relationship to one another are are defined as independent elementary items in conjunction with the level number 77 or 01.

   The following specifications are required in each data description entry:

   – level-number 77 or 01

   – data-name

   – the PICTURE clause or the USAGE clause with the INDEX, COMPUTATIONAL-1, or COMPUTATIONAL-2 phrases.
3. Elementary items in the LINKAGE SECTION which bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of record descriptions (see chapter 3, page 141ff).
4. The VALUE clause must not be specified in the LINKAGE SECTION except in level 88 condition-name entries (format 2 of the VALUE clause).

General rules

1. If a data item in the LINKAGE SECTION is accessed in a program which is not a called program, the effect is undefined.
2. In addition to the mandatory clauses (see syntax rule 2), additional data clauses can be used to complete the data description entry. Their use is described in chapter 3 (page 141ff).

EXTERNAL clause in file and data description entries

Function

With the EXTERNAL clause, a file or a record can be defined as external. External files and records can be accessed by any program in which the file or record is described.

Format for file description entries

```
FD file-name IS EXTERNAL
further file description clauses
```

Syntax rules

1. The format shows only specification of the EXTERNAL clause. This is the same for all types of file organization. The further file description clauses which differ according to the file organization are described in chapters 4 through 6.
2. If a file is defined as external, the records in this file are by implication also external.

General rules

1. If the file description entry for a sequential file contains the LINAGE clause and the EXTERNAL clause, the LINAGE-COUNTER special register is implicitly an external data item.
2. The first seven characters of the names of external files must be unique and must not be the same as:
   - external record-names from the WORKING-STORAGE SECTION
   - PROGRAM-ID names of the run unit, except for program names of contained programs of a nested program
   - names used as entry points in the ENTRY statement
   - names that identify interfaces (LZS-name, TCBENTRY-name, etc.).
3. The effect of the FILE STATUS clause for external files is always local to the program, i.e. the file status is supplied only by I-O operations in the program that contains a corresponding specification in the file description entry.
4. The EXTERNAL clause must not be specified in file or record description entries for files that use a common I-O area (SAME RECORD AREA clause).

5. The EXTERNAL clause must not be specified for files that are assigned to the system devices SYSIPT, SYSOPT, PRINTER or PRINTERnn.
6. The EXTERNAL clause must not be specified for files for which user labels and corresponding USE procedures are defined.
7. An external file must be described in largely identical manner by means of explicit clauses or implicit default values in all programs that wish to access the file. The following table shows how and to what extent the descriptions must match:

Clauses / specifications	In all programs
Name of external file	same to full length (30 characters)
OPTIONAL phrase (SELECT clause)	same specification*)
ASSIGN TO data-name	same form of assignment
ASSIGN TO PRINTER literal	same form of assignment
ORGANIZATION clause	same form of organization
ACCESS MODE clause	same access method
RELATIVE KEY phrase	same number of digits
RECORD KEY clause	same length and position
ALTERNATE RECORD KEY clause	same number, position, length and DUPLICATES phrase
BLOCK CONTAINS clause	same block size in bytes
MULTIPLE FILE TAPE clause	same position number
RECORD clause	same minimum and maximum record length
LABEL RECORDS clause	same specification*)
REPORT clause (Report Writer)	same specification*)
LINAGE clause	same specification*)
CODE SET clause	same specification*)
RECORDING MODE clause	same specification*)

*) same specification means: Either the relevant clause may be specified in none of the programs or it must be specified the same in all programs.

All programs that access the same external file must have been compiled with the same value of the compiler option ENABLE-UFS-ACCESS (see COBOL85 User Guide [1]).

8. If a file is defined as external, this does not mean that the associated file-name is implicitly a global name.

Format for record description entries

```
01 data-name-1 IS EXTERNAL

further record description clauses
```

Syntax rules

1. The EXTERNAL clause may be included only in the record description entry of the WORKING-STORAGE SECTION.
2. A data-name declared as external may not be declared again within the same program.
3. The EXTERNAL clause may refer only to a data description entry with level number 01.
4. The first seven characters of the name of the external record must be unique, since only the first seven characters are used to identify identical external records from different programs.
5. The following must not be used as the name of an external record:
   - names of external files
   - PROGRAM-ID names of the run unit, except names of contained programs in a nested program
   - names used as entry points in an ENTRY statement
   - names which identify interfaces (LZS-name, TCBENTRY-name etc.).
6. The EXTERNAL clause and the REDEFINES clause must not be specified together in the same data description entry.
7. No VALUE clauses (with the exception of VALUE clauses for condition-names (level number 88)) may be specified in data description entries which are assigned to or subordinate to a data description entry containing the EXTERNAL clause.

General rules

1. An external record that is described in several programs of a run unit must have the same name in each of these programs and must also have the same length. The compiler permits the use of different length definitions, but this possibility should not be used if CALL identifier is used in the program.
2. If a record is defined as external, the associated file is not implicitly an external file.

GLOBAL clause

Function

The GLOBAL clause can only be used within a nested program. It defines a file-name, data-name or report-name as global. A global name can be accessed by the program defining it and by any other program contained directly or indirectly in this program.

Format for file description entry

```
FD file-name IS GLOBAL
further file description clauses
```

Format for data description entry

```
01 data-name IS GLOBAL
further data description clauses
```

Format for report description entry

```
RD report-name IS GLOBAL
further report description clauses
```

Syntax rules

1. The GLOBAL clause may only be specified in the following description entries:
   a) file description entries and report description entries
   b) data description entries with level number 01 in the FILE SECTION or WORKING-STORAGE SECTION
2. If two data items are defined in the same Data Division with the same name, the GLOBAL clause must not be specified in any of the corresponding data description entries.
3. The GLOBAL clause must not be specified in file or record description entries for files that use a common I-O area (SAME RECORD AREA clause).

General rules

1. A data-name, file-name or report-name whose description contains a GLOBAL clause is a global name. All data-names subordinate to a global name and all condition-names associated with a global name are global names.
2. Any program contained in the program that describes the global name may access a global name. The global name does not need to be described again in the program that references it. In the case of references to identical names, the local names have priority (see "Determining the valid name", page 536).
3. If a data description entry also contains the REDEFINES clause in addition to the GLOBAL clause, the redefined data item is not necessarily global by implication.
4. If a global data item contains a variable-length table, then the corresponding DEPENDING ON element in the same Data Division must also be described as global.
5. The data-names in the CONFIGURATION SECTION are always implicitly global. The CONFIGURATION SECTION should therefore only be specified in the outermost containing program.
6. The index of an indexed table assigned to a global data item is also global.
7. If both the LINAGE and GLOBAL clauses are specified in a description of a sequential file, then the LINAGE-COUNTER special register is also global.
8. If a report description entry contains the GLOBAL clause, then the LINE-COUNTER and PAGE-COUNTER special registers are also global.

7.4.5 Language elements of the Procedure Division

Procedure Division header

Function

In a called program (subroutine), the Procedure Division header determines the standard entry point. Optionally, data-names may be specified if data is transferred from the calling program as a parameter.

Format

```
PROCEDURE DIVISION [USING {data-name-1}...].
```

Syntax rules

1. The USING phrase may be written only if the object program is called by a CALL statement and the CALL statement in the calling program includes a USING phrase. The number of operands in the corresponding USING phrases must be identical; otherwise, the result is unpredictable.
2. Each data-name supplied in the USING phrase of the Procedure Division header must be defined in the LINKAGE SECTION of the program containing this header, and must have the level number 01 or 77.
   The data description of data-name-1 must not contain a REDEFINES clause.

General rules

1. The standard entry point within a called program is determined by the Procedure Division header (see also ENTRY statement for nonstandard entry points, page 564). In order to transfer control from a calling program to that entry point, the calling program must contain a CALL statement. The program-name supplied in this CALL statement must be the same as the program-name specified in the PROGRAM-ID paragraph of the Identification Division of the called program.
2. The USING phrase, when specified, has the effect that data-name-1 of the Procedure Division header in the called program and identifier-2 or identifier-3 in the USING phrase of the CALL statement in the calling program refer to the same set of data, which is equally available both to the called and the calling program. It is not necessary for the names to be identical. A data-name may appear only once in the Procedure Division header of the called program, whereas the same identifier may occur several times in the USING phrase of the CALL statement.

3. In the called program, the operands of the USING phrase are treated according to the data description supplied in the LINKAGE SECTION.
4. A program may run both as a called program and as a calling program at execution time. An exception to this is the first program (called for execution by the system); it **must not** contain a USING phrase in the Procedure Division header.

Example 7-9

```
Calling program:

IDENTIFICATION DIVISION
PROGRAM-ID. A-PROG.
...
WORKING-STORAGE SECTION.
01  ALPHA ...
01  BETA ...
77  GAMMA ...
...
PROCEDURE DIVISION.
...
    CALL "B-PROG" USING ALPHA BETA GAMMA.
...

Called program:

IDENTIFICATION DIVISION.
PROGRAM-ID. B-PROG.
...
LINKAGE SECTION.
01  DELTA ...
01  EPSILON ...
77  THETA ...
...
PROCEDURE DIVISION USING DELTA EPSILON THETA.
...
```

(1)

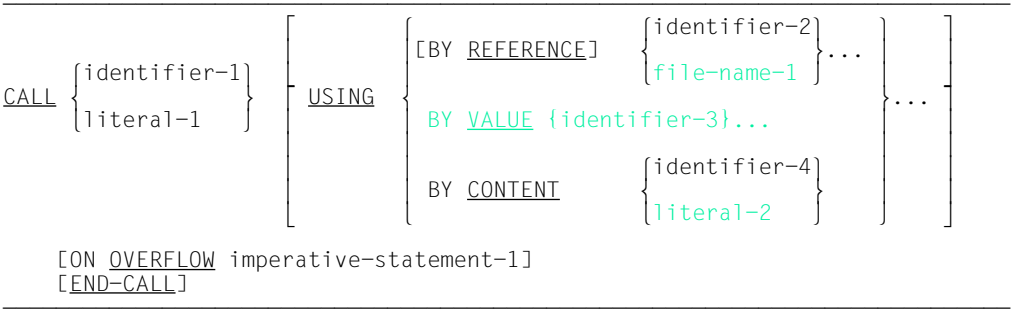
(1) The parameters of the USING phrases relate to one another in pairs, i.e. ALPHA and DELTA, BETA and EPSILON, GAMMA and THETA respectively each relate to the same data item.

CALL statement

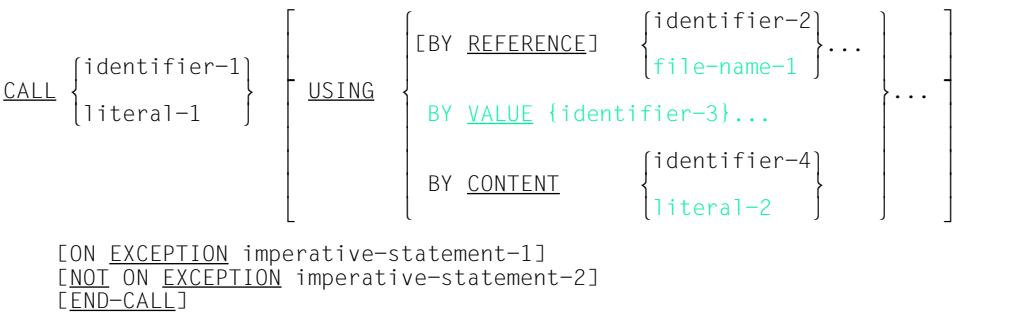
Function

The CALL statement passes control to a called program. Optionally, operands may be specified to enable the called program to access data of the calling program.

Format 1



Format 2



Syntax rules for formats 1 and 2

1. literal-1 must be a nonnumeric literal.
   If literal-1 is the program-name of an individual program or of the outermost containing program of a nested program, it must begin with an alphabetic character and may contain only uppercase letters and digits. The length of the name is dependent on the module format (see the “COBOL User Guide” [1]).

If literal-1 is the program-name of a contained program of a nested program, it must begin with an alphabetic character, may contain uppercase letters, lowercase letters and digits, and must not be longer than 30 characters.

2. identifier-1 must be defined as an alphanumeric data item so that its content can be a valid program name, as described under point 1.
3. The USING phrase in a CALL statement may be supplied only if a USING phrase has been written either after the associated Procedure Division header or in the ENTRY statement of the called program. Each USING phrase must have the same number of operands, otherwise the result will be unpredictable.
4. Every identifier-1 specified in the USING phrase must have been defined in the FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION or SUB-SCHEMA SECTION. It may have level number 01 or level number 77. However, the compiler allows every level number except 88. In order to align elementary items with USAGE INDEX, BINARY, COMPUTATIONAL, COMPUTATIONAL-5, COMPUTATIONAL-1, COMPUTATIONAL-2 in the LINKAGE SECTION, all 01-level items included are aligned on doubleword boundaries. Consequently, the user must ensure that these operands are aligned accordingly in the USING phrase.
5. file-name-1 is appropriate only if the called program is written in a language other than COBOL.
6. identifier-2 must not be a function-identifier.
7. identifier-3 must have been defined as a 2- or 4-byte data item with USAGE COMP-5 or as a 1-byte data item. If this is not the case, the result of the parameter transfer is undefined. This type of parameter transfer is not a good idea unless the called program was written in a language other than COBOL (see example 7-11).
8. literal-2 must be an alphanumeric literal or one of the figurative constants SPACE, LOW-VALUE or HIGH-VALUE.

General rules for formats 1 and 2

1. literal-1 or identifier-1 contains the name of the called program. The program which contains a CALL statement is the calling program. If the called program is a COBOL program, literal-1 or identifier-1 must contain the program-name specified in the PROGRAM-ID paragraph or in the ENTRY statement.
2. When the CALL statement is executed, control is passed to the called program. When control is returned to the calling program, imperative-statement-2 (if specified) is executed and the program then branches to the end of the CALL statement. If NOT ON EXCEPTION is omitted, the program branches immediately to the end of the CALL statement.

3. If the called program is not available during execution of the CALL statement, one of the following actions is executed:

   a) If ON OVERFLOW or ON EXCEPTION is specified, control is passed to imperative-statement-1. After completion of imperative-statement-1, control is passed to the end of the CALL statement.

   b) If ON OVERFLOW or ON EXCEPTION is not specified, an error message is issued and program execution is aborted.

4. If two programs in a run unit have the same name, then at least one of them must be a contained program within a nested program. A program having a multiply used program name can only be successfully called under the following conditions:

   a) The called program is directly contained in the calling program.

   b) The called program has the COMMON attribute and is called by the directly superordinate program or by one of the latter's sibling programs or their descendants.

   c) The calling program is a contained program within a nested program and calls a separately compiled program of a run unit.

5. The data to be passed as parameters from the calling program to the called program are specified in the USING phrase of the CALL statement with identifier-2... .
   The number and sequence of the parameters must match the specifications in the USING phrase of the Procedure Division header or the ENTRY statement. Excepted here are the indices assigned to tables (INDEXED BY phrase): The indices in a calling program and a called program always point to separate indices.

6. If file-name-1 is specified in the list of the USING phrase, the starting address of the system file control block of that file is supplied to the called program.

7. BY CONTENT and BY REFERENCE may be used together. The phrase BY CONTENT or BY REFERENCE applies to all subsequent parameters until another BY CONTENT or BY REFERENCE phrase is encountered. If neither BY CONTENT nor BY REFERENCE is specified, the default is BY REFERENCE.

8. If a parameter is passed BY REFERENCE, it occupies the same memory location in the calling program and the called program. The description of the item in the called program must specify the same number of characters as the description of the corresponding item in the calling program.

9. If a parameter is passed BY CONTENT, the calling program makes a copy of the parameter and passes this copy BY REFERENCE.
   The data description of the corresponding parameter in the called program must be chosen as follows:
   – if `identifier-4` is specified: the same as that of `identifier-4`
   – if figurative constants are specified: PIC X

– if an alphanumeric literal is specified: PIC X(n). Repetition factor n may not exceed the length of the literal. In contrast with the behavior for other transfers, the literal at the end may not be filled with blanks to the length of the corresponding parameter from the subprogram.

10. Specifying BY VALUE enables the direct, C-compliant transfer of values to C programs. If the parameter value “by value” is specified, this means that only the value of the parameter is passed to the called C program. The called program can access this value; it can also modify it, in which case the value remains unchanged in the COBOL program.
11. A called program may contain CALL statements, but must not execute any CALL statement that directly or indirectly calls the calling program via the standard entry point or an entry point generated by means of the ENTRY statement.
12. The END-CALL phrase delimits the scope of the CALL statement.

Format 3

```
CALL UPON SYSTEM USING { identifier-1 } [identifier-2]
                       { literal-1    }

    [STATUS identifier-3]

    [ON EXCEPTION imperative-statement-1]
    [NOT ON EXCEPTION imperative-statement-2]

    [END-CALL]
```

Syntax rules for format 3

1. identifier-1 and identifier-2 must be alphanumeric data items.
2. literal-1 must be an alphanumeric literal.
3. identifier-3 must be a numeric data item described with PIC S9(8) SYNC and USAGE COMP, USAGE COMP-5 or USAGE BINARY.
4. identifier-2 may not be a function identifier.

General rules for format 3

1. literal-1 or identifier-1 contains the BS2000 command to be executed.
   A slash (/) can be specified in front of the command. Lower-case letters are *not* converted into upper-case.
2. identifier-2 specifies an area for recording system output.
   This response area should be large enough to record the complete output of the specified BS2000 command.
   If identifier-2 is variable in length and contains the DEPENDING ON item, the maximum length of identifier-2 is used.
3. If identifier-2 is not specified or its current length is 0, the output is written to SYSOUT.
4. If an error occurs during execution of the command, the response area contains the system's error message text, provided it is large enough.
5. identifier-3 can be used to specify a status item in which specific values show the result of the execution of the command. The following values can occur:

00	Command executed successfully
04	The command was executed, but one or more records could not be entered because the response area is too small.
30	Error during command execution; no more information available
34	This command may not be specified in a program .
40	The current length of identifier-1 is invalid (<= 0 or > 32767 bytes).
41	The current length of identifier-2 is invalid (< 0).
90	There is not enough main memory available to execute the command; workaround: specify smaller areas (identifier-1 or identifier-2)

6. If an error occurs while the CALL statement is being executed, one of the following actions is performed:
   a) If ON EXCEPTION is specified, imperative-statement-1 is executed. When imperative-statement-1 has been completed, the program continues at the end of the CALL statement.
   b) If ON EXCEPTION is not executed, the program sequence is resumed without an error message.
7. If the CALL statement is executed successfully, one of the following actions is performed:
   a) If NOT ON EXCEPTION is specified, imperative-statement-2 is executed.
   b) If NOT ON EXCEPTION is not specified, a branch is made to the end of the CALL statement.

8. END-CALL limits the scope of validity of the CALL statement.

Note

The system output consists of variable records with length items and special characters for advance control, etc.; a record is usually represented by several lines on the screen. Only complete records are entered in the response area; the rest of the response area which is not used is deleted. For information on the structures and further processing of the response area, see User Guide "BS2000/OSD-BC DMS Macros".

Example 7-10

of the use of the CALL statement in the format CALL identifier-1

Main program:

```
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.
PMAIN.
     PERFORM UNTIL CALL-OPERAND = FUNCTION-1 OR FUNCTION-2
                                             OR FUNCTION-3
        DISPLAY "Please enter desired function"
                UPON T
        DISPLAY TABLE-FUNCTION UPON T
        ACCEPT CALL-OPERAND FROM T
     END-PERFORM
     PERFORM UNTIL RECORD-NUMBER NUMERIC
        DISPLAY "Please enter record number"
                "(numeric, 8 digits)" UPON T
        ACCEPT RECORD-NUMBER FROM T
     END-PERFORM
     CALL CALL-OPERAND USING RECORD-NUMBER
     END-CALL
     STOP RUN.
```

Subprogram "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.
```

Subprogram "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.
```

Subprogram "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.
```

Example 7-11

for the use of CALL ... USING BY VALUE

Main program:

```
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 9(8) SIGN IS LEADING SEPARATE.
PROCEDURE DIVISION.
1ST SECTION.
1.
    MOVE RETURN-CODE TO RTC.
    DISPLAY "RETURN-CODE = " RTC UPON T.
    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.
    STOP RUN.
```

Subprogram C1:

```
long C1(unsigned short c,
          unsigned long d)
{
long ret_val;
if (c && d)
   ret_val = 1;
else
   ret_val = -1;
return ret_val;
}
```

Subprogram D1:

```
long D1(signed short c, signed long d)
{
long ret_val;
if (c && d)
   ret_val = 1;
else
   ret_val = -1;
return ret_val;
}
```

Example 7-12 (for format 3)

File name assignment with the SET-FILE-LINK command in dialogs and status queries:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CMD.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS CRT.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  LINKFILE.                                                          (1)
  02  FILLER  PIC X(30) VALUE "SET-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 "PLEASE ENTER FILE NAME X(54)" UPON CRT.
    ACCEPT FILNAM FROM CRT.
    CALL UPON SYSTEM USING LINKFILE
                           STATUS RTC
      ON EXCEPTION
         MOVE RTC TO RTC-ED
         DISPLAY "ERROR DURING COMMAND CALL, STATUS= " RTC-ED UPON CRT
    END-CALL
    CALL UPON SYSTEM USING TFT-CMD
                           RESP
    END-CALL.
    DISPLAY "RESPONSE AREA RESP" RESP UPON CRT.
FIN.
    STOP RUN.
```

(1) Description of identifier-1 with substructures for BS2000 system command and file name.

(2) Description of identifier-3 (status item STAT)

(3) Description of identifier-2 (response area RESP)

CANCEL statement

Function

The CANCEL statement causes the specified program to be set to its initial state the next time it is called.

Format

```
CANCEL {identifier-1}
       {literal-1   }...
```

Syntax rules

1. literal-1 must be nonnumeric and must be a valid program name. If literal-1 is the program-name of a separately compiled program or of the outermost containing program of a nested program, it must begin with an alphabetic character and may contain only uppercase letters and digits. The length of the name depends on the module format.
   If literal-1 is the program-name of a contained program of a nested program, it must begin with an alphabetic character, may contain uppercase letters, lowercase letters and digits, and must not be longer than 30 characters.
2. identifier-1 must be defined as an alphanumeric data item so that its value may be a valid program name, as described under point 1.

General rules

1. literal-1 or the content of identifier-1 specifies the program which is to be set to its initial state.
2. Successful execution of the CANCEL statement closes the files in the specified program. If the program is called again in the same run unit or in a nested program after successful execution of a CANCEL statement, this program is in its initial state.
3. The program name specified as literal-1 or contained in identifier-1 must be unique within the run unit or the nested program unless it is a program name that may be used multiply under certain conditions (see "CALL statement", general rule 4, page 553. The program name must not be the same as the first 7 characters of the program name in the PROGRAM-ID paragraph of the program that contains the CANCEL statement.
4. Called programs may contain CANCEL statements, but a called program may not execute any CANCEL statement that either directly or indirectly references the calling program.

5. The logical connection to a program which is initialized with a CANCEL statement is established again only if this program is subsequently called again with a CALL statement.
6. A CANCEL statement has no effect if one of the following situations exists:
   - The program concerned is already in the initial state because it has not yet been called in the active run unit or in the nested program.
   - The program concerned has already been initialized with a CANCEL statement.
   - The program concerned or (with nested programs) a superordinate program has the INITIAL attribute..

     In these cases, the program continues at the next executable statement after the CANCEL statement.
7. During execution of a CANCEL statement, an implicit CLOSE statement (without any optional phrases) is executed for each open internal file assigned to the program. Any USE procedures specified for these files are not executed.
8. A CANCEL statement may only reference such programs for which the call is permitted within the call hierarchy.
9. If an explicit or implicit CANCEL statement is executed, then all contained programs in the program referenced by the CANCEL statement are also cancelled.

Example 7-12

Main program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ERROR-CODE PIC 9.
   88  O-K VALUE 0.
  ...
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    PERFORM WITH TEST AFTER UNTIL O-K
      CALL "UPROG1" USING ERROR-CODE
      IF NOT O-K
      THEN
        CANCEL "UPROG1"
      END-IF
    END-PERFORM
    STOP RUN.
```

Subprogram:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPROG1.
ENVIRONMENT DIVISION.
DATA DIVISION.
   ...
LINKAGE SECTION.
01 ERROR-CODE  PIC 9.
    88  O-K  VALUE 0.
PROCEDURE DIVISION USING ERROR-CODE.
    ...
    IF INTERN-ERROR
    THEN
      CONTINUE
    ELSE
      SET O-K TO TRUE
    END-IF
    EXIT PROGRAM.
```

UPROG1 is called repeatedly until the value 0 is returned. As long as a value other than 0 is returned, UPROG1 is set to its initial state and the loop is continued.

ENTRY statement

Function

The ENTRY statement is used in a COBOL subprogram of a run unit for specifying the entry point by which the subprogram may be called (as opposed to the standard entry point supplied by the Procedure Division header). Optionally, data-names may be specified, if data is to be transferred from the calling program as a parameter.

Format

```
ENTRY literal [USING {data-name-1}... ].
```

Syntax rules

1. literal must be a nonnumeric literal.

   literal must be a valid program-name, i.e. it must begin with an alphabetic character, must not contain any characters other than letters and digits, and have a maximum length of 7 characters.

2. The program-name specified by the literal must be unique in the programs which are linked to form a run unit. Also, it must not be the same as the first 7 characters of the program-name entered in the PROGRAM-ID paragraph of the program containing this ENTRY statement.

3. The USING phrase may be written only if the associated CALL statement in the calling program also contains a USING phrase. The number of operands in the corresponding USING phrases must be identical; otherwise the result will be unpredictable.

4. Each data-name-1... supplied in the USING phrase of the ENTRY statement must be defined as a data item in the LINKAGE SECTION of the program containing this ENTRY statement; its level number must be either 01 or 77.

5. The ENTRY statement must not be used in the programs of a nested program.

General rules

1. The ENTRY statement determines the entry point in the called program. The name of the entry point is specified by the literal. A branch to this entry point is effected by a CALL statement in another program which references this entry point.

2. The USING phrase has the effect that, at execution time, data-name-1 in the ENTRY statement of the called program and identifier-1 in the USING phrase of the CALL statement in the calling program refer to the same set of data, which is equally available both to the called program and to the calling program. The names need not be the same.

   In the USING phrase of the ENTRY statement in the called program, a data-name may occur only once; in the USING phrase of the CALL statement, however, the same identifier-1 may be specified more than once.

3. In the called program, the operands of the USING phrase are treated according to the data descriptions given in the LINKAGE SECTION.

4. An ENTRY statement may be executed only once (upon entry to the program). The remainder of the program may contain no further ENTRY statements.

5. If a data item is passed in the CALL statement as a BY CONTENT parameter, the value of this data item is transferred to a memory area which has the characteristics specified for identifier-2 in the CALL statement before the CALL statement is executed. The data type and length of each parameter in the BY CONTENT phrase of the CALL statement must be the same as those of the corresponding parameter in the USING phrase in the Procedure Division header.

EXIT PROGRAM statement

Function

The EXIT PROGRAM statement marks the dynamic end of a called program.

Format

```
EXIT PROGRAM
```

Syntax rule

If an EXIT PROGRAM statement is one of several imperative-statements within a sentence, it must be the last statement in this sentence.

General rules

1. Execution of the EXIT PROGRAM statement in a called program causes a transfer of control back to the calling program. An EXIT PROGRAM statement in a program that was not called as a subprogram has no effect.
2. An EXIT PROGRAM statement in a program which does not have the INITIAL attribute causes execution to continue with the next executable statement after the CALL statement in the calling program. The program state of the calling program remains unchanged and is identical with the state which existed when the CALL statement was executed in the calling program. This does not apply to data which was passed to the called program with the CALL statement and which has been modified by this program.
3. The EXIT PROGRAM statement in a program with the INITIAL attribute has the same effect as a CANCEL statement for this program. In the called program, the control mechanisms for all PERFORM statements are initialized.

GOBACK statement

Function

The GOBACK statement identifies the logical end of a program.

Format

```
GOBACK
```

Syntax rule

The GOBACK statement may not be used in a global USE procedure.

General rules

1. The GOBACK statement has the same effect as the following sequence:
   EXIT PROGRAM
   STOP RUN
2. For GOBACK the same rules therefore apply as for EXIT PROGRAM and STOP RUN.

USE statement with GLOBAL phrase

The USE statement with the GLOBAL phrase can be specified in nested programs in order to define procedure declarations or label routines as global.

The effect of the USE statement in nested programs is illustrated in the following.

Format 1 for the declaration of user label routines cannot contain a GLOBAL clause.

Format 2 declares procedures to be executed if an I-O error occurs for a file.

```
USE GLOBAL AFTER STANDARD {EXCEPTION} PROCEDURE ON {{file-name-1}...}
                          {ERROR    }              {INPUT          }
                                                   {OUTPUT         }
                                                   {I-O            }
                                                   {EXTEND         }
```

Format 3 declares procedures to be executed by the Report Writer prior to report output.

```
USE GLOBAL BEFORE REPORTING identifier-1
```

Syntax rules

1. A USE procedure in format 3 is only permitted if the corresponding FD or RD entry contains the GLOBAL attribute and if the USE procedure is declared in the same program as the associated FD or RD entry.
2. A USE procedure in format 2 requires no FD entry with a GLOBAL clause.
3. For further syntax rules relating to the USE statement, refer to page 411 (formats 1 and 2), page 473 (format 2) and page 626 (format 3).

General rules

1. The GLOBAL phrase for a USE procedure has the same effect as the GLOBAL clause in data and file descriptions (see page 547).
   Any program contained in the program that declares the global USE procedure may access a global USE procedure.
2. If an I-O statement requires the use of a USE procedure, then the valid USE procedure is selected from the sum total of all the USE procedures declared in the nested program, in accordance with the following rules of precedence:
   a) The valid USE procedure is the one defined in the same program.

b) If a) does not apply, the valid USE procedure is the one declared as global in the *directly* superordinate (next outer) program.

c) If neither a) nor b) applies, the valid USE procedure is the one declared as global in the *indirectly* superordinate program.

Rule c) applies until all indirectly superordinate programs have been searched for the valid global USE procedure.

If no suitable USE procedure is found, then none is executed.

3. As an extension to ANS85, the compiler described here also permits PERFORM statements that relate to program segments outside of the DECLARATIVES. These program segments may also contain CALL, GO TO, GOBACK and EXIT PROGRAM statements. When a global USE procedure is used, this may result in recursive calling of the program that activated the USE procedure. A recursive call (always inadmissible) is not detected until program execution time, and results in program abortion.
A global USE procedure should not therefore execute any EXIT PROGRAM statement.

Example 7-13

of the scope of validity of global USE procedures

```
PROGRAM-ID. A-PROG.
...
FD  FILE1  GLOBAL.
...
  PROGRAM-ID. B-PROG.
  ...
      USE GLOBAL ... ON FILE1.              (1) Global USE procedure
  ...
    PROGRAM-ID. C-PROG.
    ...
        USE GLOBAL ... ON INPUT.            (2) Global USE procedure
        USE        ... ON OUTPUT.           (3) Local USE procedure
    ...
        OPEN OUTPUT FILE1.                  USE procedure (3)
        OPEN EXTEND FILE1.                  USE procedure (1)
    ...
      PROGRAM-ID. D-PROG.
      ...
          OPEN INPUT FILE1.                 USE procedure (2)
      ...
        PROGRAM-ID. E-PROG.
        ...
        FD  FILE1.
        ...
        OPEN INPUT FILE1.                   USE procedure (2)
        OPEN OUTPUT FILE1.                  USE procedure (1)
        ...
        END PROGRAM E-PROG.
      END PROGRAM D-PROG.
    END PROGRAM C-PROG.
  END PROGRAM B-PROG.

  PROGRAM-ID. F-PROG.
  ...
      OPEN I-O FILE1.                       Not a USE procedure
  END PROGRAM F-PROG.
END PROGRAM A-PROG.
```

8 Report Writer

8.1 General description

Special language elements are available for generating reports (report writer language features). These features allow the user to define the format and contents of a report in the Data Division in such detail that only a few Procedure Division statements are required to produce the report.

A printed report displays specific information in a structured format. The total number of all files and report description entries in a source program must not exceed 254.

The Data Division contains report description entries that

– name the reports to be generated by the Report Writer

– assign the reports to a file (report file) to which they are to be written, and

– describe the content and format of reports to be generated.

The statements by which the reports are generated are specified in the Procedure Division.

The Report Writer produces the desired reports according to the defined formats by

– transferring program data into the reports in the required format,

– accumulating data sums, which are added up to subtotals and final totals,

– updating line and page counters, and

– editing and printing the generated reports.

The Report Writer described in this manual is functionally equivalent to the optional Report Writer module described in ANS85, except for some minor differences in syntax.

8.1.1 General description of the Data Division

The output file to which the report is to be written must be declared in the Data Division by means of a file description (FD) entry, which also contains the names of the reports (see "REPORT clause", pagepage 575).

The REPORT SECTION, which defines the format and content of each report, must be specified as the last section of the Data Division. This section contains two types of entries:

1. **The report description** (RD).
   It must specify the name of the report. A page format (PAGE LIMIT clause), a character (CODE clause) preceding each line of the report (not to be printed, however), and a hierarchy of control data items (CONTROL clause) may also be supplied here.
2. **The report group description entry**.
   It describes, for instance, the print fields of a report group; that is, data items with line and column positioning into which data will be entered either directly (VALUE) or indirectly (sending field, sums).

The report description entry is used to specify the format of a page by defining the number of lines per page as well as the line numbers as vertical boundaries of the region within which each type of report group may be printed. These entries are used to control the positioning of the various report groups at the time the report is produced.

Detail report groups are structured by specifying **control data items** in their hierarchical order.

Detail report groups, which are functionally related to the hierarchical items, are combined into several groups in the order in which they are created, so that all detail report groups within a given group are assigned the same value of the hierarchically lowest item. Depending on whether a control heading and/or control footing is defined for this level of hierarchy, each group is introduced by the control heading and is terminated by the control footing (which generally contains information applicable only to the current group, e.g. subtotals).

Similarly, when another hierarchical item occurs, the above series of detail report groups is, in turn, combined into groups, now governed by the next highest item in the hierarchy, etc.

This structure is generated by the Report Writer as follows:

before creating a detail report group, the Report Writer tests the control data items in their hierarchical order, from top to bottom. As soon as it detects a change in value, i.e. a **control break**, all existing control footings are created in hierarchical order from bottom to top, up to the level where the first change in value was encountered. This is followed by all existing control headings on the same level in reversed hierarchical order, and finally, by the detail report group itself.

The specification of control data items in conjunction with the report groups "control heading" and "control footing" thus enables the user to have the report produced in a structured format.

The **report group description entry**, formally similar to a record description, is used to describe the properties of all data items in the report. Beyond the ordinary framework of COBOL language elements for the description of data items, a report data item is assigned line and column, i.e. a page position; in other words, the data item is declared to be a print field. Each of these fields receives information.

Three types of information may be supplied:

SOURCE information (SOURCE clause information), which is available through a data item defined outside the REPORT SECTION.

SUM information (SUM clause information), as a result of adding up data representing, in turn, SOURCE information and/or SUM information.

VALUE information (VALUE clause information), as a predefined information value.

Since a given report may contain several types of report groups, the report group description specifies the type of the report group. The seven types that may be used are shown in Table 8-1.

With its report and all associated report group description entries, the report is completely described as to its format and content (including the summation operations required); that is, all prerequisites to writing the report are met.

Type	Definition
REPORT HEADING	A report group that is written only once, as the first group of the report.
REPORT FOOTING	A report group that is written only once, as the last group of the report.
PAGE HEADING	A report group written at the top of each page. Exceptions: 1. The page heading is written after the report heading. 2. The page heading is suppressed when the page is explicitly reserved for the report heading or footing.
PAGE FOOTING	A report group written at the bottom of each page (exceptions: same as page heading).
DETAIL	This report group type is the only one which is not produced automatically but must be specified in a GENERATE statement. Each execution of this statement produces the specified detail report group at object time.
CONTROL HEADING	This report group is generated as a heading group for a series of detail report groups, due to a control break.
CONTROL FOOTING	This report group is generated as a footing group for a series of detail report groups, due to a control break. It usually sums up data concerning the preceding series of detail report groups.

Table 8-1: Report group types used in report group description

8.1.2 General description of the Procedure Division

In the Procedure Division, the programmer uses the INITIATE, GENERATE, and TERMINATE statements to instruct the Report Writer to produce the desired report according to its full description in the Data Division.

The INITIATE statement instructs the Report Writer to perform the initialization required for writing the specified report(s). This initialization enables the Report Writer to recognize, for example, whether a given GENERATE statement is the first GENERATE statement to be executed in the sequence of a program.

The GENERATE statement is responsible for creating the major part of the report. It causes a number of automatic functions to be performed in addition to generating a detail report group. For example, the control field test is carried out and a control break takes place, if necessary; that is, control footings and control headings are created. If the associated report group zone of the page does not provide space for a body group (detail report group, control footing, and control heading), then, automatically, the page footing is generated, a page change is implied, and the heading group is created. All of the functions involved in writing the individual report groups, such as incrementing or decrementing counters, supplying value, source, or sum information to the print fields, as well as positioning and writing the line(s) into which the print fields of a report group are structured, are performed automatically.

The TERMINATE statement instructs the Report Writer to complete the creation of the specified report(s). Thus, all control footings that belong to the report and, if present, the report footing as the last report group are written.

Within the range of the DECLARATIVES, the programmer may specify user procedures for report groups following a USE BEFORE REPORTING statement. As such procedures are executed immediately prior to the actual creation (editing, etc.) of the associated report group, this provides the programmer with a means of influencing the representation of the report group at object time.

The following four special registers are known to, and used by, the Report Writer:

LINE-COUNTER, PAGE-COUNTER, PRINT-SWITCH and CBL-CTR.

A report file is a sequentially organized file which is subject to the following restrictions:

Before entering an INITIATE statement, an OPEN OUTPUT or OPEN EXTEND statement must be performed. A TERMINATE statement must be followed by a CLOSE statement (without REEL or UNIT option). No further I-O statement may be issued for this file.

8.2 Language elements of the Data Division

REPORT clause

Function

The REPORT clause relates one or several reports described in the REPORT SECTION to a file in whose file description (FD) entry this clause is specified. Each of these reports is written line by line to the file assigned by the REPORT clause and described as an output file.

Format

```
{REPORT IS  }
{           } {report-name-1}...
{REPORTS ARE}
```

Syntax rules

1. The REPORT SECTION must be the last section of the Data Division, unless SUB-SCHEMA SECTION is also specified (see General format of the Data Division, page 138).
2. Each report-name specified in the REPORT clause must be the subject of a report description entry defining that name as a report-name.

General rules

1. The name of each report to be produced by the Report Writer must appear in the REPORT clause of the file description for the sequential output file to which the report is to be written.
2. Specification of several report-names in one REPORT clause relates these reports to the file whose file description entry contains the REPORT clause. When created, these reports are written to the assigned file regardless of the order in which the report names are specified and regardless of their formats, their lengths or any similar details.
3. Each report-name defined by a report description entry must be specified in one, and only one, REPORT clause; that is, a given report may be assigned to one file only.
4. The record format (variable, fixed, or unspecified length) is determined by the entry in the RECORD CONTAINS clause.

REPORT SECTION

Function

The REPORT SECTION describes the format and contents of the reports to be generated.

Format

```
REPORT SECTION.
{report-description-entry {report-group-description-entry} ... } ...
```

General rules

1. The REPORT SECTION must be the last section in the Data Division.
2. The report group description entries must all follow the related report description entry in a body.
3. A report group description entry formally corresponds to a record description of the FILE SECTION or WORKING-STORAGE SECTION. The report group description entry comprises all of the data description entries for a single report group. The first entry in the report group description begins with level-number 01. All associated data description entries must begin with level-number 02 (see "Report group description entries", page 589).

Report description entry

Function

The report description (RD) entry is used to name a report, define its page format, identify its lines, and for structuring the report. It may perform the following functions as required.

1. Specification of a character identifying the print lines of a report (CODE clause).
2. Declaration of a group hierarchy for structuring the report (CONTROL clause).
3. Definition of a page format by specifying vertical boundaries for report group specific regions (PAGE LIMIT clause).

Format

```
RD report-name

   [CODE clause]
   [CONTROL clause]
   [PAGE LIMIT clause]
```

Syntax rules

1. RD is the level indicator, which identifies the beginning of the report description and must immediately precede the report-name.
2. The report-name must be specified in a REPORT clause of the file description entry for the file on which the report is to be written.
3. No more than 31 data-names may be specified in a CONTROL clause within a report description.
4. The report-name identifies the report and, accordingly, must be unique.

The RD entry clauses are described on the pages that follow.

CODE clause

Function

The CODE clause specifies a character for identifying the print lines that belong to a specific report. This character is inserted in the first position of each print line of the report but must not be printed. It is used to separate the print lines of two or more reports that are interleaved or written consecutively on the report file.

Format

```
CODE literal
```

Syntax rules

1. The literal must be a two-character nonnumeric literal. The COBOL85 compiler uses only the first character of the literal for marking the report.
2. If a report description entry includes the CODE clause, all other report description entries assigned to the same output file (through the REPORT clause) must have a CODE clause also.
3. The character which is written at the beginning of each print line is associated with the literal.
4. The character for identifying the print lines of a report is inserted at the beginning of the line, following the carriage control character. This character must not appear in the print format.

General rule

The CODE clause must not be specified if the report is to be printed immediately ("online") or if it is assigned to the report file SYSLST. It must be ensured that reports do not overlap, i.e. that during the time lapse between the INITIATE and TERMINATE statement for one report no GENERATE statement is executed for another report.

CONTROL clause

Function

The CONTROL clause defines the data items that are used as control data items of the report to determine the group hierarchy and, thus, the hierarchical structuring of the report.

Format

```
[CONTROL IS  ] [{data-name-1}...        ]
{            } {                        }
[CONTROLS ARE] [FINAL [data-name-1]... ]
```

Syntax rules

1. No more than 31 data-names may be entered in the CONTROL clause, including the FINAL phrase if specified.
2. data-name-1... may be qualified but must not be indexed.
3. No data item may be subordinated to a data-name if its length is defined in the OCCURS clause as variable.
4. data-name-1... must not be defined in the REPORT SECTION but rather in the FILE SECTION or WORKING-STORAGE SECTION. A data-name entered in the CONTROL clause may be defined in the LINKAGE SECTION of the called program, provided that the called program is continuously available in memory from the time the report is initiated until its production is terminated.
5. The data item used may be up to 256 characters in length.
6. Each data-name-1 must identify a different data item.

   Two separate recurrences of data-name-1 must not refer to data items which (by redefinition) refer to the same memory locations.
7. A control data item is a data item which is specified in the CONTROL clause. It is tested whenever a GENERATE statement is executed for the same report in order to ascertain whether its value has changed since the last GENERATE statement was executed for that report.

   If such a change in value is found to have taken place, a "control break" occurs, i.e. special action (described below) will be taken before the detail report group specified by the GENERATE statement is written. If changes in value have occurred in several control data items when a GENERATE statement is executed, it is always the hierarchically supreme change in value to which all control break concepts (such as control break and control break level) are related.

8. FINAL, data-name-1... define the control hierarchy of the report.
9. The data-names, in the order in which they are listed from left to right, specify the levels of the control hierarchy from major to minor. The last (i.e. rightmost) data-name is assigned the lowest level, the last but one is assigned the lowest level, the last but one is assigned the second lowest level, and so on.
10. FINAL defines the hierarchically highest control break. The associated control heading is generated when the first GENERATE statement is executed; the associated control footing when the TERMINATE statement is executed.

General rules

1. The action implied by a control break depends on whether a control heading and/or a control footing, or neither of the two are defined for each control hierarchy level. If a control break occurs, the Report Writer creates the following control headings and control footings (as present), in the order shown below:

   a) Control footing of the lowest level.

   b) Control footing of the next higher level.

      .
      .
      .

   c) Control footing of the level responsible for the control break.

   d) Control heading of the level responsible for the control break.

   e) Control heading of the next lower level.

      .
      .
      .

   f) Control heading of the lowest level.

   Next, the Report Writer writes the detail report group initiated by the GENERATE statement.

   For example, when the control break data-names for a report are specified as YEAR, MONTH, and DAY (listed in this order in the CONTROL clause), associating each of these data-names with one control heading as well as one control footing, then, if a control break occurs for YEAR (that is, the contents of the data item YEAR has changed between two chronologically successive GENERATE statements), the body groups are printed in the following order:

Control footing	for DAY
Control footing	for MONTH
Control footing	for YEAR
Control heading	for YEARf
Control heading	for MONTH
Control heading	for DAY

Detail report group produced by the GENERATE statement.

2. If the Report Writer starts creating a body group and detects that one of the conditions for a page break exists, it will first write the page footing (if specified), then skip to the next page, then write the page heading (if specified), and finally generate the body group.
3. The creation of a detail group must be requested by the programmer from the Report Writer by means of an appropriate GENERATE statement (to define the detail group) in the Procedure Division. All other report groups such as report heading, report footing, page heading, page footing, control headings, and control footings are written automatically by the Report Writer as soon the respective conditions are satisfied.
4. In order to determine a control break, an alphanumeric compare is performed, regardless of how the control data items are described. This means, for example, that the Report Writer will detect a change of value when the contents of a control data item described with COMPUTATIONAL-3 is changed from hexadecimal "7F" to hexadecimal "7C", although both variant representations print out the positive number 7.
5. When a CONTROL clause is not specified for a report, control headings and control footings must not, and cannot, be defined for the report.

Example 8-1

Extract from a report:

```
JANUARY 14  B10                  4  B    8.36
            B10                  1  C    9.00
PURCHASES & COST FOR 1-14        5     $17.36  $136.36
JANUARY 15  B10                  2  A   16.00
```

The relevant report description entry includes this CONTROL clause:

```
CONTROLS ARE FINAL MONTH WDAY
```

Except for the print line beginning with PURCHASES and constituting the control footing for WDAY, all other print lines from the above report extract are detail groups (same report description). The control footing of WDAY was written because the data changed from January 14 to January 15.

PAGE LIMIT clause

Function

The PAGE LIMIT clause is used to define the length of a page and the vertical subdivisions within which report groups are presented.

Format

```
PAGE [[LIMIT IS  ]]           [[LINE ]]
     [{          }] integer-p [{     }]
     [[LIMITS ARE]]           [[LINES]]

      [HEADING integer-h]
      [FIRST DETAIL integer-d]
      [LAST DETAIL integer-e]
      [FOOTING integer-f]
```

Syntax rules

1. integer-p indicates the maximum possible number of print lines per page.
2. integer-p must not exceed 999.
3. The other integers of the PAGE LIMIT clause are line numbers. Since the numbering of the lines on a page starts with 1, integer-p may also be interpreted as a line number.
4. The integers of the PAGE LIMIT clause are subject to the following relationships:

   integer-h must be equal to or greater than 1.
   integer-d must be equal to or greater than integer-h.
   integer-e must be equal to or greater than integer-d.
   integer-f must be equal to or greater than integer-e.
   integer-p must be equal to or greater than integer-f.
5. integer-h of the HEADING phrase specifies the first line on the page on which anything may be written.
6. integer-h must not exceed 999.
7. integer-p of the LIMIT phrase specifies the last line on the page on which anything may be written.
8. integer-d of the FIRST DETAIL phrase specifies the lowest line number permitted for any body groups.
9. integer-d must not exceed 999.

10. integer-e of the LAST DETAIL phrase specifies the highest line number permitted for any detail groups and control headings.
11. integer-e must not exceed 999.
12. No more than 127 detail report groups (DETAIL) may be specified within a report description entry.
13. integer-f of the FOOTING phrase specifies the highest line number permitted for any control footings.
14. No more than 31 control footings may be specified within a report description entry.
15. integer-f must not exceed 999.
16. If the PAGE LIMIT clause is specified but one or more of the optional integers is omitted, the following values are internally assumed by default:

Omitted integer	**Value assumed when nothing is specified**
HEADING integer-h	1
FIRST DETAIL integer-d	Value of HEADING integer-h
LAST DETAIL integer-e	Value of FOOTING integer-f
FOOTING integer-f	Value of LIMIT integer-p

17. If the PAGE LIMIT clause is omitted entirely, the compiler assumes the following values for each integer of the PAGE LIMIT clause:

Integer	**Assumed value**
LIMIT integer-p	50
HEADING integer-h	1
FIRST DETAIL integer-d	1
LAST DETAIL integer-e	48
FOOTING integer-f	48

General rules

1. The Report Writer uses the phrases of the PAGE LIMIT clause for partitioning the page into regions. Only report groups of specific types may be printed in certain regions. The page regions for each type of report group are:

   a) If the report heading description includes the NEXT GROUP NEXT PAGE clause, the report heading prints from the line whose number is integer-h through the line whose number is integer-p. Otherwise, the report heading must not print beyond the line whose number is integer-d minus 1, which requires explicit entry of integer-d (see under b. below).

b) The page heading must lie in the region from integer h through integer d minus 1. This region does not exist if the value of integer-d is established by the compiler. It is therefore necessary for integer-d to be supplied explicitly in the PAGE LIMIT clause when printing a page heading or report heading which does not appear on a page by itself.

c) Detail report groups and control headings may be printed only in the region from integer-d through integer-e, these boundaries included.

d) Printout of control footings is permitted in the region from integer-d through integer-f.

e) A page footing may be printed only in the region from integer-f+1 through integer-p. This region does not exist if the PAGE LIMIT clause is supplied without the integer-f phrase. Therefore, in this case, printing a page footing requires that integer-f be specified explicitly.

f) If a whole page is associated with the report footing by means of LINE NEXT PAGE, the report footing may print in the region from integer-h through integer-p. Otherwise, report footings are subject to rule e).

2. NEXT GROUP and LINE clauses of the report group description entries must not conflict with the PAGE LIMIT clause; that is, the lines of a report group must be within the assigned region.

3. Table 8-2 is a schematic view of the partitioning of a page into regions in those cases where each integer of the PAGE LIMIT clause has a different value.

integer	Page region and valid entries				
	Region 1 REPORT HEADING and REPORT FOOTING	Region2 PAGE HEADING and (if any) REPORT HEADING	Region 3 DETAIL and CONTROL	Region 4 CONTROL FOOTING	Region5 PAGE FOOTING and (if any) REPORT FOOTING
integer-h					
integer-h +1					
.					
.					
.					
integer-d −1					
integer-d					
integer-d +1					
.					
.					
.					
integer-e −1					
integer-e					
integer-e +1					
.					
.					
.					
integer-f −1					
integer-f					
integer-f +1					
.					
.					
.					
integer-p −1					
integer-p					

Table 8-2 Page partitioning into regions (schematic)

4. Table 8-2 shows all print page regions which result when the PAGE LIMIT clause is specified with integer-h less than integer-d less than integer-e less than integer-f less than integer-p. The same assumption applies to the following rules for using these regions.

   Region 1:

   Scope: integer-h through integer-p.

   Contents: report heading or report footing.

   Rules: If the report heading description includes the NEXT GROUP clause with the NEXT PAGE phrase, a whole print page is reserved for the report heading; that is, no other report group is printed on this page. Whether the report head is printed using the whole or any part of the region, is determined by the LINE clause of the report heading description.

   If the report footing description includes the first (or only) LINE clause with the NEXT PAGE phrase, a whole print page is reserved for the report footing; that is, the same rules apply as to the report heading.

   Region 2:

   Scope: integer-h through integer-d minus 1.

   Contents: report heading and page heading, or page heading only.

   Rules: IThe report heading will be printed in region 2 only if its report group description does not include the NEXT GROUP PAGE clause. Since a report heading is printed only once per report, the report heading appears in region 2 only on the first page of the report.

   With the exception of those pages which are exclusively reserved for the report heading and report footing, a page heading, if defined, will appear in region 2 on all pages.

   If the first page is provided with both the report heading and the page heading, the page heading must be printed after the report heading. LINE and NEXT GROUP clauses of the two report description entries must not violate this rule.

integer	Page region and valid entries				
	Region 1 REPORT HEADING and REPORT FOOTING	Region 2 PAGE HEADING and (if any) REPORT HEADING	Region 3 DETAIL and CONTROL	Region 4 CONTROL FOOTING	Region 5 PAGE FOOTING and (if any) REPORT FOOTING
integer-h					
integer-h +1					
.					
.					
.					
integer-d −1					
integer-d					
integer-d +1					
.					
.					
.					
integer-e −1					
integer-e					
integer-e +1					
.					
.					
.					
integer-f −1					
integer-f					
integer-f +1					
.					
.					
.					
integer-p −1					
integer-p					

Table 8-2 Page partitioning into regions (schematic)

Region 3:

Scope: integer-d through integer-e.

Contents: body group.

Rules: As shown by the scope, region 4 generally extends beyond region 3 at the bottom.

IAny body group may be written in region 3 or in the overlapped portion of region 4. The remaining portion of region 4 may receive control footings only.

Region 4:

Scope: integer-d through integer-f.

Contents: body group.

Rules: As shown by the scope, region 4 generally extends beyond region 3 at the bottom.

IAny body group may be written in region 3 or in the overlapped portion of region 4. The remaining portion of region 4 may receive control footings only.

Irrespective of the body's group type, region 4 applies to the NEXT GROUP clause of this group.

Region 5:

Scope: integer-f+1 through integer-p.

Contents: report footing and page footing.

Rules: If the first LINE clause in the report description entry for the report footing does not contain the NEXT PAGE phrase, the report footing will be printed in region 5 on the last page of the report.

The page footing of a report is always written in region 5.

If both the page footing and the report footing are to be printed in region 5 of the last page, then the page footing must precede the report footing. The LINE clauses of the two report groups must not be in conflict with this rule.

Report group description entry

Function

The report group description entry describes and defines the format, type, and properties of a report group as a series of elementary items and items associated with information. These items, organized in lines and columns, form the line(s) to be printed in a report group.

A prepositioning to the next report group to be written may be achieved by a NEXT GROUP phrase.

Using the type of report group, demands are placed both on the description of the report group and on the Report Writer when the report group is created. For example, the page positioning of a report group is also determined by the type of the report group. Much the same is true of the time the report group is produced. Summations, for instance, can be defined in control footings only. They are executed immediately preceding the creation of detail groups or the control footings concerned, depending on type of summation.

Format

```
01 [data-name-1]
    TYPE clause
   [LINE clause]
   [NEXT GROUP clause]

02 [data-name-2]
   [LINE clause]
   [COLUMN clause]
   [GROUP INDICATE clause]
    PICTURE clause
   [BLANK WHEN ZERO clause]
   [JUSTIFIED RIGHT clause]
   [SIGN clause]
   [USAGE clause]

   {SOURCE clause}
   {SUM clause   }
   {VALUE clause }
```

Syntax rules

1. The description of a report group must begin with a 01-level entry, followed by at least one 02-level entry.
2. With the exception of the data-name, which must immediately follow the level-number, the clauses may be written in any order.

3. A report consists of a series of report groups. Up to seven different report group types (see TYPE clause, page 618) may appear in the report.

   By definition, a report group consists of one or several elementary items, where each item with a COLUMN clause (printable item) must be assigned to one line. Report groups which do not contain printable items are nonprintable. A printable report group is a unit comprising one or more lines.

   With the exception of the report heading and the report footing, any report group may be created several times in the course of creating report; in this case, there must be no change in the report group format or in the constant information contents.

4. No more than 127 report group description entries may be specified in a report description entry.

5. The data-name which immediately follows the level-number 01 is the name of the report group. It must be supplied if the report group is to be referenced directly in a GENERATE statement (detail group), a USE BEFORE REPORTING statement, an UPON phrase within a SUM clause, or is to be used for qualification.

6. The TYPE clause specifies the type of report group, enabling the Report Writer to determine the modalities (when, where) for producing that report group.

7. The LINE clause assigns the printable items specified within its range to a line of the current or next print page. The first LINE clause of a report group description entry therefore determines the positioning of that report group. The range extends until the next LINE clause or the end of the report group entry.

8. The NEXT GROUP clause causes (after a report group is printed) the next report group to be positioned to the line number supplied in the phrases of the NEXT GROUP clause.

   The COLUMN, GROUP INDICATE, BLANK, JUSTIFIED, and PICTURE clauses define the location and format of the printable data item for a particular print line of the report group.

9. SOURCE, SUM, and VALUE clauses associate the data items with the respective information. In addition, the SUM clause also defines an internal sum counter whose contents is automatically updated by the Report Writer.

10. Each printable item must be covered by a LINE clause. This LINE clause represents a print line.

General rules

1. The name of a report group may be specified in the REPORT SECTION and in the Procedure Division. Only names of control footings and detail groups may be supplied in the REPORT SECTION. While the name of a detail group may be defined in the UPON phrase of a SUM clause, the name of a control footing may be used only for qualifying a sum counter. In the Procedure Division, the name of a report group may be specified only in two cases:

   a) The name of a detail group may be specified in a GENERATE statement.

   b) In a USE BEFORE REPORTING statement, any report group name (except names of detail groups) may be used.

   As a qualifier for a report group name, only the associated report name is allowable.

2. A data-name immediately following the level-number 02 of a report group entry should not be specified unless that data item description includes a SUM clause. In this case, the data-name will be interpreted as the name of the sum counter which is established as a result of the SUM clause (see "SUM clause", page 610). The name of a sum counter may again be specified in a SUM clause. If the data-name is supplied although no SUM clause is present, in other words, if the data-name is defined as the name of the item, its use is illegal.

   Sum counters may only be qualified by report group names and/or report names.

Report group description entry clause summary

Table 8-3 lists the report group entry clauses and briefly describes each.

The BLANK WHEN ZERO, JUSTIFIED, PICTURE, USAGE and VALUE clauses are also used in other sections of the Data Division; that is, they are assumed to be known here (refer to the pertinent descriptions).

All other report entry clauses are described on the following pages, in alphabetical order.

Clause	Brief description
BLANK WHEN ZERO	Indicates that a printable item is not to be printed when its contents are zero.
COLUMN	Defines an item as printable by specifying the starting position (column number) of that field on the print line.
GROUP INDICATE	Indicates that the printable item is to be printed only if the detail group with which it is associated is written for the first time in the report, after a page advance or a control break.
JUSTIFIED	Explicitly specifies the positioning of data in the associated item.
LINE	Specifies the line on the report page on which one or more printable items within its range are to be printed.
NEXT GROUP	Specifies prepositioning for the next report group to be printed.
PICTURE	Specifies the characteristics of a data item or of a data item and an internal sum counter.
SIGN	Specifies the position and the mode of representation of the operational sign for numeric data items.
SOURCE	Associates a data item with a sending field as its source of information.
SUM	Indicates how and which addends are to be accumulated in the internal sum counter. The sum counter is built automatically. Also, this clause associates the data item with the sum counter as its source of information.
TYPE	Specifies the particular type of report group being described.
USAGE	Specifies the data format used for storing an elementary item.
VALUE	Defines a constant value for a printable item.

Table 8-3: Report group entry clauses

COLUMN clause

Function

The COLUMN clause defines a data item as a printable field by specifying its column number (integer) to indicate the starting position of that item with respect to the print line.

Format

```
COLUMN integer
```

Syntax rules

1. integer must be an unsigned integer in the range of 1 ≤ integer ≤ 251.
2. The column number (integer) specifies the position in which the first (leftmost) character position of the printable item is to appear on the print line. The first printable character position of the print line is considered to be column 1. The highest column number depends on the type of printer being used.

General rules

1. The presence of the COLUMN clause in the description entry for a data item containing the mandatory PICTURE clause as well as either a SOURCE, SUM, or VALUE clause constitutes a printable item.
2. Every entry that defines a printable item must either contain, or be preceded by (in the same report group description), a LINE clause. The printable item will be printed on the print line specified by the LINE clause.
3. Within the scope of a LINE clause (that is, before the next LINE clause occurs or until the report the report entry ends), the printable items must be defined in ascending column number sequence. The printable items are printed on the line in the order in which they were defined.
4. Printable items on a print line must not be so defined that they overlap each other (concerning the COLUMN and PICTURE clauses).
5. Immediately before the printable items in a line are printed, i.e. before the print line is written to the report file, the information is automatically moved to these items by implicit MOVE statements. Depending on whether the description of the receiving item (=printable item) includes the SOURCE, VALUE, or SUM clause, the sending item will be the identifier from the SOURCE clause, the literal from the VALUE clause, or the internal sum counter which was established for the SUM clause.

6. The Report Writer supplies space characters for all positions of a print line that are not occupied by printable items.

Example 8-2

Assume the following description of an elementary item in a report group entry:

```
02 LINE 3 COLUMN 30 PIC A(12) VALUE IS "EXPENDITURES"
```

When this report group is produced, the character-string EXPENDITURES will be printed on line 3 of the current report page, starting at column 30.

GROUP INDICATE clause

Function

The GROUP INDICATE clause indicates to the Report Writer that the printable item with which it is associated is to be printed only if the detail report group is written for the first time on a page of the report or for the first time after a control break.

Format

```
GROUP INDICATE
```

Syntax rules

1. The GROUP INDICATE clause may be used only in a detail report group. The item description containing the GROUP INDICATE clause must also include a COLUMN clause.
2. The GROUP INDICATE clause causes the Report Writer to print the associated printable item according to the SOURCE or VALUE clause specified, only in the following cases:

   a) On the first presentation of the detail report group in the report

   b) On the first presentation of the detail report group after a page advance or

   c) On the first presentation of the detail report group after a control break.

   In all other cases, space characters are substituted for the printable field, suppressing the associated data in the print layout.

Example 8-3

a) Data Division entries:

```
          .
          .
          .
     CONTROLS ARE FINAL, MONTH, WDAY
          .
          .
          .
01   DETAIL-LINE PLUS 1 TYPE DETAIL.
     02  COLUMN 2 GROUP INDICATE PIC A(9)
         SOURCE MONTHNAME (MONTH).
     02  COLUMN 13 GROUP INDICATE PIC 99 SOURCE WDAY.
          .
          .
          .
```

b) Excerpt from a report

```
          .
          .
          .
      JANUARY 15 B11 16 A 20.00
                 B11  4 B 13.45
                 B11 20 D 35.40
```

The above extract from a report comprises three detail report groups of the same report group definition, created immediately following a control break (change in the current contents of DAY).The current values JANUARY and 15 of the sending items (SOURCE items) for the first two printable items of this 1-line detail report group appear, therefore, only in the first line (detail report group).

LINE clause

Function

The LINE clause specifies the line on the report page on which one or more printable items defined within its scope (see COLUMN clause, page 593) are to be printed. In special application, it is used simply for page advance.

Format

```
                 ⎧integer-1     ⎫
LINE NUMBER IS   ⎨PLUS integer-2⎬
                 ⎩NEXT PAGE     ⎭
```

Syntax rules

1. integer-1 and integer-2 may only be specified as unsigned integers. integer-1 must be greater than or equal to 1, integer-2 must be greater than or equal to 0. The value of integer-1 or integer-2 must not exceed 999.
2. A LINE clause with integer-1 phrase specified is called an absolute LINE clause. integer-1 is interpreted as a line number. Therefore, when creating the associated report group, integer-1 indicates the particular line on the report page on which the printable items which belong to the LINE clause and, thus, are organized as a line will ultimately be written. In this way, an absolute LINE clause always defines a print line.
3. A LINE clause with PLUS integer-2 phrase specified is called a relative LINE clause. The print line to receive the printable items associated with a relative LINE clause at the time the particular report group is generated, is determined relative to the last vertical positioning. For this purpose, integer-2 is added to the number of the line on which the report page is presently positioned (see "NEXT GROUP clause", page 602 and "LINE COUNTER special register", page 628). The sum then designates the number of the next line to be printed.

   Deviations from the above rules are only permissible for special types of report groups, when the relative LINE clause was used for the first line of a report group. These deviations will be discussed in due course.
4. A LINE clause with a NEXT PAGE phrase causes the associated report group to be printed on a free page (usually the next page), advancing the page before the report group is printed.

   Whether a LINE NEXT PAGE clause additionally specifies a print line can be seen from the rules described below.

General rules

The following programming rules are classified into two categories -"General notes" and "Report group syntax notes". Rules in the latter category describe the use of the LINE clause in the descriptions of the different types of report groups.

The abbreviations used in the report group notes have the following meaning.

Abbreviation	Meaning
A	one or more absolute LINE clauses
R	one or more relative LINE clauses
NP	a LINE clause with NEXT PAGE phrase

General notes

1. A printable report group item that is to be written on a line of the report page must either itself contain a LINE clause in its description or its description must be preceded, within the associated report group description entry, by a LINE clause. Conversely, all printable items in the associated report group description entry, which follow the LINE clause and precede the next LINE clause or the end of that report group description entry, will be written on the line determined by the LINE clause. Subsequent printable items from the report group entry are printed combined into lines in the same manner.
2. The vertical spacing parameters (LINE and NEXT PAGE clauses) must be selected for the different report group types in such a way that these report groups can be printed within the page regions provided for them (see also "PAGE clause", page 582 and "TYPE clause", page 618). It is not possible to continue a report group in the associated specific region on another page.

   If a body group cannot be printed in its specific region simply because part of that region is no longer available, then that body group as a whole will be written in the same region on the next page, after creating first the page footing on the old page and the page heading on the new page.
3. If a report group entry includes (one or more) absolute LINE clauses, they must all be specified:

   a) preceding the first relative LINE clause if any, and

   b) in the order of ascending integers.
4. The LINE clause with the NEXT PAGE phrase may only be specified in a report group description entry (01 level) to position to the next page.

Report group syntax notes

5. Report heading

   The following sequences of LINE clauses may be used in describing a report heading group:

   ```
   {A   }
   {    }
   {A R }
   ```

6. Page heading

   Only the following sequences of LINES clauses may be used in describing a page heading group:

   ```
   {A   }
   {A R }
   {R   }
   ```

   When the first LINE clause from the description of the page heading is a relative clause, then, generally, the first line of the report group is printed relative to the region boundary integer-h of the PAGE LIMIT clause (see "PAGE LIMIT clause", page 582). Only when the report heading was printed on the same page will there be deviation in that the first line of the report group is printed relative to the vertical spacing resulting from the creation of the report group heading.

7. Body groups

   Detail groups, control headings, and control footings are referred to generically as body groups.

   a) The following sequences of LINE clauses may be used in describing a body group:

   ```
   {NP     }
   {NP A   }
   {NP A R }
   {NP P   }
   {A      }
   {A   R  }
   {R      }
   ```

   b) When NP A or NP A R sequences are specified for a LINE NEXT PAGE clause, NP provides page changing instructions, and does not define a print line. Thus, the first absolute LINE clause must appear with, or prior to, the first entry defining a printable item.

   c) If a LINE clause with NEXT PAGE phrase is specified as the only LINE clause or with the sequence NP R, this implies not only page advance but also vertical spacing of the first line of that report group.

d) A LINE NEXT PAGE clause in the description of a body group indicates to the Report Writer that the body group is to be printed on the next page which is not yet occupied by a body group. This may lead to a situation where no page advance is carried out. This will certainly be the case when the body group is to be written as the first body group on a particular report.

e) If a body group defined with a LINE clause sequence NP, NP R, or R is the first body group to be printed on a page, then the first print line of that body group will be printed on the line whose number is integer-d as specified in the PAGE clause. However, it may well be the case that the paper has already been advanced beyond region boundary integer-d, e.g. through the last body group printed because of a NEXT GROUP clause (see page 602). In this case, the first line of the body group will be written on the line immediately following the present positioning.

f) If a body group whose first LINE clause is relative is not to be printed as the first body group on a page, the present positioning (line number) is advanced by the number of lines specified by integer-2 in the first LINE clause, to the new position where the first line of this report group will be printed.

8. Page footing

   Only the following sequences of LINE clauses may be used in describing a page footing:

   ```
   { A   }
   {     }
   { A R }
   ```

9. Report footing

   a) The following LINE clause sequence alternatives may be used in describing a report footing:

   ```
   { NP A   }
   { NP A R }
   { A      }
   { A   R  }
   { R      }
   ```

   b) The LINE clause with the NEXT PAGE phrase specified is used solely for page advance.Therefore, the first absolute LINE clause must appear with, or prior to, the description of the first printable item, in order to permit positioning to the first line of the report group.

Example 8-4

```
01  DETAIL-GROUP TYPE DETAIL LINE NEXT PAGE.
    02  LINE 10 COLUMN 1  PIC X(10) VALUE "1ST ITEM".
    02           COLUMN 15 PIC X(4)  SOURCE CARD-FIELD-1.
    02  LINE 12 COLUMN 1  PIC X(10) VALUE "2ND ITEM".
    02           COLUMN 15 PIC 9(5)  SOURCE-WORK-FIELD-1.
```

The two-line detail group is printed on lines 10 and 12 of a page not yet used for other body groups, because the LINE clause sequence is NP A.

Example 8-5

```
01  DETAIL-LINE LINE PLUS 1 TYPE DETAIL.
    02  COLUMN 2 GROUP INDICATE PIC A(9) SOURCE FIELD-NO-1.
```

This example shows a report group whose description includes only one relative LINE clause. The report group will be printed on the line resulting from the present position being advanced by one line. If the report group is to be printed as the first body group on that page, and the present positioning has not gone beyond the integer-d region boundary (FIRST DETAIL phrase if the PAGE LIMIT clause), the report group is printed to the line whose number is integer-d.

NEXT GROUP clause

Function

The NEXT GROUP clause indicates to the Report Writer the particular line on a page to which the paper is to advance after printing the last line of the report group containing the NEXT GROUP clause, thereby causing the paper to be prepositioned for the next (chronologically speaking) report group to be printed (minimum distance from the next report group).

Format

```
                  ⎧integer-1      ⎫
NEXT GROUP IS     ⎨PLUS integer-2 ⎬
                  ⎩NEXT PAGE      ⎭
```

Syntax rules

1. integer-1 and integer-2 must be specified as unsigned integers equal to or greater than 1 . Their value must not exceed 999.
2. The NEXT GROUP clause must not be specified in a report group description entry unless a LINE clause has been specified.
3. A NEXT GROUP clause with integer-1 phrase is called an absolute NEXT GROUP clause. After printing the last line of the associated report group, the Report Writer advances the paper to the line whose number is specified by integer-1. This may also involve a page advance, printing the page footing on the previous page and the page heading on the new page (for body groups only).
4. A NEXT GROUP clause with PLUS integer-2 phrase specified is called a relative NEXT GROUP clause. Positioning is advanced by integer-2 lines from the last print line of the associated report group.
5. A NEXT GROUP clause with NEXT PAGE phrase specified indicates that the current page is considered to be full; that is, after the associated report group is printed, a page advance is executed (automatic generation of page footing and page heading for body groups).
6. If in a USE BEFORE REPORTING procedure of a control footing, the PRINT SWITCH special register is set to 1 by a MOVE statement, the function of the NEXT GROUP clause is suppressed by the Report Writer Control System.

General rules

1. The NEXT GROUP clause may appear only in a 01-level report group description entry.
2. Rules for the report heading
   a) If the report heading is to appear on a page by itself, its description must contain the NEXT GROUP clause with the NEXT PAGE phrase.
   b) If the report heading is not to appear on a page by itself, the following rules must be observed:
      – The NEXT GROUP clause must not include the NEXT PAGE phrase.
      – integer-1 of an absolute NEXT GROUP clause must indicate a line number greater than the one assigned to the last print line of the report heading.
      – An absolute or relative NEXT GROUP clause must be selected in such a way that the page heading as the next report group can still be contained in its specific region. A positioning to the line with the number integer-d (see "PAGE LIMIT clause", page 582) or even to a line which has a higher number owing to the NEXT GROUP clause is illegal.
3. Rule for the page heading

   The NEXT GROUP clause must not be used in describing the page heading.
4. Rules for body groups
   a) integer-1 of an absolute NEXT GROUP clause must be greater than or equal to integer-d yet less than or equal to integer-f (see "PAGE LIMIT clause", page 582).

      If the line number integer-1 of the absolute NEXT GROUP clause is less than or equal to the number of the line that was printed as the last line of the body group whose description includes the NEXT GROUP clause, the Report Writer executes a page advance (automatic generation of page footing and page heading included) and positions on the line specified by integer-1.
   b) If a relative NEXT GROUP clause would advance the paper beyond the lower boundary specified by integer-f (see "PAGE LIMIT clause"), the Report Writer executes a page advance (generation of page footing and page heading included) and positions on the upper region boundary specified by integer-d (see "PAGE LIMIT clause").
   c) The NEXT GROUP clause with the NEXT PAGE phrase specified indicates that no further body group is to be written on the current page. The Report Writer positions on the upper region boundary integer-d of the next page (including the creation of page footing and page heading).

d) If several control footings are created in immediate succession as a result of a control break, the Report Writer can be instructed to execute the function of the NEXT GROUP clause only for the particular control footing associated with the level of hierarchy at which the control break occurred (see "CBL-CTR special register", page 631).

5. Rule for the page footing

   The NEXT GROUP clause must not be used in describing the page footing.

6. Rule for the report footing

   The NEXT GROUP clause must not be used in describing the report footing.

Example 8-6

```
01  LINE PLUS 2 NEXT GROUP PLUS 1.
    TYPE CONTROL FOOTING WDAY.
    02...
```

The NEXT GROUP clause causes the Report Writer to advance only one line after the above control footing is created.

SIGN clause

Function

The SIGN clause specifies the position and the mode of representation of the operational sign for numeric data items.

Format

```
[SIGN IS] {LEADING  } [SEPARATE CHARACTER]
          {TRAILING }
```

Syntax rules

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S.
2. The data description entries to which the SIGN clause applies must be described, explicitly or implicitly, as USAGE IS DISPLAY.
3. If the SIGN clause is specified in a report group description entry, the SIGN clause must contain the SEPARATE CHARACTER phrase.
4. The SIGN clause specifies the position and the mode of representation of the operational sign. It applies only to numeric data description entries whose PICTURE contains the character S. The S indicates the presence, but not the mode of representation, of the operational sign.
5. A numeric data description entry whose PICTURE contains the character S, but to which no SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character S. (For representation of the operational sign see "USAGE clause", page 192.)

General rules

1. The following rules apply with respect to the required SEPARATE CHARACTER phrase:
   a) The letter S in a PICTURE character-string is counted in determining the size of the item.
   b) The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

c) The operational signs for positive and negative are the standard data format characters + and −, respectively.

2. Every numeric data description entry whose PICTURE character-string contains the character S is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

USAGE clause

Format

```
[USAGE IS] DISPLAY
```

Syntax rule

In the REPORT SECTION, the USAGE clause is only used in order to specify the data format of printable elementary items.

For further syntax rules and general rules relating to USAGE IS DISPLAY, see "USAGE clause", (page 192).

SOURCE clause

Function

The SOURCE clause identifies the data item whose contents the Report Writer (by an implicit MOVE statement) will move to the printable item whose description includes the SOURCE clause, when this item is to be printed.

Format

```
SOURCE IS identifier
```

Syntax rule

Any identifier defined in any section of the Data Division may be specified in a SOURCE clause. In the case of the REPORT SECTION, however, a SOURCE clause may use only the PAGE-COUNTER special register, the LINE-COUNTER special register, or a sum counter if it is associated with the report in whose description the SOURCE clause appears.

General rules

1. The description of an elementary item containing a SOURCE clause must also contain a COLUMN clause, that is, the field must be printable.
2. The SOURCE clause in conjunction with the COLUMN clause generates an implicit MOVE statement. For this MOVE statement, the source (or sending) field is defined by the identifier from the SOURCE clause. The receiving field is the printable item whose description includes the SOURCE clause. The picture strings of the PICTURE clauses of the two fields must adhere to the rules for sending items in the MOVE statement (see page 294).
3. As the SOURCE clause can never change the value of the data item it specifies, it may also refer to a control data item. Generally, the system prints the value of the sending field as it is when the MOVE statement is executed (creation of the associated report group). If the previous values of the control break data items are in control footings, Function 1 of the CBL-CTR special register (see page 631) should be used.

Example 8-7

```
FILE SECTION.
    ...
    02 DEPT  PIC XXX.
    ...
REPORT SECTION.
    ...
    02 COLUMN 19 PIC XXX SOURCE DEPT.
    ...
```

The SOURCE clause has the effect that the value in the data item DEPT is moved to the printable item concerned when the associated report group is printed.

SUM clause

Function

The SUM clause instructs the Report Writer to create and (immediately or subsequently) print arithmetic sums of data items selected from the detailed information that constitutes the report. The SUM clause indicates to the Report Writer the addends which are to be used for summation. In addition, this clause provides a numeric item, which is generated automatically, for the accumulation of the addends. This field, i.e. the sum counter, is also the sending field (source item) for the implicit MOVE statement used to edit the printable field whose description contains the SUM clause.

Format

```
SUM {identifier-1}... [UPON data-name-1]

    [          {data-name-2}]
    [RESET ON  {           }]
    [          {FINAL      }]
```

Syntax rules

1. An identifier used as an addend in a SUM clause must be defined in the FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION or REPORT SECTION. From the REPORT SECTION, only the name of a sum counter may be referenced as an addend of a SUM clause.
2. Each identifier used as an addend in a SUM clause must represent a numeric data item.
3. data-name-1 is permitted only as the name of a detail report group that is defined in the current report description.
4. FINAL or data-name-2 must be specified in the CONTROL clause of the current report description.
5. identifier-1... specify the items to be accumulated in the sum counter.
6. The UPON phrase has the effect that the specified addends are added up only when a GENERATE statement is executed referencing the very detail group defined in the UPON phrase (see "Use of UPON phrase", page 615).
7. The RESET phrase overrides the standard reset function of the sum counter to zero (see "Use of RESET phrase", page 616).

General rules

1. The SUM clause can only be specified in control footing descriptors.

   Assume the following excerpt from a report:

```
JANUARY 02 B10                  2 A   3.00
           B12                  1 A   4.00
           B12                  3 C  17.00
PURCHASES & COST FOR 1-02       6   $24.00
```

   Printed here are the detailed cost figures and their sum for January 2.

   The detailed cost figures shown on the first three print lines were printed as a result of a GENERATE statement referring to the following detail group:

```
01  DETAIL-LINE TYPE DETAIL.
    02  ...
    .
    .
    .
    02  COLUMN 50 PICTURE ZZ9.99 SOURCE COST.
```

   This GENERATE statement was executed three times without control break, causing the detail report group to be printed three times in succession. The data item by the name of COST, described in the FILE SECTION of the Data Division as:

```
  02 COST PICTURE 999V99.
```

   supplied the detailed cost figures.

   When that GENERATE statement was executed once more, the value of the control data item DAY had been changed from 2 to a different value. The following control footing:

```
01  ... TYPE CONTROL FOOTING WDAY.
    02  ...
    .
    .
    .
    02  SUM-DAY COLUMN 49 PICTURE $$$9.99 SUM COST.
```

   was generated as the fourth line of the above partial report, with the contents of sum counter SUM-DAY printed starting at column 49.

   Summation took place as follows:

   During compilation, the compiler created a sum counter (named SUM-DAY) in response to the SUM clause. At object time, when executing the appropriate INITIATE statement, the Report Writer resets the sum counter to zero. Then, each time the GENERATE DETAIL-LINE statement was executed, the current value of the data item COST was added to the contents of the SUM-DAY counter. Since the Report Writer

performs this summation (see under 5. "Detail summation") immediately following the control break test and the action resulting from that test, the sum of the cost figures for January 2 is printed out with the control group footing for DAY. When the control footings are created, the sum counter is finally reset to zero, as the SUM clause contains no RESET phrase.

2. Use of the PICTURE clause

   If the description of a data item, within a given report group entry, includes a SUM clause, the associated PICTURE clause describes not only the data item but also the sum counter which the compiler will establish due to the SUM clause. The data item, if printable, is used for printing out the contents of the associated sum counter. The PICTURE clause must define the data item as a numeric or numeric-edited data item, where editing symbols for sum counters will be ignored.

3. Use of the sum counter

   If the data item entry that contains a SUM clause has a data-name immediately following the level-number, that data-name is the name of the internal sum counter, which can thus be accessed by the programmer (for example, for rounding its contents prior to printing). The sum counter is a compiler-generated data item whose USAGE is COMP-3 and whose numeric characteristics are described in the specified PICTURE clause.

4. Types of summation

   The programmer can specify three types of summation:
   detail-incrementing, rolling forward, and crossfooting.

5. Detail-incrementing

   The time at which the Report Writer adds up an addend (identifier-1... from the SUM clause) in the related sum counter depends on the addend itself.

   The addends used for detail-incrementing are those which are not themselves sum counters, in other words, are defined outside the REPORT SECTION.

   Detail-incrementing is the basis for the other two types of summation. The term "detail-incrementing" derives from the fact that typically the addends involved in it are printed with the detail groups of the report.

   Detail-incrementing occurs each time that GENERATE statements are executed. Therefore, the programmer must ensure that the operands used for detail-incrementing contain the required values at the time that GENERATE statements are executed. If a SUM clause uses the UPON phrase, the addends in that SUM clause are added into their sum counter only when this detail-incrementing operation takes place in executing a GENERATE statement referring to the same detail group as the UPON phrase (there is, therefore, no point in using the UPON phrase for a summary report). However, if the

SUM clause does not include the UPON phrase, then those addends which are not defined as sum counters are added to their related sum counters when any GENERATE statement for the report is executed (detail-incrementing).

The Report Writer performs detail-incrementing only after taking certain actions as regards control break (test; creation of the control footings and headings if test is positive). This control break processing also includes resetting the sum counters to zero after creating the control footing whose description contains the corresponding SUM clauses (see "RESET phrase", page 616). This ensures that the printed sum will contain only the values of the addends for the particular series of detail groups which is concluded by the associated control footing (for example, the sum of the cost figures for January 2).

6. Rolling-forward (hierarchical summation)

   The prerequisite for this kind of summation is that a SUM clause of a control footing must specify as an addend at least one sum counter which was defined as the result of a SUM clause of a hierarchically lower (that is, less inclusive) control footing. Therefore, rolling-forward cannot be designated unless a report description includes at least two control footings whose descriptions each contain at least one SUM clause.

   The contents of a sum counter which is specified as an addend in the SUM clause of another control footing will be added, at the time the associated (hierarchically lower) control footing is generated, to the contents of the sum counter in whose SUM clause it appears as an addend.

   Example 8-8 illustrates rolling forward:

Example 8-8

```
01  ... TYPE CONTROL FOOTING MONTH.
    02  ...
    .
    .
    02  SUM-MONTH COLUMN 46 PICTURE $$$9.99 SUM
        SUM-DAY.
```

In the above control footing description, the rolling-forward function is specified in conjunction with the following control footing description (see example 8-7):

```
01  ... TYPE CONTROL FOOTING WDAY.
    02  ...
    .
    .
    02  SUM-DAY COLUMN 49 PICTURE $$$9.99 SUM COST.
```

For each creation of the control footing with the control item DAY, the Report Writer adds the contents of the sum counter SUM-DAY to the contents of the sum counter SUM-MONTH, before resetting SUM-DAY to zero. If either a control break or execution of a TERMINATE statement causes the control footing to be generated with MONTH, the sum counter SUM-MONTH (before resetting to zero) will contain the sum of all day-sums (values of SUM-DAY at summation times) of the current month.

7. Crossfooting (adding hierarchically equal sums)

   This type of summation takes place when a SUM clause contains, as addends, the names of sum counters defined by other SUM clauses in the same control footing. Normally, such addends are sum counters whose values are created through detail-incrementing.

Example 8-9

```
01  MINOR TYPE CONTROL FOOTING...
    02  SUM-1 SUM WORKING-ITEM-1...
    02  SUM-2 SUM WORKING-ITEM-2...
    02  SUM   SUM SUM-1 SUM-2...
```

WORKING-ITEM-1 and WORKING-ITEM-2 are data items defined in the WORKING-STORAGE SECTION of the Data Division. Sum counter SUM accumulates the values of SUM-1 and SUM-2, previously generated through detail-incrementing.
The Report Writer performs crossfooting just before printing the control footing concerned. If more than one SUM clause requires such addition, the order of execution is determined by the sequence of these SUM clauses. This order is essential to the result of the addition.

Obviously, this type of addition is carried out before rolling-forward, thereby ensuring that sums hierarchically equivalent in summation may also be rolled forward.

Example 8-10

```
     ...
     CONTROLS ARE STATE, CITY.
     ...

 01  LINE PLUS 2 TYPE CONTROL FOOTING CITY.
     02  SUM-1 SUM MALES...
     02  SUM-2 SUM FEMALES...
     02  SUM-CITY SUM SUM-1, SUM-2...
 01  LINE PLUS 1 TYPE CONTROL FOOTING STATE.
     02  SUM-STATE SUM SUM-CITY...

     ...
```

The values accumulated in sum counter SUM-CITY by crossfooting the values from the hierarchically equivalent sum counters SUM-1 and SUM-2 (detail-incrementing) are rolled forward in sum counter SUM-STATE (the control footing with STATE is higher in hierarchy than the control footing with CITY). This is possible only because the sum counter SUM-CITY contains the proper value before rolling-forward takes place in the sum counter SUM-STATE.

8. Mixing operands

   A SUM clause that does not contain an UPON phrase may include one or more of each of the following kinds of operands (= addends):

   a) Operands defined in the FILE SECTION, WORKING-STORAGE SECTION and LINKAGE SECTION.

   b) Operands defined as sum counters in a hierarchically inferior control footing.

   c) Operands defined as sum counters in the same control footing (whose description contains the SUM clause).

      Summing for each of the above kinds of operands occurs at the times indicated in the preceding discussions.

9. Use of the UPON phrase

   a) When an UPON phrase is used in a SUM clause, all addends of that clause must be defined outside the REPORT SECTION; that is, they may be defined only in the FILE SECTION, WORKING-STORAGE SECTION and LINKAGE SECTION.

   b) The UPON phrase has the effect of preventing detail-incrementing of the addends from the present SUM clause, unless a GENERATE statement is executed specifying the detail report group indicating in the UPON phrase. A detail-incrementing caused by any other GENERATE statement will not, therefore, affect any of the addends in the SUM clause in question.

Example 8-11

```
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  MINOR 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.
```

Because MUELLER and MEIER are the names of two different records on the same file, they cannot be available in memory concurrently. When a MUELLER record is read, the statement GENERATE MUELLER-DETAIL is executed; at that time, the current value of MUELLER is added to sum counter SUM-1. The present value of MEIER, on the other hand, is not added to the contents of sum counter SUM-2 at this time. When a MEIER record is read, the statement GENERATE MEIER-DETAIL is executed; at this time, the detail-incrementing occurs in sum counter SUM-2, and not in SUM-1.

10. Use of the RESET phrase

    a) Only a data-name (FINAL included) supplied in the CONTROL clause of the same report may be used in a RESET phrase. Moreover, the control data item must be at a higher level in hierarchy than the control footing whose description contains the RESET phrase.

    b) Normally, the Report Writer resets a sum counter to zero immediately after printing the control footing in whose description it is defined. A sum counter whose SUM clause contains the RESET phrase will be reset to zero only at a time when the (explicit or implicit) control footing for the control data item (or FINAL) that appears in the RESET phrase, is (or would be) created. Thus, the RESET phrase serves the purpose of creating a total for the specified hierarchical level.

Example 8-12

```
01  ... TYPE CONTROL FOOTING DAY.
    02  ...
    .
    .
    02  COLUMN 65 PIC $$$$9.99 SUM COST RESET ON FINAL.
01  ... TYPE CONTROL FOOTING FINAL.
    02  ...
    .
    .
    02  COLUMN 45 PIC $$$$9.99 SUM SUM-DAY.
```

Because the SUM clause in the description of the control footing with the control item DAY contains the phrase RESET ON FINAL, the current value of the associated sum counter is printed every time the control footing of DAY is generated, without ever resetting the sum counter to zero. Only when the control footing for FINAL is generated will the sum counter be reset to zero. Therefore, each printed control footing for DAY shows the running cost figures from the first detail report group of the report (1st day) through to the last detail report group written before the current control heading.

A control data item that appears in a RESET phrase does not have to be associated with a control footing. A sum counter will be reset, as mentioned earlier, even when no control footing exists for a control item specified in a RESET entry.

11. Actions taken by the Report Writer

    When generating a control footing, the Report Writer executes the following steps (schematically speaking, because steps may be omitted):

    a) Adding hierarchically equivalent sums (crossfooting).

    b) Execution of the USE BEFORE REPORTING procedures for the control footing (see page 626).

    c) PRINT SWITCH test.
    If the value of the PRINT SWITCH special register is 1, step d) is skipped, i.e. step e) immediately follows step c) after resetting the special register to zero. Otherwise, step d) comes next.

    d) Creation of the control footing (if printable).

    e) Hierarchical incrementing (rolling-forward).

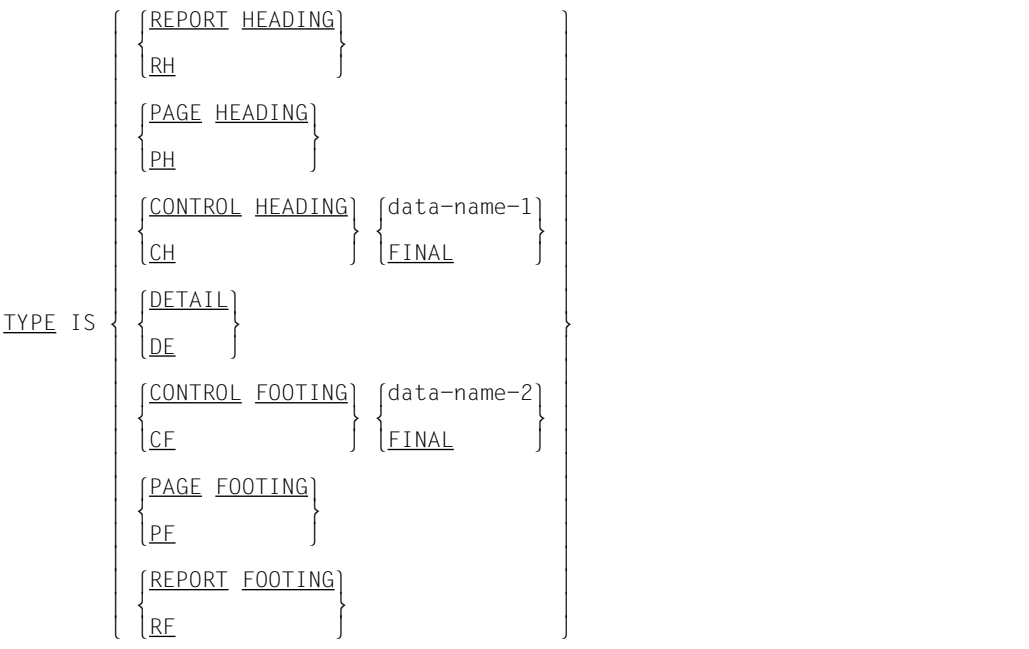
    f) Any sum counters of the control footings whose SUM clauses do not contain RESET phrases are reset to zeros by implicit MOVE statements. The same applies to all those sum counters of the other control footings whose SUM clauses each contain one such RESET phrase which is referring to the control data item associated with the current (i.e. newly-created) control footing.

TYPE clause

Function

The TYPE clause indicates the type of the report group in whose description it appears; that is, it defines the functional characteristics of the report group, thereby also specifying the circumstances under which the Report Writer will generate that report group.

Format



Syntax rules

1. RH is the abbreviation for REPORT HEADING.

   PH is the abbreviation for PAGE HEADING, etc.

2. FINAL, data-name-1 and data-name-2 must be defined in the CONTROL clause of the associated report description (RD) entry.

3. REPORT HEADING indicates the report group which is created only once per report as the first report group in that report. It is created automatically when the first GENERATE statement is executed.

4. PAGE HEADING indicates a report group which is created as the first group on every page. A page which is wholly reserved for the report heading or report footing is not assigned a page heading. If a report heading is present but does not appear on a page by itself, the page heading is generated as the second group on the first page of the report.
5. CONTROL HEADING indicates report groups which are written in series when executing the (chronologically) first GENERATE statement and upon every control break. Each control heading is associated (by FINAL or data-name-1) with one, and only one, hierarchical level, which determines the order in which the control headings are printed (see "CONTROL clause", page 579).
6. DETAIL indicates those report groups which are written because they are supplied in a GENERATE statement. The name of a detail group must be unique throughout the report.
7. CONTROL FOOTING indicates those report groups which are written in series upon every control break and when the TERMINATE statement is executed. Each control footing is associated (by FINAL or data-name-2) with one, and only one, hierarchical level, which determines the order in which the control footings are printed (see "CONTROL clause", page 579).
8. PAGE FOOTING indicates the report group which is generated as the last group of each report page. Any page that is wholly reserved for the report heading or report footing is not assigned a page footing. If the report footing does not appear on a page by itself, the page footing is generated as the last but one group on the last page of the report.
9. REPORT FOOTING indicates a report group that is produced only once, as the last group in the report. The report footing is the last group printed when the TERMINATE statement is executed.

General rules

1. Rules for the report heading
   a) Only one report heading may be defined for each report.
   b) The report heading may be printed on a whole page by itself. The NEXT GROUP clause with the NEXT PAGE phrase specified may be used to make sure that the same page will contain no further group besides the report heading (see "NEXT GROUP clause", page 602).
2. Rules for page heading and page footing
   a) Only one page heading and one page footing may be defined for each report.

b) Normally, the page heading appears as the first report group and the page footing appears as the last report group on every page of the report, with the following exceptions:

– On the first page of the report, the page heading is preceded by the report heading in those cases where the report heading is not to appear on a page by itself.

– The page footing of the last page of the report is followed by the report footing in those cases where the report footing is not to appear on a page by itself.

3. Rules for control headings and control footings

a) Only one control heading and one control footing may be specified for each level of hierarchy.

b) Control headings and control footings are printed at the following times:

– When the first GENERATE statement for a report is executed, the Report Writer prints the entire hierarchy of control heading report groups (in the order highest level first, lowest level last) before printing the detail group as the immediate product of the GENERATE statement.

– If the Report Writer detects a control break when executing one of the (chronologically) next GENERATE statements, it creates the appropriate series of control footings and control headings before the detail groups directly referenced by the current GENERATE statement (see "GENERATE statement", page 622 and "CONTROL clause", page 579).

– When production of a report ends by execution of the TERMINATE statement, the Report Writer generates all control footings in increasing order of hierarchy levels.

– FINAL, data-name-1 or data-name-2 specified in the TYPE clause of a control heading or control footing must appear in the CONTROL clause of the associated report description entry.

– A control heading or control footing associated with FINAL may only be generated once for each report, as the first body group of the report (execution of the first GENERATE statement) or as the last (execution of the TERMINATE statement).

4. Rules for detail report groups

The Report Writer generates a detail report group only when it is specified in a GENERATE statement and when this GENERATE statement is executed. At least one detail report group must be defined for each report. This is true, even if the detail report group is not explicitly used for generating the report; that is, even if it is mainly a summary report without going into details (that is, detail report groups) (see "GENERATE statement", page 622).

5. Rules for the report footing

   a) Only one report footing may be defined for each report, and it is printed as the last report group in the report.

   b) The report footing may appear on a page by itself, by specifying the first LINE clause with the NEXT PAGE phrase in the description of the report footing (see "LINE clause", page 597).

6. Sequence of report groups within a report

   a) No report group of a report may be printed either before the report heading or following the report footing.

   b) The (schematic) sequence of the heading and footing groups for a report is this:

      Report heading (may only appear once)
      Page heading
      ...
      Control heading
      Detail
      Control footing
      ...
      Page footing
      Report footing (may only appear once)

   c) The control headings are always created in immediate succession in hierarchical order:

      Control heading with FINAL (highest level, may only appear once)
      Control heading at next highest level
      ...
      Control heading at lowest level

   d) The control footings are always created in immediate succession in hierarchical order:

      Control heading at lowest level
      Control heading at next lowest level
      ...
      Control footing with FINAL (highest level of hierarchy, may only appear once).

8.3 Language elements of the Procedure Division

GENERATE statement

Function

The GENERATE statement directs the Report Writer to produce a portion of the report in accordance with the report description specified in the REPORT SECTION of the Data Division.

Format

```
GENERATE {data-name  }
         {           }
         {report-name}
```

Syntax rules

1. data-name must be defined in the REPORT SECTION of the Data Division, as the name of a detail report group (01-level entry).
2. The report-name must be defined as such (RD entry) in a report description entry of the REPORT SECTION in the Data Division.
3. As the result of a GENERATE statement referring to a detail report group, part of the report printed (see rule 5).
4. As the result of a GENERATE statement referring to a report-name, part of a report is printed (see rule 8).
5. As the result of a GENERATE statement referring to a detail report group, the Report Writer generates part of the report. The composition of this portion is indicated in syntax rules 6 and 7. In addition, various summations are generally executed (see "SUM clause", page 610).
6. When executing the (chronologically) first GENERATE statement (relative to the execution of the related INITIATE statement), the following report groups (if defined) will be printed, provided this statement specifies a detail report group:
   a) report heading
   b) page heading
   c) all control headings from the highest to the lowest levels of hierarchy, and
   d) the detail report group specified in the GENERATE statement.

7. If a GENERATE statement specifying a detail report group is not executed as the (chronologically) first statement, the Report Writer will first check whether a control break occurred. If so, the following report groups are written in the order stated:

   a) All control footings from the lowest level up to, and including, the level at which the control break occurred, and all control footings from the control break level down to the lowest level of hierarchy.

   b) The detail report group specified by the GENERATE statement.

   Step a) is not applicable unless there was a control break.

8. As a result of a GENERATE statement referring to a report, the Report Writer takes the same actions, except for the creation of a detail report group, which is not applicable here. No additional actions beyond syntax rules 5, 6 and 7 will be taken.

9. When a page becomes full in the course of printing a report, the Report Writer automatically generates a page advance, preceded by the page footing (if present) of the previous page and followed by the page heading of the new page.

General rules

1. At the time of executing a GENERATE statement which specifies a detail report group, the following information must be available to the Report Writer:

   a) All of the SOURCE clause information assigned to the detail report group and all other report groups to be created by the GENERATE statement.

   b) The numeric data of those addends the Report Writer needs to accumulate the necessary sums.

2. Summary report printing is meaningful only for those reports for which control footings, too, are defined whose descriptions include SUM clauses.

3. Summary reporting should not be attempted for reports whose descriptions contain more than one detail report group because it does not allow relations to be established to the individual detail report groups.

4. The CBL-CTR special register (see page 631) is interrogated by the Report Writer at the time of the (chronologically) first GENERATE statement. By using the MOVE statement to supply one of several defined values to this special register any time between the execution of the INITIATE statement and the (chronologically) first GENERATE statement, the programmer can select one or both of the following Report Writer options:

   a) Supply appropriate control data item values to the control footings, the page footing and the page heading.

   b) Condition execution of the NEXT GROUP clauses in the control footings (see "CBL-CTR special register", page 631).

INITIATE statement

Function

The INITIATE statement causes the Report Writer to begin the processing of one or more reports.

Format

```
INITIATE {report-name-1}...
```

Syntax rules

1. report-name-1... must be defined by a report description in the REPORT SECTION of the Data Division (RD entry).
2. The INITIATE statement indicates those reports (report-name-1...) which the Report Writer is to start generating.
3. The INITIATE statement instructs the Report Writer to perform the following initialization functions for each named report:
   – Set all sum counters to zero.
   – Set LINE-COUNTER special register to zero.
   – Set PAGE-COUNTER special register to one.

General rules

1. The INITIATE statement does not open the file the named report is associated with. This report file, which is defined as a sequential output file, must therefore be OPENed as OUTPUT before execution of the INITIATE statement.
2. If an INITIATE statement is followed by a second (different) INITIATE statement specifying the same report as the first one, an intervening TERMINATE statement must first be executed for this report.
3. If an INITIATE and a TERMINATE statement were executed for a given report without an intervening GENERATE statement, the TERMINATE statement cancels the INITIATE statement without printing the report (or any portion thereof).
4. Following execution of the INITIATE statement, and prior to execution of the (chronologically) first GENERATE statement, the programmer may select certain optional functions of the Report Writer by using the MOVE statement to supply the proper value to the CBT-CTR special register (see page 631).

TERMINATE statement

Function

The TERMINATE statement instructs the Report Writer to complete the processing for the specified reports.

Format

```
TERMINATE {report-name-1}...
```

Syntax rules

1. report-name-1 must be defined by a report description in the REPORT SECTION of the Data Division (RD entry).
2. The TERMINATE statement indicates those reports whose processing the Report Writer is to complete.
3. For each report specified in the TERMINATE statement, the Report Writer performs the following steps in the stated order:

   a) All control footings are created as if they were to be printed as the result of a control break at the highest level of hierarchy (obviously, this implies creating the page footing and the page heading if a page advance is required, and the execution of summations).

   b) The page footing is generated.

   c) The report footing is generated.

   However, the above actions are not taken if no GENERATE statement was executed for the report between the INITIATE and TERMINATE statements.

General rules

1. Each TERMINATE statement for report-name-1 must be chronologically preceded by an INITIATE statement for report-name-1.
2. Since the TERMINATE statement does not close the associated report file, it must be chronologically followed by a CLOSE statement. Each INITIATEd report in a particular file must be TERMINATEd before a CLOSE statement is executed for that report file.

USE BEFORE REPORTING statement

Function

The USE BEFORE REPORTING statement introduces Procedure Division statements that are to be executed just before the specified report group is printed by the Report Writer.

Format

```
USE [GLOBAL] BEFORE REPORTING report-group-name.
```

Syntax rules

1. The report-group-name must be defined as a name (data-name) in a 01-level entry of a REPORT SECTION report group entry in the Data Division. Any report group type except the detail report group may be specified in the USE BEFORE PRINTING statement.
2. The report-group-name identifies the report group for which the USE declaratives (USE BEFORE REPORTING procedures) following the USE BEFORE REPORTING statement are to be carried out.
3. The USE BEFORE REPORTING statement itself is never executed; it merely defines the conditions calling for the execution of the subsequently declared USE procedures.
4. The USE procedures declared for a report group are executed immediately before that report group is created.
5. No more than 39 USE BEFORE REPORTING statements may appear in the Procedure Division.
6. Use of the GLOBAL clause is described in chapter 7, "USE statement" (page 568).

General rules

1. The name of a given report group may be specified in one declarative section only.
2. The INITIATE, GENERATE, EXIT PROGRAM, GOBACK, CANCEL and TERMINATE statements must not appear in a paragraph within any declarative section.
3. A USE BEFORE REPORTING procedure must not alter the value of any control data item, nor any value of the subscripts used by the Report Writer to access the control data item.
4. The rules on references between a USE BEFORE REPORTING declarative and the remainder of the Procedure Division are the same as for other USE procedures.

5. The following are typical applications of USE BEFORE REPORTING procedures:

   a) Suppressing the printing of a report group:
   If the statement MOVE 1 TO PRINT-SWITCH is executed in a USE BEFORE REPORTING declarative, the Report Writer will not print the report group the USE procedure was associated with. Since the Report Writer always resets the PRINT-SWITCH special register (see page 630) to zero immediately after suppressing the report group, this register must be set to 1 again each time that printing is to be suppressed; with report groups that are written automatically, this can only be done by means of USE BEFORE REPORTING declaratives.

   b) Modifying the contents of a data item specified in a SOURCE clause:
   In editing a line with a printable item whose description includes a SOURCE clause, the contents of the item defined by the SOURCE clause is transferred to the printable item by means of an implicit MOVE statement. A USE BEFORE REPORTING procedure may be used to modify the contents of the sending field just prior to the execution of the implicit MOVE statement.

   c) Rounding sum counters:
   If the value of a sum counter is to be rounded before it is transferred to the associated print item for editing, by an implicit MOVE statement, this can only be achieved by means of a USE BEFORE REPORTING declarative for the control footing in which the sum counter was defined.

   d) If special actions depending on the level within the control hierarchy at which a control break has occurred are to be taken before writing a page heading, control heading, control footing, or page footing, this can only be effected via USE BEFORE REPORTING declaratives for this report group, owing to their automatic generation (see "SOURCE clause", page 608 and "CBL-CTR special register", page 631).

8.4 Special registers of the Report Writer

LINE-COUNTER special register

LINE-COUNTER is a special register which is automatically defined for each report described in the REPORT SECTION of the Data Division. The internal COBOL notation of this special register is PICTURE S999 USAGE COMPUTATIONAL.

The Report Writer always uses the LINE-COUNTER register to control the vertical spacing of the groups to be printed in a report. For this reason, the LINE-COUNTER register must not be changed by any Procedure Division statement.

As regards the PAGE LIMIT, NEXT GROUP, and LINE clauses, the special register LINE-COUNTER is automatically interrogated and incremented (or set to zero for page advance) by the Report Writer. The INITIATE statement resets the LINE-COUNTER register to zero.

The LINE-COUNTER may be used both in the REPORT SECTION and in the Procedure Division. As far as the REPORT SECTION is concerned, LINE-COUNTER may only be specified in SOURCE clauses as indicated below:

```
SOURCE IS LINE-COUNTER [OF report-name]
```

As the Report Writer always sets the LINE-COUNTER register to the current line (print line or NEXT GROUP positioning), the line number will be displayed on which the printable item associated with the above SOURCE clause is to be written. The LINE-COUNTER special register may be interrogated at any time in the Procedure Division; at the time of interrogation, it will contain the value assigned to it by the Report Writer as a result of the NEXT GROUP clause of the last report group generated before the interrogation (last print line of the report group if no NEXT GROUP clause was specified).

If two or more reports are described in the REPORT SECTION, each explicit reference to a LINE-COUNTER must be qualified by the report-name.

PAGE-COUNTER special register

PAGE-COUNTER is a special register that is defined automatically for each report described in the REPORT SECTION of the Data Division. The internal COBOL notation of this special register is PICTURE S9(7) USAGE COMPUTATIONAL-3.

When the INITIATE statement is executed for a report, the Report Writer increments the PAGE-COUNTER special register to 1 by means of an internal MOVE statement. As soon as the Report Writer issues a page advance - after printing the page footing and before printing the page heading - it increments the current contents of the special register PAGE-COUNTER by 1.

The PAGE-COUNTER register is freely accessible. That is, its contents may be modified as well as interrogated. The most common use of PAGE-COUNTER is for printing page numbers in page headings or page footings. For this purpose, the PAGE-COUNTER register is assigned to a printable item of the associated report, by the following SOURCE clause:

```
SOURCE IS PAGE-COUNTER [OF report-name]
```

Program references to PAGE-COUNTER must be qualified by the report name if the REPORT SECTION describes more than one report.

PRINT-SWITCH special register

PRINT-SWITCH is a special register that is defined automatically for a program whose Data Division includes the REPORT SECTION. The internal COBOL notation of this register is PICTURE S9 USAGE COMPUTATIONAL-3. Only one such special register is defined for each source program.

The purpose of PRINT-SWITCH is to enable the program to suppress the printing of a report group. This is achieved by including the statement MOVE 1 TO PRINT-SWITCH within a USE BEFORE REPORTING procedure for the appropriate report group. The Report Writer checks the contents of PRINT-SWITCH immediately after each execution of a USE BEFORE REPORTING declarative associated with a report group, and suppresses printing if PRINT-SWITCH contains a 1 . After evaluating PRINT-SWITCH, the Report Writer restores its value to 0, thereby preventing any inadvertent suppression of other report groups.

Report group suppression by the Report Writer has the following effects:

– No page spacing as specified by the LINE clauses of the report group.

– No editing of print lines.

– No page spacing as specified by any NEXT GROUP clause of the report group.

The suppression of a report group owing to the PRINT-SWITCH special register implies, in consequence of the above items a) and c), that the LINE-COUNTER special register is not changed.

PRINT-SWITCH should be set only within USE BEFORE REPORTING declaratives. If it is set anywhere else in the Procedure Division, it is generally impossible to predict which report group might be suppressed.

CBL-CTR special register

CBL-CTR (**C**ontrol **B**reak **L**evel **C**ounter) is a special register that is defined automatically for each report described in the REPORT SECTION of the Data Division. The internal COBOL notation of this special register is PICTURE S999 USAGE COMPUTATIONAL.

The Report Writer and the program use this special register to pass information to each other concerning control breaks and the action to be taken when control breaks occur.

CBL-CTR can be used in connection with two different functions. Either, neither, or both of these functions may be used for a given report. For the sake of simplicity, the two functions are known as "function 1" and "function 2". The user program notifies the Report Writer which function it requires for a report. By a MOVE statement, the CBL-CTR register is assigned a value that symbolizes the requested function. Value and function interact as shown below:

Function requested	**MOVE statement**
Function 1	MOVE 1 TO CBL-CTR.
Function 2	MOVE 2 TO CBL-CTR.
Function 1 and 2	MOVE 3 TO CBL-CTR.

The corresponding MOVE statement must be executed after the INITIATE statement for the report but before the (chronologically) first GENERATE statement. This is the only time that the programmer is permitted to change the value of the CBL-CTR special register.

If two or more reports are defined in the REPORT SECTION, each reference to CBL-CTR must be qualified by the report-name.

Function 1 of the CBL-CTR special register

This function is invoked by a MOVE statement which places 1 (or 3) in the CBL-CTR special register.
The function consists of two parts; the respective actions are covered in the following rules.

a) Restoration of the previous values of the control data items in control footings.

Part 1 of function 1 makes previous values of control data items available to control footing SOURCE clauses or control footing USE BEFORE REPORTING declaratives.
SOURCE clauses (or USE BEFORE REPORTING declaratives) in control footings referring to control data items produce a problem. At the time the control footings are printed, some or all of the specified control data items have changed values.

However, since the control footings printed because of a control break obviously belong to the precontrol break values, it is often desirable that the previous values (i.e. prior to the control break) should be printed. When function 1 of CBL-CTR is requested, these previous values of the control data items will be obtained by SOURCE clauses (or USE BEFORE REPORTING declaratives) of control footings.

For example, assume that the item MONTH-NAME is defined as a control data item for a report and that the control footing MONTH-FOOTING is defined as follows:

```
01 MONTH-FOOTING TYPE CONTROL FOOTING
                        MONTH-NAME LINE PLUS 1.
   02 COLUMN 10 PIC X(21) VALUE "***** END OF DATA FOR".
   02 COLUMN 33 PIC X(9)  SOURCE MONTH-NAME.
```

In this case, the programmer wants the following control footing to be printed after all of the JANUARY data has been printed in the detail lines of the report:

```
***** END OF DATA FOR JANUARY.
```

Since the above control footing was printed because the item MONTH-NAME changed from JANUARY to FEBRUARY, FEBRUARY (the current contents) would be printed rather than JANUARY. By requesting function 1, the programmer can cause the prior value (JANUARY) to be printed instead of FEBRUARY.

b) Indicating the control break level in CBL-CTR

Part 2 of function 1 causes the Report Writer to place, in CBL-CTR, a value indicating the control break level in the hierarchy. This is done before control is passed to a USE BEFORE REPORTING procedure for execution. This kind of procedure begins with:

```
section-name SECTION. USE BEFORE REPORTING report-group-name.
```

Indicating the level of the current control break in the hierarchy is accomplished by numbering the control data items consecutively, starting from the highest level in the hierarchy, not including FINAL. Thus, the first (leftmost) data-name of the CONTROL clause is numbered 1, the next is numbered 2, and so on.

For instance, assume that a report description entry contains this CONTROL clause:

```
CONTROLS ARE FINAL STATE COUNTY CITY
```

In this case, STATE is assigned number 1, COUNTY number 2, and CITY number 3.

The meanings of the CBL-CTR values in the USE procedures declared for the page heading and control headings are listed below in Table 8-4.

If, in the above example, the value of the control data item COUNTY changes, the value 2 will be placed in the CBL-CTR register by the Report Writer (provided that the value of CITY has not changed at the same time also).

Value	Meaning
0	Indicates that no GENERATE statement has been executed or that the very first GENERATE statement is being executed.
1-254	Indicates that the control data item with the corresponding number has just changed values, and that actions currently taking place were caused by this control break.
255	Indicates that no control break has occurred (cannot be encountered with control footing USE procedures).

Table 8-4: CBL-CTR values in USE procedures for page headings and control headings

The meanings of CBL-CTR values in USE declaratives for the page footing and control footings are listed below in Table 8-5.

Value	Meaning
0	Indicates the final stage of processing, i.e. the TERMINATE statement is being executed.
1-254	Indicates that the control data item with the corresponding number has just changed values, and that actions currently taking place were caused by this control break.
255	Indicates that no control break has occurred (cannot be encountered with control footing USE procedures).

Table 8-5: CBL-CTR values in USE procedures for page footings and control listings

Function 2 of the CBL-CTR special register

Function 2 of CBL-CTR is invoked by moving a value 2 (or 3) to CBL- CTR. This causes conditional execution of the NEXT GROUP clauses in the control footings.

Normally, the NEXT GROUP clause is executed immediately after the creation of the report group in whose description it is supplied (vertical spacing). If a control break causes several control footings to be printed in succession, the NEXT GROUP clauses of these control footings lead to inadvertent blank lines. Such blank lines are really intended for spacing between a control footing and a control heading or a detail report group but not to another control footing.

When function 2 of CBL-CTR is requested, the Report Writer makes sure that only the last NEXT GROUP clause is executed, i.e. the NEXT GROUP clause associated with the last control footing generated in the series of control footings printed in succession as a result of a control break. Existing NEXT GROUP clauses of the other control footings of a closed sequence of control footings will be ignored. This is true even if the last control footing of the sequence has no NEXT GROUP clause.

9 Segmentation

9.1 General description

The Segmentation feature allows the programmer, at compile time, to specify program overlay requirements. Only the Procedure Division may be segmented. The Procedure and Environment Divisions are therefore provided to define the segmentation requirements for the program.

Note

Segmentation is superfluous when generating shareable code. Shareable code can be generated simply by means of a control statement to the compiler (see "COBOL85 User Guide" [1]).

9.2 Organization

Although it is not mandatory, the Procedure Division for a source program is usually written as several consecutive sections, each of which is composed of a series of statements which, taken together, perform a particular function. When segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion of the program or to one of the independent segments of the program. On the other hand, segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

9.3 Fixed portion of the program

The fixed portion of the program is defined as that portion that is logically treated as if it were always resident in memory. This portion of the program is composed of two types of computer storage segments: permanent segments and fixed overlayable segments.

a) A permanent segment is a segment in the fixed portion that cannot be overlaid by another part of the program.

b) A fixed overlayable segment is a segment in the fixed portion which can overlay, or be overlaid by, either another fixed overlayable segment or an independent segment. From the execution point of view, however, it is considered to be resident in internal storage. When a fixed overlayable segment is called, it is always made available in the state in which it was last used; GO TO statement branch addresses modified by ALTER are not returned to their initial states.

The number of the permanent segments in the fixed portion can be varied through the use of the SEGMENT-LIMIT clause.

9.4 Independent segments

An independent segment is defined as that part of the program that can overlay other segments and can be overlaid by an overlayable fixed segment or by another independent segment. If procedures contained within an independent segment are referenced by a PERFORM or GO TO statement from outside that independent segment, the program is always provided with an independent segment which is in its initial state. GO TO statement branch addresses that have been modified by ALTER are returned to their initial state.

9.5 General rules for segmentation

1. If the sections that belong to a given segment, i.e. sections which have the same segment-number, are scattered through the source program, they must be reordered by the compiler. However, the compiler will maintain the logic flow of the source program. The compiler will also insert statements necessary to load and/or initialize a segment as necessary. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.
2. Only the fixed segments remain in internal storage throughout program execution. Both the overlayable fixed segments and the independent segments may overlay one another.
3. The following criteria should be noted when classifying segments:

   Logical requirements:

   Sections that must be available for reference at all times or which are referred to frequently should be classified as one of the permanent segments; sections that are less frequently used should be classified as one of the overlayable fixed segments or one of the independent segments, in accordance with the logic requirements of the program flow.

   Relationship to other sections:

   Sections that frequently communicate with one another should be given equal segment-numbers. All sections with the same segment-number each constitute a program segment.
4. When segmentation is used, the following rules apply to the ALTER, PERFORM, MERGE and SORT statements as well as the called programs:

   ALTER statement:

   a) A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referenced by an ALTER statement in a section with a different segment-number.

   b) A GO TO statement in a section whose segment-number is less than 50 may be referenced by an ALTER statement in any section, even if the GO TO statement referenced by the ALTER statement belongs to a program segment which has not yet been called for execution.

PERFORM statement:

a) A PERFORM statement that appears in a section whose segment-number is less than the segment-number supplied in the SEGMENT-LIMIT clause can have within its range only the following:

   – Sections each having a segment-number less than 50.

   – Sections wholly contained in a single segment whose segment number is greater than 49.
   However, the compiler permits the PERFORM statement to reference, within its range, sections having any section number.

a) A PERFORM statement appearing in a section whose segment-number is equal to or greater than the segment number specified in the SEGMENT-LIMIT clause can have within its range only one of the following:

   – Sections each having the same segment-number as that of the section containing the PERFORM statement.

   – Sections each having a segment-number less than that specified in the SEGMENT-LIMIT clause.

However, the compiler permits the PERFORM statement to reference, within its range, sections having any section number.

SORT/MERGE statement:

a) When using a SORT or MERGE statement within a section which is not an independent segment, the input procedures or output procedures referenced by that SORT or MERGE statement:

   – must be contained wholly within independent segments,

   – or must be contained wholly within a single independent segment.

b) When using a SORT or MERGE statement within an independent segment, the input procedures or output procedures referenced by that SORT or MERGE statement:

   – must be wholly contained within independent segments, or

   – must be wholly contained within the same independent segment as the SORT or MERGE statement.

These restrictions do not apply to the compiler discussed in this manual.

Called programs

A program that was called by a CALL statement may have its entry points only in the permanent segment.

9.6 Language elements

There are two COBOL language elements which are directly related to segmentation. Both language elements are optional and should only be specified when segmentation is used:

SEGMENT-LIMIT clause	specified in the OBJECT-COMPUTER paragraph of the Environment Division.
Segment number	specified in the appropriate section header in the Procedure Division.

9.6.1 Language elements of the Environment Division

SEGMENT-LIMIT clause

Function

The SEGMENT-LIMIT clause enables the user to vary the number of permanent and overlayable fixed segments in his program, while still retaining the logical properties of fixed portion segments (segment-number 0 through 49) and keeping constant the total number of segments in the fixed portion.

Format

```
SEGMENT-LIMIT IS segment-number
```

Syntax rules

1. The SEGMENT-LIMIT clause is supplied in the OBJECT-COMPUTER paragraph. This is an optional clause.
2. segment-number must be an unsigned integer that ranges in value from 1 through 50.
3. When the SEGMENT-LIMIT clause is specified, only those segments having segment-numbers up to, but not including, the segment-number designated in the SEGMENT-LIMIT clause are considered as permanent.
4. Those segments having segment-numbers from the segment limit through 49 are considered to be overlayable fixed segments.
5. When the SEGMENT-LIMIT clause is omitted, all segments having segment numbers from 0 through 49 are considered as permanent segments of the program.

General rule

Ideally, all program segments having segment-numbers ranging from 0 through 49 are treated as permanent segments. However, when insufficient storage is available to contain all permanent segments plus the largest overlayable segment, the number of permanent segments must be reduced. This may be done by subsequently including a SEGMENT-LIMIT clause with appropriate entries without otherwise changing the program.

Example 9-1

```
SEGMENT-LIMIT IS 40
```

This clause indicates that all segments having segment-numbers from 0 through 39 are considered to be permanent segments of the program. Segments having segment-numbers from 40 through 49 are overlayable fixed segments.

9.6.2 Language elements of the Procedure Division

Segment number

Function

Section classification is achieved by a system of segment-numbers. The segment-number is included in the section heading.

Format

```
section-name SECTION [segment-number].
```

Syntax rules

1. Section-name identifies the section.
2. Segment-number indicates whether the chapter belongs to a permanent, overlayable fixed, or independent segment.
3. segment-number must be an unsigned integer ranging in value from 0 through 99.
4. All sections that have the same segment-number constitute a program segment with that number.
5. Segments with segment-numbers 0 through 49 belong to the fixed portion of the object program. The SEGMENT-LIMIT clause indicates which of these segments are treated as permanent and which as overlayable fixed segments.
6. Segments with segment-numbers 50 through 99 are independent segments.
7. If the segment-number is omitted from the section heading the segment number is assumed to be 0.
8. Sections in the DECLARATIVES subdivision of the Procedure Division must not contain segment-numbers in their section headings. They are treated as permanent segments whose segment-number is 0.

General rule

When a procedure-name in an independent segment is referred to by a PERFORM or GO TO statement contained in a segment with a different segment-number, the segment referred to is made available in its initial state for each execution of the PERFORM or GO TO statement. If the PERFORM statement is repeated, the initial state is restored only for the first execution of the PERFORM statement.

10 Sorting of records

Records can be sorted in two different ways:

– sorting of records contained in a file (file sorting)
– sorting of records contained as elements in a table (table sorting).

Both sorting methods use the BS2000 utility SORT for sort processing.

File sorting is described in section 10.1, table sorting in section 10.2.

10.1 Sorting and merging of files

10.1.1 Sort processing

The user has the capability of sorting a file on the basis of a series of keys. These sort keys are specified by the user and are present in each record of the file. The records may be sorted so that all of their keys are in ascending or in descending order, or so that some of their keys are ascending and others are descending.

Records are sorted on a file called a sort-file. This file is defined in the Data Division in a sort-file description (SD) entry, and in the record description associated with the SD. The SORT statement in the Procedure Division initiates the sort operation.

Sort processing consists of:

1. Releasing all records to the sort-file.
2. Sorting the records on the sort-file.
3. Returning all records from the sort-file.

The user may either provide an input procedure to process records and transfer them to the sort-file, or he may specify an input file containing the records to be sorted and allow the SORT statement to transfer these records to the sort-file. Accordingly, he may either provide an output procedure to retrieve records from the sort-file and process them further, or he may specify an output file and allow the SORT statement to output the sorted records to this file.

Several sort operations may be specified in a single program.

10.1.2 Merge processing

The user has the capability of merging from two to 16 sorted input files with the same record format and transferring them to an output file.

Merging takes place in a file called a sort-file. This file is defined in the Data Division within a sort-file description (SD) entry, and in the record description associated with the SD. Merge processing is initiated by means of the MERGE statement in the Procedure Division. The merge operation is performed in three stages:

1. Release the records of all input files to the sort-file.
2. Merge the records in the sort-file.
3. Return all records from the sort-file.

The MERGE statement releases the records of all specified input files to the sort-file. The user can specify an output file into which the MERGE statement returns the merged records. In connection with returning, it is also possible to specify an output procedure. This procedure accepts records from the sort-file and continues processing them.
Several merge operations may be specified in a single program.

10.1.3 Sort and merge without input/output procedures

If no input or output procedures are specified, the compiler will generate procedures to take charge of record input/output. In this case the user must describe the following files:

1. One or more input files containing the records to be sorted or merged,
2. One sort-file and one or more output files to which the records will be returned.

When referenced, a SORT or MERGE statement performs the following functions:

1. Opens the input and sort-files.
2. Releases all the records in the input file to the sort-file.
3. Closes input files.
4. Sorts or merges the records on the sort-file.
5. Opens output files.
6. Returns the records from the sort-file to the output file.
7. Closes the sort, and output files.

10.1.4 Sort with input/output procedures

If input and output procedures are specified, the following actions are taken:

1. When a SORT statement is executed, it transfers control to the input procedure.
2. The input procedure performs the following functions:
   a) Processes a record (for example, reads a record from a file or creates a new record).
   b) Releases the record to the sort-file.
   c) Repeats steps a) and b) until all records have been released.
3. The SORT statement sorts the records on the sort-file.
4. Control is then passed to the output procedure.
5. The output procedure performs the following functions:
   a) Accepts a record from the sort-file.
   b) Processes the record (for example, writes the record to an output file).
   c) Repeats steps a) and b) until all records have been returned and processed.
6. Control is returned to the statement following the SORT statement.

If options are mixed (e.g. if only an input procedure is specified), appropriate variants of the above procedures are performed.

10.1.5 Overview of language elements

To use the sort feature, the user must provide additional information in the Environment, Data, and Procedure Divisions of the source program. This information is summarized in the following table and is described in detail in the remainder of this section.

Source program division	Contents and meaning
Environment Division	A SELECT clause in the FILE-CONTROL paragraph forthe sort-file (sort-file-name).
	SELECT clauses in the FILE-CONTROL paragraph for all files used as input and output procedures for the sort-file (section-name-1, section-name-2, section-name-3, section-name-4 for SORT; section-name-1, section-name-2 for MERGE) and for files appearing in the USING or GIVING phrase of a SORT or MERGE statement (file-name-1,...).
	To permit restart of programs containing the SORT statement, the RERUN clause may be used in the I-O-CONTROL paragraph.
	The SAME SORT AREA or the SAME SORT-MERGE AREA clause in the I-O-CONTROL paragraph is intended to optimize the allocation of internal storage to a sort-file.
Data Division	A sort-file description (SD) entry and associated record description entries for the sort-file; record descriptions must include sort-key fields.
	File description (FD) entries and associated record description entries for all files used in input/output procedures for the sort-file section-name-1, section-name-2, section-name-3, section-name-4 for SORT; section-name-1, section-name-2 for MERGE) and for files appearing in the USING or GIVING phrase of a SORT or MERGE statement (file-name-1,...).
	Data description entries for the above files.
Procedure Division	A SORT or MERGE statement for the sort-file specifying the following information: – Sort-key names – Whether the sort is to be in ascending or descending order of key. – Input/output information specified by the following options:

Source program division	**Contents and meaning**	
Procedure Division	Specification	Meaning
	INPUT PROCEDURE	Records will be released to the sort-file by an input procedure.
	USING file-name-1,...	Specifies files from which the SORT statement will accept records to be released to the sort-file.
	OUTPUT PROCEDURE	Records will be returned from the sort-file by an output procedure.
	GIVING file-name-3,...	Specifies a file to which the SORT and MERGE statements will return sorted and merged records, respectively.
	If input and/or output procedures are specified in the SORT or MERGE statement, these procedures must be included in the Procedure Division. An input procedure must include a RELEASE statement, to transfer records to the sort-file. An output procedure must include a RETURN statement, to accept records from the sort-file.	
	Special sort-registers can be referenced in the Procedure Division (see "Special registers for file SORT", page 670).	

10.1.6 Language elements of the Environment Division

RERUN clause

Function

The RERUN clause in the I-O CONTROL paragraph makes it possible for programs containing a SORT or MERGE statement to be restarted.

Format

```
RERUN ON { file-name-1       } EVERY SORT [OF file-name-2...]
         { implementor-name  }
```

Syntax rules

1. file-name-2... must be defined in an FD entry.
2. file-name-1 must not be defined in an FD entry.
3. implementor-name must not appear in a SELECT clause.

General rules

1. When the OF phrase is not specified, the checkpoint is written prior to each sort operation.
2. When file-name-2 is specified, checkpoints are written for this specific sort operation only.

SAME AREA clause

Function

The SAME AREA clause in the I-O CONTROL paragraph indicates that two or more files are to share a specified internal storage area during program execution.

Format

```
SAME { RECORD     } AREA FOR file-name-1 {file-name-2}...
     { SORT       }
     { SORT-MERGE }
```

Syntax rules

1. SORT and SORT-MERGE are equivalent.
2. If SAME SORT AREA or SAME SORT-MERGE AREA is used, at least one of the file-names must refer to a sort or merge file. Files which are not sort or merge files may also be specified in the clause.
3. More than one SAME AREA clause may appear in a given program, but the following restrictions must be observed:
   a) A given file-name must not occur in more than one SAME RECORD AREA clause.
   b) A file-name representing a sort-file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.
   c) If a file-name which does not represent a sort-file appears in a SAME AREA clause and in one (or more) SAME SORT AREA or SAME SORT-MERGE AREA clause(s), then all files specified in the SAME AREA clause must also be supplied in the SAME SORT AREA or SAME SORT-MERGE AREA clause.
4. The files that appear in the SAME SORT AREA, SAME SORT-MERGE AREA or SAME RECORD AREA clause need not all have the same organization or access mode.

General rule

The SAME RECORD AREA clause indicates that two or more files should share the storage area in which the current logical record is being processed. All of the files can be open at the same time. A logical record in the "same record area" is considered as a logical record of each opened output file whose file-name occurs in the SAME RECORD AREA clause; it is also considered as a logical record of the last read input

file whose file-name occurs in the SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the internal storage area, where the records are aligned on the leftmost character position.

SELECT clause

Function

The FILE-CONTROL paragraph is specified only once in a COBOL program; within this paragraph, one SELECT clause must be provided for each file referenced in the program.

Files used in input and output procedures, as well as files occurring in the USING or GIVING phrases of a SORT or MERGE statement, are specified in a SELECT clause, as described in the Environment Division.

Format

```
SELECT sort-file-name ASSIGN TO {implementor-name}
                                {literal          }
```

Syntax rules

1. sort-file-name denotes a sort-file.
2. implementor-name is DISC.
3. literal is SORTWK.

General rules

1. This format is applicable only to a SELECT clause referring to a sort-file description entry. No further clauses other than the ASSIGN clause are permitted.
2. Each sort-file described in the Data Division must be designated once, and only once, as a file-name in the FILE-CONTROL paragraph.
3. Each sort-file specified in the FILE-CONTROL paragraph must have a sort-file description entry in the Data Division.
4. The following file-names must not be used in SELECT clauses within a program that uses SORT and runs under BS2000:

```
MERGExx(xx = 01,...,99)
SORTIN
SORTINxx(xx = 01,...,99)
SORTOUT
SORTWK
SORTWKx(x = 1,...,9)
SORTWKxx(xx = 01,...,99)
SORTCKPT
```

10.1.7 Language elements of the Data Division

Sort-file description

Function

A sort-file description (SD) entry provides information concerning the physical structure, identification, and size of the records on a sort-file.

Format

```
SD sort-file-name

   [RECORDING MODE IS mode]

   [       {RECORDS ARE} {OMITTED  }]
   [LABEL {            } {         }]
   [       {RECORD IS  } {STANDARD}]

   [       {RECORD IS  }                 ]
   [DATA   {           } {data-name-1}...]
   [       {RECORDS ARE}                 ]

   [       {CONTAINS integer-1 CHARACTERS                                  }]
   [       {IS VARYING IN SIZE [[FROM integer-2] [TO integer-3] CHARACTERS]}]
   [RECORD {    [DEPENDING ON data-name-2]                                 }]
   [       {                                                               }]
   [       {CONTAINS integer-4 TO integer-5 CHARACTERS                     }]
```

Syntax rules

1. The level indicator SD identifies the beginning of the sort-file description entry and must precede the file-name.
2. The clauses following the name of the file are optional and may appear in any order.
3. One or more record description entries for data-name-1,... must follow the sort-file description entry.

General rules

1. sort-file-name indicates the sort-file.
2. data-name-1... in the DATA RECORDS clause refer to records described in the record description entries associated with this sort-file description (SD) entry.

3. The FILE SECTION must contain a sort-file description entry for each sort-file, i.e. for each file that is supplied as the first operand within a SORT or MERGE statement.
4. The RECORDING MODE clause specifies the organization format of data on external devices.
5. The LABEL RECORDS clause is optional; however, if it is not specified LABEL RECORDS ARE OMITTED is assumed. This means that existing labels will be overwritten.
6. When the LABEL RECORDS ARE STANDARD clause is specified, the work tapes must have standard labels for sorting purposes; however, label handling is not performed. In this case, the labels are retained intact.
7. The DATA RECORDS clause specifies the names of the records on the file to be sorted.
8. If more than one data-name is present, the file contains two or more types of records. These records may differ in length, format etc. They may be listed in any order.
9. If more than one record description is specified for the logical record of a file, these records automatically occupy the same internal storage area; this is equivalent to an implicit redefinition of the area.
10. The RECORDING MODE, DATA RECORDS and RECORD CONTAINS clauses are optional, as the compiler can determine the modes, names and sizes of the records from the associated record descriptions (complete details on these clauses are given in chapter 4 under "Data Division").
11. If any of the record descriptions associated with the sort-file description contains an OCCURS clause with the DEPENDING ON phrase, variable-length records are assumed (for further information on assumptions made by the compiler when the RECORDING MODE clause is omitted, see "RECORDING MODE clause").
12. Sort-file-names may only be used in the SORT, MERGE and RETURN statements..

10.1.8 Language elements of the Procedure Division

MERGE statement

Function

The MERGE statement creates a sort-file into which records are accepted from two or more similarly sorted input files. It merges the records in the sort-file on the basis of a set of specified data items (keys) and, once this merge operation is finished, makes each record from the sort-file available to an output procedure or to output files.

Format

```
MERGE sort-file-name

     { ON { DESCENDING } { KEY    } {data-name-1}... }...
          { ASCENDING  } { KEY-YY }

      [COLLATING SEQUENCE IS alphabet-name]

      USING {file-name-1}...

      { OUTPUT PROCEDURE IS section-name-1 [{ THRU     } section-name-2] }
      {                                    { THROUGH  }                 }
      { GIVING {file-name-2}...                                         }
```

Syntax rules

1. sort-file-name must be defined in a sort-file description (SD) entry in the Data Division.
2. sort-file-name must correspond to the sort-file-name defined in the SELECT clause (format 2).
3. The file names specified in the USING phrase must not be specified in the GIVING phrase.
4. data-name-1... are key data-names. A key is that part of a record which is used as a basis for sorting. Key data-names must be defined in a record description belonging to an SD description entry. They are subject to the following rules:
   a) The data items identified by key data-names must not be of variable length.
   b) The data-names describing the keys may be qualified (see "Qualification", page 77).

c) When two or more record descriptions are supplied, the keys need only be described in one of these descriptions. If a key is defined in more than one record description, the descriptions of that key must be identical, and must ensure that the key appears in the same position within each record.

d) A key must not be defined with an OCCURS clause and must not be subordinate to a data item defined with an OCCURS clause.

e) If the sort-file contains variable length records, all the data items identified by key data-names must be contained within the first n character positions of the record, where n equals the minimum record size specified for the sort-file.

f) A maximum of 64 keys may be specified for any file.

g) Each key must lie within the first 4096 bytes.

h) Keys are always listed from left to right in the order of their decreasing significance, regardless of whether they are ascending or descending. Hence, the first occurrence of data-name-1 would be the principal sort-key and the second occurrence of data-name-1 the subsidiary key.

i) A key, when expressed as a packed decimal, may have no more than 16 digits.

j) The keys following the KEY-YY specification must be defined either with PIC 99 USAGE DISPLAY or USAGE PACKED-DECIMAL (see section 10.3).

5. section-name-1 identifies the first or only section in the output procedure. section-name-2 identifies the last section in the output procedure. It is required only if the output procedure consists of more than one section.

6. file-name-1..., file-name-2... must be defined in a file description (FD) entry in the Data Division.

7. The size of the records that can be passed to a MERGE operation or written to an output file is dependent on the record format of the sort-file (variable or fixed) and will be discussed in more detail in the "General rules" when the USING/GIVING phrases are mentioned.

8. At least one ASCENDING/DESCENDING phrase must be specified in a MERGE statement.

9. The MERGE statement may be written anywhere in the program except

   a) in the declaratives area,

   b) in an input/output procedure belonging to a SORT/MERGE statement.

10. The sort- and input-files named in the MERGE statement must not be specified together in the same SAME AREA, SAME SORT AREA or SAME SORT-MERGE AREA (see "SAME AREA clause", page 650).

11. If file-name-2 references an indexed file, the first specification of data-name-1 must be associated with an ASCENDING phrase and the data item referenced by that data-name-1 must occupy the same character positions in its record as the record key occupies within the file referenced by file-name-2.

General rules

1. The collating sequence is set by means of the ASCENDING/DESCENDING option:
   a) When ASCENDING is specified, sorting proceeds from the lowest to the highest value of the key, i.e. in ascending order.
   b) When DESCENDING is specified, sorting proceeds from the highest to the lowest value of the key, i.e. in descending order.

   The collating sequence is governed by the same rules as apply to the comparison of operands in a relation condition (see page 223).
2. When, according to the rules for the comparison of operands in a relation condition, the contents of all the key data items of one record are equal to the contents of the corresponding key data items of one or more other records, the order of return of these records:
   a) Follows the order of the associated input files as specified in the MERGE statement.
   b) Is such that all records associated with one input file are returned prior to the return of records from another input file.
3. When the program is executed, the collating sequence for the comparison of non-numeric sort items is set as follows:
   a) If COLLATING SEQUENCE has been specified in the MERGE statement, this entry is used as a sort criterion.
   b) If COLLATING SEQUENCE was not specified in the MERGE statement, the program-specific collating sequence will be used (see "OBJECT-COMPUTER paragraph", page 124).
4. All records from the input files (file-name-1...) are transferred to the sort-file designated by sort-file-name. At the start of execution of the MERGE statement, the input files must not be in the open mode. The MERGE statement is executed for each of the referenced files in the following way:
   a) The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed.
   b) The logical records are obtained and released to the merge operation. Each record is obtained as if a READ statement with the NEXT and the AT END phrases had been executed. If the input file contains the RECORD clause with the DEPENDING phrase, the associated DEPENDING ON data item will not be provided for this

READ operation. Relative files must be described in the FILE CONTROL paragraph with ACCESS MODE IS SEQUENTIAL.
If the sort-file contains variable length records, the size which the record had when it was input is used as the length for a record when it is transferred to a MERGE operation. This length must be within the range defined for the sort-file in the RECORD clause (see "RECORD clause", page 390). If the sort-file has a fixed length record format, records shorter than the specified format length will be supplied with blanks, longer records will not be permitted.

c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed.

5. OUTPUT PROCEDURE indicates that the Procedure Division contains an output procedure to process records after they have been merged. If OUTPUT PROCEDURE is specified, control passes to it after the sort-file has been processed by the MERGE statement. During RETURN statement processing, the output procedure accepts the records from the sort-file. The compiler inserts a return mechanism at the end of the last section of the output procedure. When control passes to the last statement in the output procedure, the return mechanism provides for termination of the sort, and then passes control to the statement following the MERGE statement.

   The following rules apply to the output procedure, which is a self-contained section within the Procedure Division:

   a) It must consist of one or more sections that are written consecutively.

   b) It must contain at least one RETURN statement, to make merged records available for processing.

   c) It must not lead to execution of a MERGE, RELEASE, or SORT statement.

   d) It may include any procedures needed to select, modify, or copy records.

   e) A branch may be made from the output procedure if the programmer makes sure that a transfer of this type is followed by a return to the output procedure in order to effect a proper exit from this procedure (i.e. to processing its last statement).

   f) A branch may be made from points outside an output procedure to procedure-names within that procedure if the branch does not involve a RETURN statement or the end of the output procedure.

6. When the OUTPUT PROCEDURE phrase is used, control is passed from the specified procedure as though a format-1 PERFORM statement is executing. That is, all sections constituting the procedure are executed once, and execution of the procedure is terminated after its last statement has been processed. Thus, any procedure may be terminated by using an EXIT statement.

7. If MERGE statements are supplied in segmented programs, the following restrictions apply:

   a) If a MERGE statement appears in a section which is outside any independent segment, then all input or output procedures referred to by that MERGE statement must be either wholly contained within one fixed segment, or wholly contained in a single independent segment.

   b) If a MERGE statement appears in an independent segment, then all input or output procedures referred to by that MERGE statement must be either wholly contained within one fixed segment, or wholly contained in the same independent segment as the MERGE statement.

   These restrictions do not apply to the compiler discussed in this manual.

8. If the GIVING phrase is specified, all the merged records are written on the output file (file-name-3...). At the start of execution of the MERGE statement, the output file must not be in the open mode. The MERGE statement is executed for each of the referenced files in the following way:

   a) The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed.

   b) The merged logical records are returned and written onto the output file as if a WRITE statement without any optional phrases had been executed. The length of these records must be within the range defined for the output file (see RECORD clause, page 390).
   For a relative file, the relative key data item for the first record returned contains the value ’1’; for the second record returned, the value ’2’, etc. After execution of the MERGE statement, the content of the relative data item indicates the last record returned to the file. The file must be defined in the FILE-CONTROL paragraph with ACCESS MODE IS SEQUENTIAL.

   c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. If the output file contains the RECORD clause with DEPENDING ON phrase, the associated DEPENDING ON data item is not evaluated when a record is written. The current record length is used instead.

RELEASE statement

Function

The RELEASE statement transfers records from the input file to the sort-file. It can only be used during a sort operation.

Format

```
RELEASE sort-record-name [FROM identifier]
```

Syntax rules

1. sort-record-name must be the name of a logical record in the associated sort-file description.
2. sort-record-name and identifier must not refer to the same internal storage area.

General rules

1. Execution of a RELEASE statement has the effect that the record specified by sort-record-name is transferred to the sort-file, to be further processed by the SORT routine of the system.
2. The FROM phrase makes the RELEASE statement equivalent to a MOVE statement followed by a RELEASE statement.

   When this phrase is written, data is moved from the area specified by identifier to the area identified by sort-record-name, and is then released to the sort-file. The move takes place according to the rules which govern the MOVE statement without the CORRESPONDING phrase.
3. A RELEASE statement may be used only within an input procedure in connection with a SORT statement for the sort file containing sort-record-name.
4. After a RELEASE statement is executed, the logical record is no longer available in the record area, unless the associated sort-file, appears in a SAME RECORD AREA clause. The logical record is, also at program execution time, available as a record of other files specified in the SAME RECORD AREA clause as the associated sort-file; and it is available to the file associated with sort-record-name. When control is transferred to the input procedure, that file consists of all those records which were passed by means of the execution of RELEASE statements.
5. After a RELEASE statement with the FROM phrase is executed, the record is still available in "identifier".

RETURN statement

Function

The RETURN statement obtains individual records in sorted order from the sort-file.

Format

```
RETURN sort-file-name RECORD [INTO identifier]
       AT END imperative-statement-1
      [NOT AT END imperative-statement-2]
      [END-RETURN]
```

Syntax rules

1. sort-file-name must be defined in a sort-file description entry.
2. The storage area associated with identifier and the record area associated with sort-file-name must not be the same storage area.

General rules

1. Execution of a RETURN statement has the effect that the next record is made available according to the order specified by the keys of the SORT/MERGE statement. Then it can be processed in the record areas of the sort-file.
2. If more than one record description is supplied for the logical record of a file, these records will automatically share the same storage area; this corresponds to an implicit redefinition of the area. The contents of any data item which may be outside the area of the current record will be undefined after a RETURN statement is executed.
3. The INTO phrase may be specified in two cases:
   – if only one record description is subordinate to the sort-merge file description entry,
   – if all record names associated with file-name and the data item referenced by identifier describe a group item of an elementary alphanumeric item.
4. The INTO phrase, when specified, makes the RETURN statement equivalent to a RETURN statement followed by a MOVE statement.

5. When this phrase is written, records are returned from the sort-file and then moved to the area specified by identifier; identifier must not refer to a data item within the record of that sort-file. The move is made according to the rules specified for a MOVE statement without CORRESPONDING phrase. The size of the current record is determined by rules specified for the RECORD clause (see "RECORD clause", page 390). If the sort file description entry contains a RECORD IS VARYING clause, the implied move is a group move.
6. The MOVE statement is not carried out if an at end condition occurs.
7. Any subscripting or indexing of identifier is evaluated after the record has been obtained and immediately before it is moved into the data area.
8. If the INTO phrase was used, the data will be available both in the record area and in the data item specified by identifier.
9. The execution of the RETURN statement causes the next existing record in the file referenced by file-name-1, as determined by the keys listed in the SORT or MERGE statement, to be made available in the record area associated with file-name-1.
   If no next logical record exists in the file referenced by file-name-1, the at end condition exists. Execution continues according to the rules for each statement specified in imperative-statement-1. If a procedure branching or conditional statement which causes explicit transfer of control is executed, control is transferred according to the rules for that statement; otherwise, upon completion of the execution of imperative-statement-1, control is transferred to the end of the RETURN statement and the NOT AT END phrase is ignored, if specified. When the at end condition occurs, execution of the RETURN statement is unsuccessful and the contents of the record area associated with file-name-1 are undefined. After the execution of imperative-statement-1 in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.
10. If an at end condition does not occur during the execution of a RETURN statement, then after the record is made available and after executing any implicit move resulting from the presence of an INTO phrase, control is transferred to imperative-statement-2, if NOT AT END was specified; otherwise, control is transferred to the end of the RETURN statement.
11. END-RETURN delimits the scope of the RETURN statement.
12. A RETURN statement may be used only within the range of the output procedure associated with a SORT/MERGE statement for sort-file-name.

SORT statement

Function

The SORT statement is used to sort records either created in an input procedure or contained in a file according to a data items (set of specified keys). The sorted records are released to an output procedure or entered in a file.

Format

```
SORT sort-file-name

     { ON {DESCENDING} {KEY    }{data-name-1}...  }...
     {    {ASCENDING } {KEY-YY}                   }

     [WITH DUPLICATES IN ORDER]

     [COLLATING SEQUENCE IS alphabet-name]

     { INPUT PROCEDURE IS paragraph-name-1 [{THRU     } paragraph-name-2]}
     {                                     [{THROUGH  }                 ]}
     { USING {file-name-1}...                                           }

     { OUTPUT PROCEDURE IS paragraph-name-3 [{THRU     }  paragraph-name-4]}
     {                                      [{THROUGH  }                 ]}
     { GIVING {file-name-2}...                                          }
```

Syntax rules

1. The SORT statement may be specified anywhere in the Procedure Division except
   – in DECLARATIVES and
   – in input/output procedures which belong to a SORT statement.
2. sort-file-name must be described in a sort-file description (SD) entry in the Data Division.
3. sort-file-name must correspond to the sort-file-name defined in the SELECT clause (format 2).

4. data-name-1... are key data-names. A key is that part of a record which is used as a basis for sorting. Keys must be defined in a record description belonging to an SD entry. They are subject to the following rules:

   a) The data items must not be of variable length.

   b) Key data-names may be qualified (see "Qualification", page 77).

   c) When two or more record descriptions are supplied, the keys need only be described in one of these descriptions. If a key is defined in more than one record description, the descriptions of that key must be identical, and the descriptions must ensure that the key appears in the same position within each record.

   d) A key must not be defined with an OCCURS clause and must not be subordinate to a data item defined with an OCCURS clause.

   e) If the sort file referenced by file-name-1 contains variable length records, all the data items identified by key data-names must be contained within the first n character positions of the record, where n equals the minimum record size specified for the sort file.

   f) A maximum of 64 keys may be specified for any file.

   g) A key must begin within the first 4096 bytes of the record.

   h) Keys for sorting purposes are always listed from left to right in order of significance, regardless of whether they are ascending or descending. Hence, the first specification of data-name-1 is the principal key and the second specification of data-name-1 the subsidiary key.

   i) The maximum length of a key which can be processed by SORT depends on the format of the key. The maximum length is 16 bytes for the packed decimal format (PD), the length of the record for a character-string, and 256 bytes for all other formats.

   j) The keys following the KEY-YY specification must be defined either with PIC 99 USAGE DISPLAY or USAGE PACKED-DECIMAL (see section 10.3).

5. file-name-1..., file-name-2... must be defined in a file description (FD) entry in the Data Division.

6. The record length of records from input files passed to the SORT operation are governed by the following rules: If the sort file has a fixed length record format, program execution is terminated if the record is too long. If the record is too short, the missing spaces will be filled with blanks. If the sort file has a variable length record format, the input records are accepted without any alteration of their length. This length must be within the range specified for the sort file in the RECORD clause (see "RECORD clause", page 390).

7. At least one ASCENDING/DESCENDING phrase must be specified in a SORT statement.
8. No pair of file-names in the same SORT statement may be specified in the SAME SORT AREA or SAME SORT-MERGE AREA clause.
9. If file-name-2 references an indexed file, the first specification of data-name-1 must be associated with an ASCENDING phrase and the data item referenced by that data-name-1 must occupy the same character positions in its record as the data item associated with the prime record key for that file.

General rules

1. The collating sequence is set by means of the ASCENDING/DESCENDING option:
   a) When ASCENDING is specified, the sorted sequence will be from the lowest to the highest value of the key, i.e. in ascending order.
   b) When DESCENDING is specified, the sorted sequence will be from the highest to the lowest value of the key, i.e. in descending order.

   The collating sequence is governed by the same rules as apply to the comparison of operands in "Relation conditions".
2. If DUPLICATES is specified and several records have the same contents in all of their sort items, then the order of return of these records is as follows:
   a) If no input procedure is specified, the records are returned in the order in which the associated files were specified in the SORT statement. If records with identical sort item contents exist in one and the same file, the records are returned in the order in which they were entered.
   b) If an input procedure is specified, the records are returned in the order in which they left the input procedure.
3. If DUPLICATES is not specified and several records have the same contents in all of their items, then the order of return of these records is undefined.

4. When the program is executed, the collating sequence for the comparison of nonnumeric sort items is set as follows:

   a) If COLLATING SEQUENCE has been specified in the SORT statement, this entry is used as a sort criterion.

   b) If COLLATING SEQUENCE was not specified in the SORT statement, the program-specific collating sequence will be used (see "OBJECT-COMPUTER paragraph", page 124).

5. INPUT PROCEDURE indicates that the Procedure Division contains an input procedure to process records prior to sorting. If INPUT PROCEDURE is specified, control passes to it when the input section of the SORT program is ready to accept the first record. During RELEASE statement processing the input procedure releases records to the sort-file (see "RELEASE statement", page 660). The compiler inserts a return mechanism at the end of the last section in the input procedure, i.e. once the last statement in the input procedure has been processed the input procedure is terminated and the released records will be sorted in the sort-file. The following rules apply to the input procedure, which is a self-contained section within the Procedure Division:

   a) It must consist of one or more sections that are written consecutively.

   b) It must contain at least one RELEASE statement so that records can be released to the sort-file (see "RELEASE statement", page 660).

   c) It must not lead to the execution of a MERGE, RETURN, or SORT statement.

   d) It may include any procedures needed to select, modify or copy records.

   e) It is permitted to leave the input procedure if the programmer makes sure that a transfer from the input procedure is followed by a return to it, in order to effect a proper exit from this procedure (i.e. processing its last statement).

   f) It is permitted to branch from points outside an input procedure to procedure-names within that procedure if such a branch does not involve a RELEASE statement or the end of the input procedure.

6. An input procedure, when specified, is processed before the records in the sort-file are sorted.

7. If the USING phrase is specified, all the records in the input files (file-name-1...) are transferred to the sort file referenced by sort-file-name. The input files must not be in the open mode when execution of the SORT statement begins. The execution of the SORT statement for each of the named files consists of the following phases:

   a) The processing of the file is initiated. The initiation is performed as if an OPEN statement with the INPUT phrase had been executed.

b) The logical records are obtained and released to the sort operation. Each record is obtained as if a READ statement with the NEXT RECORD and the AT END phrases had been executed. For a relative file, the content of the relative key data item is undefined after the execution of the SORT statement if file-name-1 is not additionally referenced in the GIVING phrase.
A relative file must be defined in the FILE CONTROL paragraph with ACCESS MODE IS SEQUENTIAL.

c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed. This termination is performed before the sort operation begins.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the input files or their record area declarations.

8. OUTPUT PROCEDURE means that the Procedure Division contains an output procedure in which records are processed after sorting. When the OUTPUT PROCEDURE phrase is used, control is passed to the output procedure after the sort-file has been processed by the SORT command. During RETURN statement processing the output procedure accepts the records from the sort-file. The compiler inserts a return mechanism at the end of the last section in the output procedure, i.e. once the last statement in the output procedure has been executed the procedure is terminated and control passes to the statement that follows the SORT statement.
The following rules apply to the output procedure, which is a self-contained section within the Procedure Division:

   a) It must consist of one or more sections that are written consecutively and that do not form part of an input procedure.

   b) It must contain at least one RETURN statement to make the sorted records available for processing (see "RETURN statement", page 661).

   c) It must not lead to the execution of a MERGE, RELEASE, or SORT statement.

   d) It may include any procedures needed to select, modify, or copy records before they are transferred.

   e) It is permitted to leave the output procedure if the programmer makes sure that a transfer from the output procedure is followed by a return to it, in order to effect a proper exit from this procedure (i.e. processing its last statement).

   f) It is permitted to branch from points outside an output procedure to procedure-names within that procedure if such a branch does not involve a RETURN statement or the end of the output procedure.

9. When the INPUT PROCEDURE or OUTPUT PROCEDURE phrase is used, a branch is performed in the program as if it were a format-1 PERFORM statement. This means that all sections that form the procedure are run once and procedure execution terminates once the last statement has been processed. Thus, either procedure (or both) may be terminated by an EXIT statement.

10. In segmented programs, the following restrictions apply:

    a) If a SORT statement appears in a section that is not located in an independent segment, then all input or output procedures referred to by that SORT statement must be either wholly contained within one fixed segment, or wholly contained in a single independent segment.

    b) If a SORT statement appears in an independent segment, then all input or output procedures referred to by that SORT statement must be either wholly contained within one fixed segment, or wholly contained in the same independent segment as the SORT statement.

    These restrictions do not apply to the compiler described in this manual.

11. If GIVING file-name-2... is specified, all the sorted records are written on the output file (file-name-2).
    The output file must not be in the open mode when execution of the SORT statement begins. The SORT statement is executed for each of the referenced files in the following way:

    a) The processing of the file is initiated. The initiation is performed as if an OPEN statement with the OUTPUT phrase had been executed. This initiation is performed after the execution of any input procedure.

    b) The sorted logical records are returned and written onto the output file as if a WRITE statement without any optional phrases had been executed. The length of these records must be within the range defined for the output file (see "RECORD clause", page 390).
    For a relative file, the relative key data item for the first record returned contains the value "1"; for the second record returned, the value "2", etc. After execution of the SORT statement, the content of the relative data item indicates the last record returned to the file. The file must be defined in the FILE-CONTROL paragraph with ACCESS MODE IS SEQUENTIAL.

    c) The processing of the file is terminated. The termination is performed as if a CLOSE statement without optional phrases had been executed.

These implicit functions are performed such that any associated USE AFTER EXCEPTION/ERROR procedures are executed; however, the execution of such a USE procedure must not cause the execution of any statement manipulating the input files or their record area declarations. On the first attempt to write beyond the externally defined boundaries of the file, any USE AFTER STANDARD EXCEPTION/ERROR procedure specified for the file is executed; if control is returned from this USE procedure or if no such USE procedure was specified, the processing of the file is terminated, as described in c) above.

12. Since the SORT statement is not directed at individual records, it does not conform to the standard input/output statements (READ, WRITE, etc.). The READ statement, when executing, reads a single record; likewise, the WRITE statement writes an individual record. The SORT statement, on the other hand, does not treat an individual record but an entire file. Thus, this entire file must be placed at the disposal of SORT, either via the USING phrase or by repeated use of the RELEASE statement within an input procedure, before SORT can function. The SORT routine alters the sequence of the records within the file, and hence the first record returned by the SORT routine is, as a rule, not the first record released to the routine. SORT cannot provide any output before it has received the whole of the input.

10.1.9 Special registers for file SORT

In the compiler, four special registers are provided for communication between the programmer and the SORT control routine. Each of these registers is a 4-byte binary data item with a fixed name; the description of each register is PICTURE S9(8) USAGE IS COMPUTATIONAL. The data description entries for the registers are generated automatically by the compiler and cannot be declared by the programmer.

SORT-FILE-SIZE register

The programmer may set the contents of the SORT-FILE-SIZE register to the approximate number of records to be processed in the next sort. This must be done before the SORT statement is executed. From this information, the sort routine calculates how much space it will need on internal working files. SORT-FILE-SIZE is initially set to zero by the compiler; and, if the value of the register is still zero at the time a sort begins, the SORT control routine will make a default space allocation. Thus, the programmer does not have to enter any estimate of file size into this special register, although supplying such information aids the efficiency of the sort. The value entered in the special register by the user (e.g. MOVE 25 TO SORT-FILE-SIZE) is released to the control routine.

SORT-CORE-SIZE register

The programmer may set the contents of the SORT-CORE-SIZE register to the number of bytes of memory that are to be available for use by the SORT routine. This must be done before the SORT statement is executed.

SORT-MODE-SIZE register

SORT-MODE-SIZE may be set if variable-length records are to be sorted.

The programmer may set this register to the most commonly used record length in the input file. The value may not be less than the lowest value and not greater than the highest value in the sort-file description. The appropriate value must be moved into SORT-MODE-SIZE before the execution of the SORT statement. The compiler initializes the contents of this register to zero; and, if the programmer has not supplied any other value before the SORT statement is executed, the maximum record length will be assumed to be the most common record length.

SORT-RETURN register

After a SORT statement or RELEASE/RETURN statement has been executed, the special register SORT-RETURN contains a value to indicate whether the sort has been successful. A value of zero indicates that the sort was successful. A non-zero value indicates that the sort terminated abnormally.

Summary

None of the special registers (except SORT-RETURN) has its value reset or set to zero by the execution of the SORT statement. If a program contains several sorts, the programmer must move appropriate values into the SORT-FILE-SIZE, SORT-CORE-SIZE and SORT-MODE-SIZE registers before each sort is executed.

It should be noted that the special registers are binary items and, therefore, cannot be used as immediate operands of ACCEPT statements.

Example 10-1

```
ACCEPT MODE-SIZE FROM MASTER-FILE.
MOVE MODE-SIZE TO SORT-MODE-SIZE.
```

Since the ACCEPT statement transfers input data from the terminal without conversion, the MOVE statement is used to achieve conversion to COMPUTATIONAL.

10.1.10 Examples of file SORT

Example 10-2

Sort processing with one output file

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORT1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE ASSIGN TO "MASTER-FILE".
    SELECT OUTPUT-FILE ASSIGN TO "OUTPUT-FILE".
    SELECT SORT-FILE ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE LABEL RECORD STANDARD.
01  MASTER-RECORD.
    02  E0            PIC X.
    02  E1            PIC 9(4).
    02  E2            PIC 9(4).
    02  E3            PIC 9(4).
FD  OUTPUT-FILE LABEL RECORD STANDARD.
01  OUTPUT-RECORD.
    02  A0            PIC X.
    02  A1            PIC 9(4).
    02  A2            PIC 9(4).
    02  A3            PIC 9(4).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD.
    02  S0            PIC X.
    02  S1            PIC 9(4).
    02  S2            PIC 9(4).
    02  S3            PIC 9(4).
PROCEDURE DIVISION.
P1 SECTION.
SORTING.
    SORT SORT-FILE ASCENDING S1 S2 S3          }
                                               } ---------------------- (1)
         USING MASTER-FILE GIVING OUTPUT-FILE.}
    STOP RUN.
```

(1) The sort operation takes place in the following stages:

1. The records are released from the input file MASTER-FILE to the SORT-FILE.
2. The records are sorted in the sort-file according to ascending S1, or (in those records with identical S1) according to ascending S2, or (in records with identical S1 and S2) according to ascending S3.
3. The records are released from the sort-file to the OUTPUT-FILE.

Example 10-3

Sort processing with two output files

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORT2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE ASSIGN TO "MASTER-FILE".
    SELECT OUTPUT-FILE-1 ASSIGN TO "OUTPUT-FILE-1".
    SELECT OUTPUT-FILE-2 ASSIGN TO "OUTPUT-FILE-2".
    SELECT SORT-FILE ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE LABEL RECORD STANDARD.
01  INPUT-RECORD.
    02  E0           PIC X.
    02  E1           PIC 9(4).
    02  E2           PIC 9(4).
    02  E3           PIC 9(4).
FD  OUTPUT-FILE-1 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-1 PIC X(13).
FD  OUTPUT-FILE-2 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-2 PIC X(13).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD.
    02  S0            PIC X.
    02  S1            PIC 9(4).
    02  S2            PIC 9(4).
    02  S3            PIC 9(4).
WORKING-STORAGE SECTION.
01  INPUT-STATUS PIC X VALUE LOW-VALUE.
    88  INPUT-END VALUE HIGH-VALUE.
01  SORT-STATUS PIC X VALUE LOW-VALUE.
    88  SORT-END VALUE HIGH-VALUE.
PROCEDURE DIVISION.
MAIN SECTION.
M1.
    OPEN INPUT MASTER-FILE OUTPUT OUTPUT-FILE-1 OUTPUT-FILE-2.
    SORT SORT-FILE ASCENDING S1 S2 S3                          ┐
         INPUT PROCEDURE IPROC                                 ├── (1)
         OUTPUT PROCEDURE OPROC.                               ┘
    CLOSE MASTER-FILE OUTPUT-FILE-1 OUTPUT-FILE-2.
ME.
    STOP RUN.
IPROC SECTION.
IP0.
    PERFORM UNTIL INPUT-END                          ┐
      READ INPUT                                     │
      AT END                                         │
        SET INPUT-END TO TRUE                        │
      NOT AT END                                     │
        IF E0 NOT = "C"                              ├── (2)
          THEN                                       │
          RELEASE SORT-RECORD FROM INPUT-RECORD      │
        END-IF                                       │
      END-READ                                       │
    END-PERFORM.                                     ┘
```

```
OPROC SECTION.
A0.
    PERFORM UNTIL SORT-END                                    ┐
      RETURN SORT-FILE                                        │
      AT END                                                  │
        SET SORT-END TO TRUE                                  │
      NOT AT END                                              │
        IF SO = "A"                                           } ———————————————— (3)
        THEN                                                  │
          WRITE OUTPUT-RECORD-1 FROM SORT-RECORD              │
        ELSE                                                  │
          WRITE OUTPUT-RECORD-2 FROM SORT-RECORD              │
        END-IF                                                │
      END-RETURN                                              │
    END-PERFORM.                                              ┘
```

(1) 1. Control passes to the input procedure (IPROC SECTION).

2. The records in the sort-file are sorted in ascending order according to S1 S2 S3.

3. Control passes to the output procedure (OPROC SECTION).

(2) A record is read from the input file. Only those records without a "C" in their first character position are to be processed. If the INPUT-RECORD is valid, it is released to the sort-file as SORT-RECORD. This process continues until the end of the input-file is encountered. Control then returns to the SORT statement.

(3) A record is released from the sort-file. If its first character position contains an "A", it is written to the first output file (OUTPUT-FILE-1). All other records are written to the second output file (OUTPUT-FILE-2). When all the records in the sort-file have been processed, the statement following the SORT statement is executed.

Example 10-4

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MERGE1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE-1 ASSIGN TO "MASTER-FILE-1".
    SELECT MASTER-FILE-2 ASSIGN TO "MASTER-FILE-2".
    SELECT OUTPUT-FILE-1 ASSIGN TO "OUTPUT-FILE-1".
    SELECT OUTPUT-FILE-2 ASSIGN TO "OUTPUT-FILE-2".
    SELECT SORT-FILE ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE-1 LABEL RECORD STANDARD.
01  MASTER-RECORD-1.
    02  E10           PIC X.
    02  E11           PIC 9(4).
    02  E12           PIC 9(4).
    02  E13           PIC 9(4).
FD  MASTER-FILE-2 LABEL RECORD STANDARD.
01  MASTER-RECORD-2.
    02  E20           PIC X.
    02  E21           PIC 9(4).
    02  E22           PIC 9(4).
    02  E23           PIC 9(4).
FD  OUTPUT-FILE-1 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-1 PIC X(13).
FD  OUTPUT-FILE-2 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-2 PIC X(13).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD.
    02  S0            PIC X.
    02  S1            PIC 9(4).
    02  S2            PIC 9(4).
    02  S3            PIC 9(4).
PROCEDURE DIVISION.
MAIN SECTION.
M01.
    MERGE SORT-FILE ON ASCENDING S1 S2 S3  }
    USING MASTER-FILE-1 MASTER-FILE-2      } ------------------------- (1)
    GIVING OUTPUT-FILE-1 OUTPUT-FILE-2.    }
M02.
    STOP RUN.
```

(1) The records from two files are sorted according to the same criteria and output to two identical files in sorted order.

All files are sorted in ascending order according to the sort criteria S1 S2 S3.

Example 10-5

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MERGE2.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE-1 ASSIGN TO "MASTER-FILE-1".
    SELECT MASTER-FILE-2 ASSIGN TO "MASTER-FILE-2".
    SELECT OUTPUT-FILE-1 ASSIGN TO "OUTPUT-FILE-1".
    SELECT OUTPUT-FILE-2 ASSIGN TO "OUTPUT-FILE-2".
    SELECT SORT-FILE ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE-1 LABEL RECORD STANDARD.
01  MASTER-RECORD-1.
    02  E10           PIC X.
    02  E11           PIC 9(4).
    02  E12           PIC 9(4).
    02  E13           PIC 9(4).
FD  MASTER-FILE-2 LABEL RECORD STANDARD.
01  MASTER-RECORD-2.
    02  E20           PIC X.
    02  E21           PIC 9(4).
    02  E22           PIC 9(4).
    02  E23           PIC 9(4).
FD  OUTPUT-FILE-1 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-1 PIC X(13).
FD  OUTPUT-FILE-2 LABEL RECORD STANDARD.
01  OUTPUT-RECORD-2 PIC X(13).
SD  SORT-FILE LABEL RECORD STANDARD.
01  SORT-RECORD.
    02  S0            PIC X.
    02  S1            PIC 9(4).
    02  S2            PIC 9(4).
    02  S3            PIC 9(4).
WORKING-STORAGE SECTION.
01  MERGE-STATUS PIC X VALUE LOW-VALUE.
    88  MERGE-END VALUE HIGH-VALUE.
PROCEDURE DIVISION.
MAIN SECTION.
M01.
    OPEN OUTPUT OUTPUT-FILE-1 OUTPUT-FILE-2.
M02.
    MERGE SORT-FILE ON ASCENDING S1 S2 S3  }
    USING MASTER-FILE-1 MASTER-FILE-2      } ------------------------ (1)
    OUTPUT PROCEDURE IS OPROC1.            }
M03.
    CLOSE OUTPUT-FILE-1 OUTPUT-FILE-2.
    STOP RUN.
STOP RUN.
                .
                .
                .
```

```
OPROC SECTION.
A01.
    PERFORM UNTIL MERGE-ENDE                    ⎫
      RETURN SORT                               ⎪
      AT END                                    ⎪
        SET MERGE-END TO TRUE                   ⎬ ———————————————— (2)
      NOT AT END                                ⎪
        WRITE OUTPUT-RECORD-1 FROM SORT-RECORD  ⎪
        WRITE OUTPUT-RECORD-2 FROM SORT-RECORD⎭
      END-RETURN
    END-PERFORM.
```

(1) The records of both files are sorted according to the same criteria and released to the sort-file in sorted order.
Control then passes to the output procedure (OPROC SECTION).

(2) The sort-file is released record-by-record and written to the output files.

10.2 Sorting of tables

SORT statement

Function

The SORT statement causes table elements to be arranged according to a user-defined collating sequence.

Format

```
SORT data-name-2 {ON {ASCENDING } { KEY     } {data-name-1}...}...
                     {DESCENDING}   KEY-YY

    [WITH DUPLICATES IN ORDER]
    [COLLATING SEQUENCE IS alphabet-name]
    [USING data-name-3]
```

Syntax rules

1. The SORT statement may be used anywhere in the Procedure Division except in DECLARATIVES and input and output procedures that belong to a SORT statement.
2. data-name-2 and data-name-3, if USING is specified, specify the table to be sorted. data-name-1... denotes one or more data items to be used as sort keys (key fields).
3. The record specified by data-name-2 must be defined in a data description entry, and is subject to the following rules:
   a) data-name-2 may be qualified.
   b) The data description entry for data-name-2 must contain an OCCURS clause, i.e. be defined as a a table element.
   c) If the table specified by data-name-2 is subordinate to a table (multidimensional table), then data-name-2 must be defined as an indexable table, i.e. an index-name must be specified in the data description entry for the superordinate table by means of the INDEXED-BY phrase. Before execution of the SORT statement, the index-name must be supplied with the desired element number (see "Indexing", page 82).
4. The key fields specified by data-name-1 must be defined in the data description entry for data-name-2, where the following rules apply:

a) data-name-1 is either the same as data-name-2 or the same as a data item subordinate to data-name-2.

b) data-name-1 may be qualified.

c) If data-name-1 refers to a data item subordinate to data-name-2, then the description of this data item must neither contain an OCCURS clause itself nor be subordinate to a data item whose description contains an OCCURS clause.

d) data-name-1 must not refer to a data item whose description contains a SIGN clause.

e) If the data item specified by data-name-1 is defined as numeric, it must not comprise more than 16 digits including sign.

f) The keys following the KEY-YY specification must be defined either with PIC 99 USAGE DISPLAY or with USAGE PACKED-DECIMAL (see section 10.3).

5. For data-name-3, the same rules apply as for data-name-2.
6. If the USING phrase is used, data-name-2 and data-name-3 must be described as alphanumeric data items.

General rules

1. The key words ASCENDING and DESCENDING apply to all subsequent data-name-1 specifications up to the next key word ASCENDING or DESCENDING.
2. The data items specified by data-name-1 are the sort keys. The sort keys are used hierarchically from left to right for sort processing, irrespective of whether ASCENDING or DESCENDING is specified. The first instance of data-name-1 is thus the main sort key, the second instance of data-name-1 is the next most significant sort key etc.
3. If DUPLICATES is specified and two or more table elements are found to match in respect of all their key fields, then the relative sequence of these table elements will not be changed by sort processing.
4. If DUPLICATES is not specified and two or more table elements are found to match in respect of all their key fields, then the relative sequence of these table elements will be undefined after sort processing.
5. The collating sequence that is used in comparing the nonnumeric key fields is defined as follows on commencement of execution of the SORT statement:

a) If COLLATING SEQUENCE is specified in the SORT statement, this specification serves as the criterion for the collating sequence,

b) If COLLATING SEQUENCE is not specified in the SORT statement, the program-specific collating sequence is used (see "OBJECT-COMPUTER paragraph", page 124).

c) If no program-specific collating sequence is specified, EBCDIC sorting takes place.

6. The table elements sorted in accordance with the ASCENDING/DESCENDING KEY phrases are stored in the table specified by data-name-2.
7. The table elements are sorted through comparison of the contents of the data items defined as sort keys, in accordance with the rules for relation conditions:

   a) If the contents of the compared key fields differ and the ASCENDING phrase is in effect, then the table element whose key field contains the lower value has the lower element number.

   b) If the contents of the compared key fields differ and the DESCENDING phrase is in effect, then the table element whose key field contains the higher value has the lower element number.

   c) If the contents of the compared key fields are the same, the next sort key specified is used for the comparison.

8. Sort keys specified in the SORT statement take priority over any sort keys that may be specified in the data description entry for data-name-2.
9. By specifying USING data-name-3, a second table can be used for sort processing. In this case, the elements of the table specified by data-name-3 are sorted as described above and then transferred to the table specified by data-name-2 in accordance with the rules for the MOVE statement. Note the following with regard to the transfer of the individual table elements:
   If the "sending" element is shorter than the "receiving" element, it is padded with blanks; if it is longer, it is truncated.
10. Transfer of the sorted data-name-3 table elements to the data-name-2 table ends with the last sorted table element (data-name-3) or on reaching the number of table elements for data-name-2. This means that if the data-name-3 table contains more elements than the data-name-2 table, the excess elements are not transferred. If it contains fewer elements, the excess elements of the data-name-2 table are retained unchanged.
11. Sort processing does not change the contents of the data-name-3 table.

Example 10-6

Sorting a table

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABSORT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NAMETAB.
   02 TAB-ELEM OCCURS 10 TIMES.
      03 FORENAME    PIC X(8).
      03 FILLER      PIC X(3) VALUE SPACES.
      03 NAME        PIC X(10).
PROCEDURE DIVISION.
SINGLE SECTION.
INITIALIZATION.
    MOVE "PETER" TO FORENAME (1) MOVE "KRAUS" TO NAME (1).
    MOVE "JANE" TO FORENAME (2) MOVE "FONDA" TO NAME (2).
    MOVE "PETER" TO FORENAME (3) MOVE "FONDA" TO NAME (3).
    MOVE "KARL" TO FORENAME (4) MOVE "KRAUS" TO NAME (4).
    MOVE "UWE" TO FORENAME (5) MOVE "SEELER" TO NAME (5).
    MOVE "WALT" TO FORENAME (6) MOVE "DISNEY" TO NAME (6).
    MOVE "CLARA" TO FORENAME (7) MOVE "WIECK" TO NAME (7).
    MOVE "LEONID" TO FORENAME (8) MOVE "KOGAN" TO NAME (8).
    MOVE "ERICH" TO FORENAME (9) MOVE "FROMM" TO NAME (9).
    MOVE "ELVIS" TO FORENAME (10) MOVE "PRESLEY" TO NAME (10).
    DISPLAY NAMETAB UPON T.
SORTING.
    SORT TAB-ELEM ON ASCENDING KEY NAME FORENAME.
    DISPLAY NAMETAB UPON T.
ENDE.
    STOP RUN.
```

Sorted table:

```
Element number
(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
```

10.3 Sorting two-digit year numbers with a century window

General description

This description applies both to file sorting and to table sorting. The century window is defined in the same way as in the COBOL function YEAR-TO-YYYY. The last year belonging to the window is specified relative to the current year. The specified value determines the number of years in the future which belong to the century window. For example, the value 50 when run in 1998 represents the period from 1949 through 2048.

A SORT special register SORT-EOW (SORT-END-OF-WINDOW) is defined to determine the century window. It is made available in COBOL programs with the SORT or MERGE statement, and implicitly described by the compiler with PIC(7) PACKED-DECIMAL. The value saved in SORT-EOW must be between 0 and 99. The default value is 50.

The ASCENDING/DESCENDING phrases in the SORT and MERGE statements have been extended to enable two-digit year numbers to be used as a SORT key depending on a century window.

In the MERGE statement the position of the century window is determined at the start of processing (evaluation of SORT-EOW and current year). If, for example, files are first sorted with the SORT statement, the same century window must be selected.

Example 10-7

SORT with century window selection

```
IDENTIFICATION DIVISION.
PROGRAM-ID.SORTIERY.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT MASTER-FILE ASSIGN TO "MASTER-FILE".
   SELECT OUTPUT-FILE ASSIGN TO "OUTPUT-FILE".
   SELECT SORT-FILE ASSIGN TO "SORTWK".
DATA DIVISION.
FILE SECTION.
FD MASTER-FILE.
01 MASTER-RECORD.
   02 E0      PIC X.
   02 EY1     PIC 99.
   02 EY2     PIC 99 USAGE PACKED-DECIMAL.
   02 E3      PIC X(10).
FD OUTPUT-FILE.
01 OUTPUT-RECORD.
   02 A0      PIC X.
   02 AY1     PIC 99.
   02 AY2     PIC 99 USAGE PACKED-DECIMAL.
   02 A3      PIC X(10).

SD SORT-FILE.
01 SORT-RECORD.
   02 S0       PIC X.
   02 SY1      PIC 99.
```

```
   02 SY2      PIC 99 USAGE PACKED-DECIMAL.
   02 S3       PIC X(10).
PROCEDURE DIVISION.
P1 SECTION.
SORT.
* Specification of the century window: starting from the
* year 1998, 2008 is the final year of the century window

     MOVE 10 TO SORT-EOW

* The keys SY1 and SY2 are handled as two-digit year
* numbers within the century window of 1909-2008:
* 06 is greater than 75

     SORT SORT-FILE ASCENDING KEY-YY SY1 SY2
                  DESCENDING KEY S3
     USING MASTER-FILE GIVING OUTPUT-FILE.
SORTEND.
     STOP RUN.
```

11 Source text manipulation

The statements COPY and REPLACE can be used to extend and/or modify a source program text with texts stored in a library. These two statements, which are effective at compile time, can be used either independently or together.

A COPY statement extends an existing source program text with COBOL text lines taken from a library element. The compiler then treats the inserted text as part of the source program. The REPLACING phrase offers the additional possibility of replacing each occurrence of a literals, an identifier, a word or a group of words with another text.

The REPLACE statement causes parts of the source program text to be replaced with new texts. With a REPLACE statement, source programs written by the programmer in his self-defined notation can be converted at compile time into syntactically correct phrases, clauses and statements.

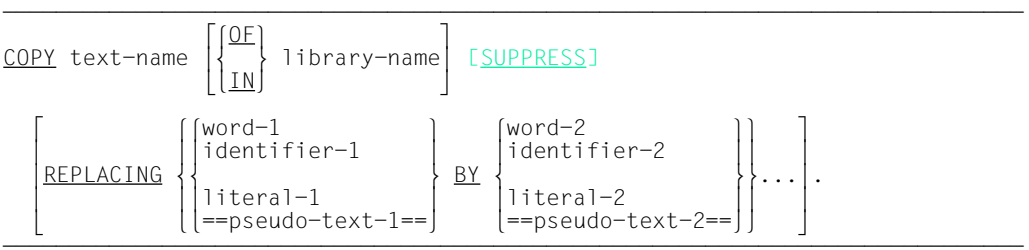
The compiler does not check the source program text until execution of all COPY and REPLACE statements has been completed.

COPY statement

Function

The COPY statement inserts texts from a library into a source program. At the same time, it permits parts of the new texts to be replaced.

Format



Syntax rules

1. text-name is the name of the text which is to be inserted in the source program. The text is stored as a library element ("member") with this name in a library.
   text-name is a user-defined name with a length of 1 to 30 characters.
2. If text-name is qualified by library-name, the specified library is searched for the text.
   If text-name is not qualified by library-name, up to ten libraries are searched for the text.
   Assignment of libraries at compile time is described in more detail in the "COBOL85 User Guide" [1].
3. The COPY statement must be preceded by a space or some other separator symbol.
4. pseudo-text-1 must contain at least one text word, while pseudo-text-2 may be empty (====) or may contain only spaces, comma, semicolon and comment lines.
5. Text words* in pseudo-text-1 and pseudo-text-2 and text words which form identifier-1, identifier-2, literal-1, literal-2, word-1, word-2 may be continued on the next line. The pseudo-text separator (==), on the other hand, must not be continued.
6. word-1 and word-2 may be any single COBOL word except "COPY".
7. Lowercase letters that are used in text words are equivalent to the corresponding uppercase letters.

* ("Text word" is defined in section 2.1, “Glossary”)

8. A COPY statement is recognized anywhere within a program except:
   – in comment lines
   – within nonnumeric literals.
9. The text generated by one COPY statement must not contain another COPY statement. However, the compiler described here permits this in the outermost COPY. If a COPY statement does not contain the REPLACING clause, the related library text may contain COPY statements; these statements may contain the REPLACING clause.

General rules

1. Execution of a COPY statement causes the library text specified via text-name to be copied to the source program. The library text then replaces the entire COPY statement (including the terminating separator period).
2. If SUPPRESS is specified, the library text is not included in the source program list.
3. If REPLACING is not specified, the library text is copied unchanged.

COPY...REPLACING...

4. If REPLACING is specified, the library text is copied and the text preceding BY (A-operand) is replaced by the text following BY (B-operand).
5. Text is replaced only if each text word in the text preceding BY matches, character for character, the corresponding library text.
6. identifier-1, word-1 and literal-1 are treated as pseudo-text containing nothing but identifier-1, word-1 and literal-1.
7. The comparison of the library text with the text preceding BY (A-operand, the result of which decides whether or not a text is replaced, is carried out as follows:
   a) Everything which precedes the first text word in the library text is copied into the source program. Starting with the first of the A-operands (pseudo-text-1, identifier-1, word-1, literal-1), each A-operand is in turn compared with the corresponding number of library text words, starting each time with the first library word.
   b) Each of the separators comma (,␣), semicolon (;␣) or space in pseudo-text-1 or in the library text is treated as a single space. Each string of one or more spaces is treated as a single space.
   c) If there is no match, the comparison is repeated with the next A-operand. This process is continued until a match is found or until there are no more A-operands.

d) When all A-operands of the REPLACING phrase have been compared with the library text without finding a match, the first word of the library text is copied into the source program. The next library text word is then regarded as the first word and the comparison operation is started again with the first A-operand.

e) Whenever a match occurs between an A-operand and the library text, the corresponding B-operand (pseudo-text-2, identifier-2, word-2 or literal-2) is placed in the source program and replaces the library text corresponding to the A-operand. The library text word following the last text word which was replaced is then regarded as the first text word and the comparison operation is repeated starting with the first A-operand.

f) The comparison cycle is repeated until the last library text word has been either copied or replaced.

8. Comment lines or empty lines in the library text or pseudo-text-1 are treated as space characters for the comparison. Any comment lines or empty lines in pseudo-text-2 are moved to the source program unchanged. Comment lines and empty lines in the library text are copied unchanged into the source program unless they are located within a text word sequence which matches pseudo-text-1 and which is therefore replaced.

9. Debugging lines are permitted within a library text or pseudo-text. Text words in a debugging line are subjected to the comparison rules as if there were no 'D' in the indicator area (column 7) of the debugging line. A debugging line lies within a pseudo-text if the opening pseudo-text separator appears on a line which precedes the debugging line.

10. Each word copied from the library, but not replaced, is copied so that it starts in the same area of the line as in the library text.
    If a text word in the library text starts in area A, but a preceding text word has been replaced by a longer text, then the following word begins in area B if there is insufficient space in area A.
    If replaced by pseudo-text, each text word begins in the same area as specified in pseudo-text-2. If replaced by identifier-2, literal-2 and word-2, the replacement text begins in the area in which the leftmost replaced text word would stand if it had not been replaced.
    Text from a library is copied to the same area of the source program which it occupied in the library. For this reason, it must comply with the rules of the COBOL reference format

11. If there is a COPY statement in a debugging line, the copied text is treated as if it were also in debugging lines. This also applies to additional lines created by replacements in the source program.

12. If a COPY statement with the REPLACING phrase causes a line to be lengthened such that additional lines are required, corresponding continuation lines are generated. These lines may contain a continuation character in column 7.

13. If parts of a text word are to be replaced, the string to be replaced must be enclosed between appropriate separators (e.g. colons or parentheses) in the library text and in the A-operand of the REPLACING phrase, thus identifying it as a separate text word (see example 11-4).

Example 11-1

Assume a German library text named ADR, stored in library ADRLIB and consisting of the following lines:

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

In a source program, the COPY statement is written as follows:

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

After execution of the COPY statement, the source program contains:

```
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) Though appearing in the source listing, the COPY statement is treated as comments during compilation.

Example 11-2

In order to modify library text, the REPLACING phrase is specified in the COPY statement:

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

After execution of this COPY statement, the source program contains:

```
01  ADRESSE.
    COPY ADR OF ADRLIB                                    )
     REPLACING ADRESSE BY ADDRESS                         |
               STRASSE BY STREET                          |
               ==PLZ PIC 9(5)== BY ==POST CODE PIC X(6)==  } ——— (1)
               ORT BY TOWN                                |
               LAND BY COUNTRY.                           |
    05  STREET PIC X(20).                                 )
    05  POST CODE PIC X(6).
    05  TOWN PIC X(20).
    05  COUNTRY PIC X(20).
```

(1) Though appearing in the source listing, the COPY statement is treated as comments during compilation.

The changes only affect this source program; the text in the library remains unchanged.

Example 11-3

In order to replace the following string in the library text:

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

by the string

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

a space must be inserted after the period in pseudo-text-1 and pseudo-text-2. Otherwise a search will be conducted for a period, but the library text contains a separator period:

```
...REPLACING ==PIC X(30).␣== BY ==PIC X(40).␣==
```

Example 11-4

The library text in ELEM

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

is expanded by the COPY statements

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

to

```
...
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).
...
```

REPLACE statement

Function

The REPLACE statement is used to replace source program text.

Format 1

```
REPLACE {==pseudo-text-1== BY ==pseudo-text-2==}... .
```

Format 2

```
REPLACE OFF.
```

Syntax rules

1. If a REPLACE statement is not the first statement in a separately compiled program, it must be preceded by the period separator character or some other separator symbol. One REPLACE statement must be terminated before another REPLACE statement is started.
2. A REPLACE statement is recognized anywhere within a program except:
   – in comment lines
   – within nonnumeric literals
3. pseudo-text-1 must contain at least one word, while pseudo-text-2 may be empty (====) or may contain only spaces, comma, semicolon, and comment lines.

General rules

1. Format 1 of the REPLACE statement specifies that each occurrence of pseudo-text-1 is to be replaced by pseudo-text-2.
2. Format 2 of the REPLACE statement terminates the current text replacement operation.
3. A REPLACE statement remains effective from the point where it is specified up to the next REPLACE statement or up to the end of a separately compiled program.

4. All REPLACE statements in a source program or in a library text are executed only after all COPY statements in the source program or library text have been executed. The operands of a REPLACE statement cannot be modified with the REPLACING phrase of a COPY statement.
5. The text which results from execution of a REPLACE statement must not contain either a REPLACE statement or a COPY statement.
6. Lowercase letters that are used in text words are equivalent to the corresponding uppercase letters.
7. The comparison operation whose results determine whether or not a text is to be replaced is executed as follows (the term "source text" is used here to mean both the source program text and the library text):
   a) Starting with the first text word of the source text after the REPLACE statement and the first word of pseudo-text-1, pseudo-text-1 is compared with the corresponding number of consecutive text words.
   b) pseudo-text-1 matches the source text only if the sequence of text words in pseudo-text-1 is identical, character for character, with the corresponding sequence of text words in the source text.
   c) If no match is found, the comparison is repeated with each subsequent occurrence of pseudo-text-1 until a match is found or until there is no further pseudo-text-1.
   d) When each specified pseudo-text-1 has been compared with the source text without finding a match, the next word of the source text is regarded as the first text word and the comparison operation starts again with the first pseudo-text-1.
   e) If a match between pseudo-text-1 and the source text is found, this part of the source text is replaced with the corresponding pseudo-text-2. The text word in the source text which immediately follows the replaced text is then regarded as the first text word and the comparison operation starts again with the first pseudo-text-1.
   f) The comparison operation continues until the last text word affected by the REPLACE statement has been replaced or has been compared as the first text word with every pseudo-text-1.
8. Comment lines and empty lines in the source text or in pseudo-text-1 are treated as spaces for the comparison operation. The order of the text words in the source text and in pseudo-text-1 is defined by the reference format (see "Reference format", p. 102). Any comment lines or empty lines in pseudo-text-2 are transferred unchanged to the source text. Any comment line or empty line in the source text which lies within a sequence of text words which matches pseudo-text-1 will not appear in the final version of the program text.

9. Debugging lines are permitted in the pseudo-text. Text words in pseudo-text-1 which lie in a debugging line are treated, during comparison, as if there were no 'D' in the indicator area (column 7).
10. Except for the COPY and REPLACE statements themselves, the syntax of the source text cannot be checked until all COPY and REPLACE statements have been fully executed.
11. Text words introduced by a REPLACE statement are placed in the program text in accordance with the reference format. When text words from pseudo-text-2 are moved to the program text, additional spaces are inserted only where they existed in the original text or in pseudo-text-2. This includes the implicit space between lines of the source program.
12. If a REPLACE statement introduces additional lines to the source program, the indicator area of each of these lines is marked with the character specified in the first line which was replaced. The only exception to this is where the original source program line contained a hyphen: in this case, the inserted line contains a space.
    If a COPY statement with the REPLACING phrase causes a line to be lengthened such that additional lines are required, corresponding continuation lines are generated. These lines may contain a continuation character in column 7.
    If the replacement operation requires continuation of a literal in a debugging line, there is an error in the source program.

SOURCE FIXED compiler directive

General description

The next COBOL standard permits the source formats `free` and `fixed`, and enables these formats to be switched at will. For COPY elements to be imported, the format set in the source applies. If the COPY element is to be independent of this format, the format must be set in the COPY element **itself**. This format then applies only to the corresponding COPY element.

Function

The SOURCE FIXED compiler directive protects the source format of COPY elements from external interference.

Format

```
>>SOURCE FIXED.
```

Syntax rules

1. The directive may not appear before column 8.
2. There must be one blank between SOURCE and FIXED.

General rules

This directive is currently treated as a comment, since the compiler supports only the `fixed` source format.

12 Intrinsic functions

Intrinsic functions make it possible to reference a temporary data item that is made available by the COBOL program and whose value is calculated automatically when the function is used.

12.1 General

Function-name

Each intrinsic function has a name that the programmer uses to address it. A function-name is a key word from a special list of COBOL words and is a necessary part of the function-identifier. Outside the function-identifier context, a key word can also be used as a user-defined word.

Returned value of a function

Each function that executes successfully returns a function result, the *returned value*. To determine the returned value, the function processes the data values supplied by the arguments specified in the function-identifier.
The function result is defined

a) by the length of the returned value in alphanumeric functions,

b) by the sign of the returned value, or by the integer in the case of numeric and integer functions,

c) or, in all remaining cases, by the returned value itself.

For the arguments particular rules apply: data type, number, length and value range of the arguments are determined by the definition of the function. The function returns a defined returned value only if these rules are observed.

Error default value

If a function is supplied with an invalid argument, the function result is undefined. A compiler option can be used in such cases to check the argument values and assign the function the *error default value* that indicates that the function has not executed successfully. A further compiler option can be used to ensure that the error is reported at run time (error messages COB9123 - COB9128). Refer to the COBOL85 User Guide [1] for further information.

Date conversion

The Gregorian calendar is used in the date conversion functions.
The start date of Monday 1, January 1601 was selected to establish a simple relationship between the standard date and DAY-OF-WEEK; in other words, the integer date form 1 was a Monday, DAY-OF-WEEK 1.

Arguments

Arguments specify the values used when executing a function. The arguments are specified in the function-identifier (see 2.4.4) as identifiers, arithmetic expressions, or literals. The format description of each function indicates the number of arguments required (zero, one or more). For some functions the number of arguments can be variable.

Arguments belong to a particular data class or to a subset of a data class. There are the following argument types:

a) Numeric. An arithmetic expression must be specified. The value of the arithmetic expression, including the sign, is used to determine the function value.

b) Alphabetic. A data item of class alphabetic or a nonnumeric literal comprising exclusively alphabetic characters must be specified. The length of the data item specified as argument can be used to determine the function value.

c) Alphanumeric. A data item of class alphabetic or alphanumeric or a nonnumeric literal must be specified. The length of the data item specified as argument can be used to determine the function value.

d) Integer. An arithmetic expression that always produces an integer value must be specified. The value of the arithmetic expression, including sign, is used to determine the function value.

A table can be referenced if the format of a function permits argument repetition. Reference is made by specifying the data-name and any identifiers for the table, followed directly by subscription in which one or more subscripts is the word ALL.

If ALL is specified as a subscript, it has the same effect as specifying each table element at this subscript position.

Example:

ALL subscripts in a three-dimensional table with ten items in each dimension.

```
FUNCTION MAX (TAB(ALL 2 ALL))
```

is the same as

```
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))
```

If the ALL subscript is linked with an OCCURS DEPENDING ON clause, the value range is determined by the object of the OCCURS DEPENDING ON clause. This object must be greater than zero at the time of evaluation of the arguments. If an argument subscripted with ALL is reference-modified, the reference-modifier refers to each implicitly referenced table item.

Function types

Functions are elementary data items. They return alphanumeric, numeric or integer values, and cannot be receiving operands. There are the following types of function:

a) Alphanumeric functions. They belong to class and category alphanumeric.
   Alphanumeric functions have implicit DISPLAY format.

b) Numeric functions. They belong to class and category numeric.
   A numeric function is always treated as signed.
   A numeric function can be used only in an arithmetic expression.
   A numeric function must not be referenced if an integer operand is required, even if evaluation of the function results in an integer value.

c) Integer functions. They belong to class and category numeric.
   An integer function is always treated as signed.
   An integer function may be used only in an arithmetic expression.

12.2 Overview of intrinsic functions

The following table lists the available functions.

The "Arguments" column defines type and number of arguments as follows:

A alphabetic
I integer
N numeric
X alphanumeric

The "Type" column defines function type as follows:

I integer
N numeric
X alphanumeric

Function-name	Arguments	Type	Returned value
ACOS	N1	N	Arccosine of N1
ADDR	A1 or N1 or X1	I	Address of the argument
ANNUITY	N1, I2	N	Ratio of annuity paid for 12 periods at interest of N1 to initial investment of one
ASIN	N1	N	Arcsine of N1
ATAN	N1	N	Arctangent of N1
CHAR	I1	X	Character in position I1 of program collating sequence
COS	N1	N	Cosine of N1
CURRENT-DATE	None	X	Current data and time
DATE-OF-INTEGER	I1	I	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYYYMMDD	I1, I2	I	YYYYMMDD equivalent, depending on the value of I2, of I1 from a standard date with a 2-digit year (YYMMDD)
DAY-OF-INTEGER	I1	I	Julian date equivalent (YYYYDDD) of integer data
DAY-TO-YYYYDDD	I1, I2	I	YYYYDDD equivalent, depending on the value of I2, of I1 from a standard Julian date (YYDDD)
FACTORIAL	I1	I	Factorial of I1

Function-name	Arguments	Type	Returned value
INTEGER	N1	I	Greatest integer not greater than N1
INTEGER-OF-DATE	I1	I	Integer data equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	I1	I	Integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	N1	I	Integer part of N1
LENGTH	A1 or N1 or X1	I	Length of argument
LOG	N1	N	Natural logarithm of N1
LOG10	N1	N	Logarithm to base 10 of N1
LOWER-CASE	A1 or X1	X	All letters in the argument are set to lowercase
MAX	A1... or I1... or N1... or X1...	X I N X	Value of maximum argument
MEAN	N1...	N	Arithmetic mean of arguments
MEDIAN	N1...	N	Median of arguments
MIDRANGE	N1...	N	Mean of minimum and maximum arguments
MIN	A1... or I1... or N1... or X1...	X I N X	Value of minimum argument
MOD	I1, I2	I	I1 modulo I2
NUMVAL	X1	N	Numeric value of simple numeric string
NUMVAL-C	X1, X2	N	Numeric value of numeric string with optional commas and currency sign
ORD	A1 or X1	I	Ordinal position of A1 / X1 in collating sequence
ORD-MAX	A1... or N1... or X1...	I	Ordinal position of maximum argument
ORD-MIN	A1... or N1... or X1...	I	Ordinal position of minimum argument
PRESENT-VALUE	N1 N2...	N	Principal amount repaid by a series of deferred payments, N2, at an interest rate of N1
RANDOM	I1	N	Random number
RANGE	I1... or N1...	I N	Value of maximum argument minus value of minimum argument

Function-name	Arguments	Type	Returned value
REM	N1, N2	N	Remainder of N1/N2
REVERSE	A1 or X1	X	Reverse order of the characters of the argument
SIN	N1	N	Sine of N1
SQRT	N1	N	Square root of N1
STANDARD-DEVIATION	N1...	N	Standard deviation of arguments
SUM	I1... or N1...	I N	Sum of arguments
TAN	N1	N	Tangent of N1
UPPER-CASE	A1 or X1	X	All letters in the argument are set to uppercase
VARIANCE	N1...	N	Variance of arguments
WHEN-COMPILED	None	X	Date and time program was compiled
YEAR-TO-YYYY	I1, I2	I	4-digit equivalent, depending on the value of I2, of the 2-digit year I1

ACOS - Arcosine

The ACOS function returns a numeric value in radians that approximates the arc cosine of argument-1.
The type of this function is numeric.

Format

```
FUNCTION ACOS (argument-1)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to −1 and less than or equal to +1.

Returned values

1. The returned value is the approximation of the arc cosine of argument-1 and is greater than or equal to zero and less than or equal to pi.
2. The error default value is −2.

See also: COS, SIN, ASIN, TAN, ATAN

Example 12-1

```
...
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.
```

Result: +1.82

ADDR - Address of an identifier

The ADDR function returns an integer representing the address of argument-1t.
The type of this function is integer

Format

```
FUNCTION ADDR (argument-1)
```

Arguments

1. argument-1 can be a literal or adata item of any class or category.

Returned values

1. The return value is an integer representing the address of argument-1 at run time.

ANNUITY - Annuity

The ANNUITY function returns a numeric value that approximates the ratio of an annuity paid at the end of each period for the number of periods specified by argument-2 to an initial investment of one. Interest is earned at the rate specified by argument-1 and is applied at the end of the period, before the payment.
The type of this function is numeric.

Format

```
FUNCTION ANNUITY (argument-1 argument-2)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to zero.
3. Argument-2 must be a positive integer.

Returned values

1. When the value of argument-1 is equal to zero, the value of the function is the approximation of: 1 / argument-2
2. When the value of argument-1 is not equal to zero, the value of the function is the approximation of:
   argument-1 / (1 − (1 + argument-1)**(− argument-2))
3. The error default value is −2.

See also: PRESENT-VALUE

Example 12-2

The following program calculates the annual payments for a loan of 100000 at three different interest rates over a period of 1 to 10 years.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. INTEREST-TABLE.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     TERMINAL IS WINDOW
     DECIMAL-POINT IS COMMA.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 CAPITAL              PIC 9(9).
 01 PD                   PIC 99.
 01 CALC-TABLE.
    02 INTEREST PIC V9(7) OCCURS 3 INDEXED BY R-IND-S.
 01 HEADER-LINE.
    02 PIC XX VALUE SPACE.
    02 OCCURS 3 INDEXED BY T-IND-S.
      10 INTR-ED PIC BBBZZ9,999999B.
      10 PIC X VALUE FROM (1) "%" REPEATED TO END.
 01 OUTPUT-TABLE.
    02 THIS-LINE OCCURS 10 INDEXED BY A-IND-Z.
      10 PERIOD  PIC Z9.
      10 RATE PIC BZZZBZZZBZZ9,99 OCCURS 3 INDEXED BY A-IND-S.
 PROCEDURE DIVISION.
 ONLY SECTION.
 PARA.
     MOVE 100000 TO CAPITAL
*** Interest 5,75 % ***
     MOVE 0,0575 TO INTEREST (1)
*** Interest 8,90 % ***
     MOVE 0,0890 TO INTEREST (2)
*** Interest 12,10 % ***
     MOVE 0,1210 TO INTEREST (3)
     PERFORM VARYING R-IND-S FROM 1 BY 1 UNTIL R-IND-S > 3
       SET T-IND-S TO R-IND-S
       MULTIPLY INTEREST (R-IND-S) BY 100 GIVING INTR-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 PD TO A-IND-Z
          MOVE PD TO PERIOD (A-IND-Z)
          COMPUTE RATE (A-IND-Z  A-IND-S) = CAPITAL *
              FUNCTION ANNUITY (INTEREST (R-IND-S) PD)
       END-PERFORM
     END-PERFORM
     DISPLAY HEADER-LINE UPON WINDOW
     PERFORM VARYING A-IND-Z FROM 1 BY 1 UNTIL A-IND-Z > 10
       DISPLAY THIS-LINE (A-IND-Z) UPON WINDOW
     END-PERFORM
     STOP RUN.
```

Result:

```
     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    41 706,46
 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
```

ASIN - Arcsine

The ASIN function returns a numeric value in radians that approximates the arc sine of argument-1.
The type of this function is numeric.

Format

```
FUNCTION ASIN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than or equal to −1 and less than or equal to +1.

Returned values

1. The returned value is the approximation of the arcsine of argument-1 and is greater than or equal to −pi/2 and less than or equal to +pi/2.
2. The error default value is −2.

See also: SIN, COS, ACOS, TAN, ATAN

Example 12-3

```
...
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.
```

Result: 0.46

ATAN - Arctangent

The ATAN function returns a numeric value in radians that approximates the arctangent of argument-1.
The type of this function is numeric.

Format

```
FUNCTION ATAN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the approximation of the arctangent of argument-1 and is greater than −pi/2 and less than +pi/2.

See also: TAN, SIN, ASIN, COS, ACOS

Example 12-4

```
...
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.
```

Result: 0.4228

CHAR - Character in the collating sequence

The CHAR function returns a one-character alphanumeric value that is a character in the program collating sequence having the ordinal position equal to the value of argument-1. The type of this function is alphanumeric.

Format

```
FUNCTION CHAR (argument-1)
```

Arguments

1. Argument-1 must be an integer.
2. The value of argument-1 must be greater than zero and less than or equal to the number of positions in the collating sequence.

Returned values

1. If more than one character has the same position in the program collating sequence, the character returned as the function value is that of the first literal specified for that character position in the ALPHABET clause.
2. If the current program collating sequence was not specified by an ALPHABET clause, the EBCDIC collating sequence is assumed.
3. The error default value is a space.

See also: ORD

Example 12-5

```
...
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.
```

Result: A
The ordinal position of "A" in EBCDIC is 194.

COS - Cosine

The COS function returns the cosine of the angle or arc that is specified in radians by argument-1.
The type of this function is numeric.

Format

```
FUNCTION COS (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the approximation of the cosine of argument-1 and is greater than or equal to −1 and less than or equal to +1.

See also: ACOS, SIN, ASIN, TAN, ATAN

Example 12-6

```
...
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.
```

Result: −0.9999995883

CURRENT-DATE - Current date

The CURRENT-DATE function returns a 21-character alphanumeric value that represents the calendar date and time of day.
The type of this function is alphanumeric.

Format

```
FUNCTION CURRENT-DATE
```

Returned values

1. The character positions returned, numbered from left to right, are:

Character position	Contents
1-4	Four numeric digits of the year (Gregorian calendar).
5-6	Two numeric digits of the month of the year,in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-21	0000000

See also: DATE-OF-INTEGER, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, WHEN-COMPILED

Example 12-7

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  A-DATE PIC XXXX/XX/XXBBXXBXXBXXBXXBBX(5).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION CURRENT-DATE TO A-DATE.
    DISPLAY A-DATE UPON T.
    STOP RUN.
```

Result: 1995/08/10 14 36 19 00 00000

DATE-OF-INTEGER - Date conversion

The DATE-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to standard date form (YYYYMMDD).
The type of this function is integer.

Format

```
FUNCTION DATE-OF-INTEGER (argument-1)
```

Arguments

1. Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

Returned values

1. The returned value represents the ISO Standard date equivalent of the integer specified in argument-1.
2. YYYY represents a year in the Gregorian calendar, MM represents the month of that year, and DD represents the day of that month.
3. The error default value is 0.

See also: DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, CURRENT-DATE, WHEN-COMPILED

Example 12-8

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  A-DATE PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE A-DATE = FUNCTION DATE-OF-INTEGER (50000).
    DISPLAY A-DATE UPON T.
    STOP RUN.
```

Result: 17371123
The 50000th day as of 31.12.1600 was 23.11.1737.

DATE-TO-YYYYMMDD - year conversion

The DATE-TO-YYYYMMDD function converts a date specified with argument-1 from the standard date format with a 2-digit year into the standard date format with a 4-digit year. The end of the 100-year interval in which the year specified in argument-1 falls is determined by adding argument-2 to the current year (the year in which the function is executed) ("floating window").
The type of this function is integer.

Format

```
FUNCTION DATE-TO-YYYYMMD (argument-1 [argument-2])
```

Arguments

1. argument-1 must be a positive number less than 1000000.
2. argument-2, if specified, must be an integer.
3. If argument-2 is not specified, the value 50 is taken for the second argument.
4. The sum of the current year and argument-2 must be less than 10000 and greater than 1699.
5. No check is run to verify whether argument-1 is a valid date. This means that the values 0 and 999999 are also valid arguments for the DATE-TO-YYYYMMDD function if a check of the arguments is required by one of the options CHECK-FUNCTION-ARGUMENTS = YES or SET-FUNCTION-ERROR-DEFAULT = YES.

Returned values

1. The returned value is the date specified in argument-1 with a 4-digit year. For an argument of the form YYMMDD, the returned value is defined by:
   FUNCTION YEAR-TO-YYYY (YY, argument-2) * 10000 + MMDD
2. The error default value is 0.

See also: DAY-TO-YYYYDDD, YEAR-TO-YYYY

Example 12-9

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  A-DATE          PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE A-DATE = FUNCTION DATE-TO-YYYYMMDD (590123).
    DISPLAY A-DATE UPON T.                                          (1)
    COMPUTE A-DATE = FUNCTION DATE-TO-YYYYMMDD (470101 -50).
    DISPLAY A-DATE UPON T.                                          (2)
    STOP RUN.
```

A more detailed example is provided with the description of the YEAR-TO-YYYY function.

Result:

In the year 1996 the program returns the following results:
(1) 19590123
(2) 18470101

In the year 2009 the program returns the following results:
(1) 20590123
(2) 19470101

DAY-OF-INTEGER - Date conversion

The DAY-OF-INTEGER function converts a date in the Gregorian calendar from integer date form to Julian date form (YYYYDDD).
The type of this function is integer.

Format

```
FUNCTION DAY-OF-INTEGER (argument-1)
```

Arguments

1. Argument-1 is a positive integer that represents a number of days succeeding December 31, 1600, in the Gregorian calendar.

Returned values

1. The returned value represents the Julian equivalent of the integer specified in argument-1.
2. YYYY represents a year in the Gregorian calendar, and DDD represents the day of that year.
3. The error default value is 0.

See also: DATE-OF-INTEGER, INTEGER-OF-DAY, INTEGER-OF-DATE, CURRENT-DATE, WHEN-COMPILED

Example 12-10

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DAYS PIC 9999 VALUE 5000.
01  A-DATE PIC X(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE A-DATE = FUNCTION DAY-OF-INTEGER (DAYS)
    DISPLAY A-DATE UPON T.
    STOP RUN.
```

Result: 1614252
The 5000th day as of 31.12.1600 was the 252nd day of the year 1614.

DAY-TO-YYYYDDD year conversion

The DAY-TO-YYYYDDD function converts a date specified with argument-1 from the Julian date format with a 2-digit year into the Julian date format with a 4-digit year. The end of the 100-year interval in which the year specified in argument-1 falls is determined by adding argument-2 to the current year (the year in which the function is executed) ("floating window").
The type of this function is integer.

Format

```
FUNCTION DAY-TO-YYYYDDD (argument-1 [argument-2])
```

Arguments

1. argument-1 must be a positive number less than 100000.
2. argument-2, if specified, must be an integer.
3. If argument-2 is not specified, the value 50 is taken for the second argument.
4. The sum of the current year and argument-2 must be less than 10000 and greater than 1699.
5. No check is run to verify whether argument-1 is a valid date. This means that the values 0 and 99999 are also valid arguments for the DAY-TO-YYYYDDD function if a check of the arguments is required by one of the options CHECK-FUNCTION-ARGUMENTS = YES or SET-FUNCTION-ERROR-DEFAULT = YES.

Returned values

1. The returned value is the date specified in argument-1 with a 4-digit year. For an argument of the form YYDDD, the returned value is defined by:
   FUNCTION YEAR-TO-YYYY (YY, argument-2) * 1000 + DDD
2. The error default value is 0.

See also: DATE-TO-YYYYMMDD, YEAR-TO-YYYY

Example 12-11

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  A-DATE         PIC 9(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE A-DATE = FUNCTION DAY-TO-YYYYDDD (59001).
    DISPLAY A-DATE UPON T.                                          (1)
    COMPUTE A-DATE = FUNCTION DAY-TO-YYYYDDD (47365 -50).
    DISPLAY A-DATE UPON T.                                          (2)
    STOP RUN.
```

A more detailed example is provided with the description of the YEAR-TO-YYYY function.

Result:

In the year 1996 the program returns the following results:
(1) 1959001
(2) 1847365

In the year 2009 the program returns the following results:
(1) 2059001
(2) 1947365

FACTORIAL - Factorial

The FACTORIAL function returns an integer that is the factorial of argument-1.
The type of this function is integer.

Format

```
FUNCTION FACTORIAL (argument-1)
```

Arguments

1. Argument-1 must be an integer greater than or equal to zero and less than or equal to 19.

Returned values

1. If the value of argument-1 is zero, the value 1 is returned.
2. If the value of argument-1 is positive, its factorial is returned.
3. The error default value is −2.

See also: LOG, LOG10, SQRT

Example 12-12

```
...
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.
```

Result: 00005040

INTEGER - Next smaller integer

The INTEGER function returns the greatest integer that is less than or equal to the value of argument-1.
The type of this function is integer.

Format

```
FUNCTION INTEGER (argument-1)
```

Arguments

1. Argument-1 must be class numeric and must be greater than or equal to $-10^{18}+1$ and less than 10^{18}.

Returned values

1. The returned value is the greatest integer less than or equal to the value of argument-1. For example, if the value of argument-1 is −1.5, −2 is returned. If the value of argument-1 is +1.5, +1 is returned.
2. The error default value is −999’999’999’999’999’999.

See also: INTEGER-PART

Example 12-13

```
...
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.
```

Result: 00000004

INTEGER-OF-DATE - Date conversion

The INTEGER-OF-DATE function converts a date in the Gregorian calendar from standard date form (YYYYMMDD) to integer date form.
This type of this function is integer.

Format

```
FUNCTION INTEGER-OF-DATE (argument-1)
```

Arguments

1. Argument-1 must be an integer of the form YYYYMMDD, whose value is obtained from the calculation: (YYYY * 10000) + (MM * 100) + DD
   a) YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.
   b) MM represents a month and must be a positive integer less than 13.
   c) DD represents a day and must be a positive integer less than 32 provided that it is valid for the specified month and year combination.

Returned values

1. The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600, in the Gregorian calendar.
2. The error default value is 0.

See also: INTEGER-OF-DAY, DATE-OF-INTEGER, DAY-OF-INTEGER, CURRENT-DATE, WHEN-COMPILED

Example 12-14

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DAYS PIC 9(8).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE DAYS = FUNCTION INTEGER-OF-DATE (19530410).
    DISPLAY DAYS UPON T.
    STOP RUN.
```

Result: 00128665
10.4.1953 was the 128665th day as of 31.12.1600.

INTEGER-OF-DAY - Date conversion

The INTEGER-OF-DAY function converts a date in the Gregorian calendar from Julian date form (YYYYDDD) to integer date form.
The type of this function is integer.

Format

```
FUNCTION INTEGER-OF-DAY (argument-1)
```

Arguments

1. Argument-1 must be an integer of the form YYYYDDD, whose value is obtained from the calculation: (YYYY * 1000) + DDD
   a) YYYY represents the year in the Gregorian calendar. It must be an integer greater than 1600.
   b) DDD represents the day of the year. It must be a positive integer less than 367 but 366 may only be specified for a leap year.

Returned values

1. The returned value is an integer that is the number of days the date represented by argument-1 succeeds December 31, 1600, in the Gregorian calendar.
2. The error default value is 0.

See also: INTEGER-OF-DATE, DAY-OF-INTEGER, DATE-OF-INTEGER, CURRENT-DATE, WHEN-COMPILED

Example 12-15

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DAYS PIC 9(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE DAYS = FUNCTION INTEGER-OF-DAY (1993299).
    DISPLAY DAYS UPON T.
    STOP RUN.
```

Result: 0143474
The 299th day of the year 1993 is the 143474th day as of 31.12.1600.

INTEGER-PART - Integer part of a floating-point value

The INTEGER-PART function returns an integer that is the integer portion of argument-1. The type of this function is integer.

Format

```
FUNCTION INTEGER-PART (argument-1)
```

Arguments

1. Argument-1 must be class numeric and must be greater than -10^{18} and less than 10^{18}.

Returned values

1. If the value of argument-1 is zero, the returned value is zero.
2. If the value of argument-1 is positive, the returned value is the greatest integer less than or equal to the value of argument-1. For example, if the value of argument-1 is +1.5, +1 is returned.
3. If the value of argument-1 is negative, the returned value is the least integer greater than or equal to the value of argument-1. For example, if the value of argument-1 is −1.5, −1 is returned.
4. The error default value is −999’999’999’999’999’999.

See also: INTEGER

Example 12-16

```
...
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.
```

Result: -00000003

LENGTH - Number of characters

The LENGTH function returns an integer equal to the length of argument-1 in character positions.
The type of this function is integer.

Format

```
FUNCTION LENGTH (argument-1)
```

Arguments

1. Argument-1 may be a nonnumeric literal or a data item of any class or category.
2. If argument-1 or any data item subordinate to argument-1 is described with the DEPENDING phrase of the OCCURS clause, the contents of the data item referenced by the data-name specified in the DEPENDING phrase are used at the time the LENGTH function is evaluated.

Returned values

1. If argument-1 is a nonnumeric literal or an elementary data item or a group data item that does not contain a variable occurrence data item, the value returned is an integer equal to the length of argument-1 in character positions.
2. If argument-1 is a group data item containing a variable occurrence data item, the returned value is an integer determined by evaluation of the data item specified in the DEPENDING phrase of the OCCURS clause for that variable occurrence data item. This evaluation is accomplished according to the rules in the OCCURS clause dealing with the data item as a sending data item (see OCCURS clause and USAGE clause).
3. The returned value includes implicit FILLER characters, if any.

See also: REVERSE

Example 12-17

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC 9(3).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE RES = FUNCTION LENGTH ("abbreviation").
    DISPLAY RES UPON T.
    STOP RUN.
```

Result: 012

LOG - Logarithm

The LOG function returns a numeric value that approximates the logarithm to the base e (natural log) of argument-1.
The type of this function is numeric.

Format

```
FUNCTION LOG (argument-1)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than zero.

Returned values

1. The returned value is the approximation of the logarithm to the base e of argument-1.
2. The error default value is −999’999’999’999’999’999.

See also: LOG10, FACTORIAL, SQRT

Example 12-18

```
...
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.
```

Result: 01.098612

LOG10 - Logarithm of base 10

The LOG10 function returns a numeric value that approximates the logarithm to the base 10 of argument-1.
The type of this function is numeric.

Format

```
FUNCTION LOG10 (argument-1)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be greater than zero.

Returned values

1. The returned value is the approximation of the logarithm to the base 10 of argument-1.
2. The error default value is −999’999’999’999’999’999.

See also: LOG, FACTORIAL, SQRT

Example 12-19

```
...
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.
```

Result: 00.477121

LOWER-CASE - Lowercase letters

The LOWER-CASE function returns a character string that is the same length as argument-1 with each uppercase letter replaced by a corresponding lowercase letter.
The type of this function is alphanumeric.

Format

```
FUNCTION LOWER-CASE (argument-1)
```

Arguments

1. Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

Returned values

1. The returned value is the same character string as argument-1, except that each uppercase letter is replaced by the corresponding lowercase letter.
2. The character string returned has the same length as argument-1.
3. Umlauts are not converted.
4. The error default value is a space.

See also: UPPER-CASE

Example 12-20

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC X(20).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION LOWER-CASE ("SIEMENS NIXDORF") TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Result: siemens nixdorf

MAX - Value of maximum argument

The MAX function returns the content of the argument that contains the maximum value. The type of this function depends upon the argument types as follows:

Argument type	Function type
alphabetic	alphanumeric
alphanumeric	alphanumeric
all arguments integer	integer
numeric	numeric
numeric/integer	numeric

Format

```
FUNCTION MAX ({argument-1}...)
```

Arguments

1. If more than one argument is specified, all arguments must be of the same class.

Returned values

1. The returned value is the content of the argument having the greatest value. The comparisons used to determine the greatest value are made according to the rules for simple conditions.
2. If more than one argument has the same greatest value, the content of the argument returned is the leftmost argument having that value.
3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument.
4. The error default value is 0.

See also: MIN, ORD-MAX, ORD-MIN, RANGE, MEAN, MEDIAN, MIDRANGE, SUM

Example 12-21

```
...
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 "Argument with greatest value: " RES UPON T.
    DISPLAY "Argument with greatest value: " RES1 UPON T.
    STOP RUN.
```

Result: Argument with greatest value RES: 032
Argument with greatest value RES1: THEO

MEAN - Arithmetic mean of arguments

The MEAN function returns a numeric value that is the arithmetic mean (average) of its arguments.
The type of this function is numeric.

Format

```
FUNCTION MEAN ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the arithmetic mean of the argument series.
2. The returned value is defined as the sum of the argument series divided by the number of arguments specified.
3. The error default value is 0.

See also: MIN, MAX, RANGE, MEDIAN, MIDRANGE, SUM

Example 12-22

```
...
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.
```

Result: 013.833

MEDIAN - Median of arguments

The MEDIAN function returns the contents of the argument whose value is the middle value in the list formed by arranging the arguments in sorted order.
The type of this function is numeric.

Format

```
FUNCTION MEDIAN ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the content of the argument having the middle value in the list formed by arranging all the argument values in sorted order.
2. If the number of occurrences referenced by argument is odd, the returned value is such that at least half of the occurrences referenced by argument are greater than or equal to the returned value and at least half are less than or equal. If the number of occurrences referenced by argument is even, the returned value is the arithmetic mean of the values referenced by the two middle occurrences.
3. The comparisons used to arrange the argument values in sorted order are made according to the rules for simple conditions (see section 3.9.4).
4. The error default value is 0.

See also: MIN, MAX, RANGE, MEAN, MIDRANGE, SUM

Example 12-23

```
...
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.
```

Result: 032

MIDRANGE - Mean of minimum and maximum arguments

The MIDRANGE function returns a numeric value that is the arithmetic mean (average) of the values of the minimum argument and the maximum argument.
The type of this function is numeric.

Format

```
FUNCTION MIDRANGE ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the arithmetic mean of the greatest argument value and the least argument value. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions (see section 3.9.4).
2. The error default value is 0.

See also: MIN, MAX, RANGE, MEAN, MEDIAN, SUM

Example 12-24

```
...
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.
```

Result: 018.500

MIN - Value of minimum argument

The MIN function returns the content of the argument that contains the minimum value. The type of this function depends upon the argument types as follows:

Argument type	Function type
alphabetic	alphanumeric
alphanumeric	alphanumeric
all arguments integer	integer
numeric	numeric
numeric/integer	numeric

Format

```
FUNCTION MIN ({argument-1}...)
```

Arguments

1. If more than one argument is specified, all arguments must be of the same class.

Returned values

1. The returned value is the content of the argument having the least value. The comparisons used to determine the least value are made according to the rules for simple conditions.
2. If more than one argument has the same least value, the content of the argument returned is the leftmost argument having that value.
3. If the type of the function is alphanumeric, the size of the returned value is the same as the size of the selected argument.
4. The error default value is 0.

See also: MAX, RANGE, MEAN, MEDIAN, MIDRANGE, SUM

Example 12-25

```
...
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.
```

Result: 005

MOD - Modulo

The MOD function returns an integer value that is argument-1 modulo argument-2. The type of this function is integer.

Format

```
FUNCTION MOD (argument-1 argument-2)
```

Arguments

1. Argument-1 and argument-2 must be integers.
2. The value of argument-2 must not be zero.
3. The value of argument-1 must be >= $-10^{18}+1$ and < 10^{18}.

   If the value of argument-1 is outside this range, the function value can be calculated but is not guaranteed to be correct.

Returned values

1. The returned value is argument-1 modulo argument-2. The returned value is defined as:
   argument-1 − (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))
2. The error default value is −999’999’999’999’999’999.

See also: REM

Example 12-26

argument-1	argument-2	Returned value
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

NUMVAL - Numeric value of string

The NUMVAL function returns the numeric value represented by the character string specified by argument-1. Leading and trailing spaces are ignored.
The type of this function is numeric.

Format

```
FUNCTION NUMVAL (argument-1)
```

Arguments

1. Argument-1 must be a nonnumeric literal or an alphanumeric data item whose content has one of the following two formats::

```
[␣] [+] [␣] {digits[.[digits]]} [␣]
    [-]     {.digits          }
oder
[␣] {digits[.[digits]]} [␣] [+ ]
    {.digits          }     [- ]
                            [CR]
                            [DB]
```

␣	String of one or more spaces
digits	String of one to 18 digits

2. The total number of digits in argument-1 must not exceed 18.
3. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, a comma must be used in argument-1 rather than a decimal point.

Returned values

1. The returned value is the numeric value represented by argument-1.
2. The error default value is −999’999’999’999’999’999.

See also: NUMVAL-C

Example 12-27

```
...
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.
```

Result: 15.00

NUMVAL-C - Numeric value of string with optional currency sign

The NUMVAL-C function returns the numeric value represented by the character string specified by argument-1. Any optional currency sign specified by argument-2 and any optional commas preceding the decimal point are ignored.
The type of this function is numeric.

Format

```
FUNCTION NUMVAL-C (argument-1 [argument-2])
```

Arguments

1. Argument-1 must be a nonnumeric literal or an alphanumeric data item whose content has one of the following two formats: :

```
[␣] [+]          {digits[,digits...[.[digits]]]}
    [ ] [␣] [cs] [␣] {                          } [␣]
    [-]          {.digits                       }
```

or

```
                 {digits[,digits]...[.[digits]]}     [+ ]
[␣] [cs] [␣]     {                              } [␣] [- ] [␣]
                 {.digits                       }     [CR]
                                                      [DB]
```

␣	String of one or more spaces
wz	Currency symbol: string of one or more characters (argument-2)
digits	String of one to 18 digits

2. If the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the comma and decimal point in argument-1 are reversed.
3. The total number of digits in argument-1 must not exceed 18.
4. Argument-2, if specified, must be a nonnumeric literal or an alphanumeric data item.
5. If argument-2 is not specified, the character used for cs is the currency symbol specified for the program.

Returned values

1. The returned value is the numeric value represented by argument-1.
2. The error default value is −999’999’999’999’999’999.

See also: NUMVAL

Example 12-28

```
...
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.
```

Result: 15.00

ORD - Ordinal position in collating sequence

The ORD function returns an integer value that is the ordinal position of argument-1 in the collating sequence for the program. The lowest ordinal position is 1.
The type of this function is integer.

Format

```
FUNCTION ORD (argument-1)
```

Arguments

1. Argument-1 must be one character in length and must be class alphabetic or alphanumeric.

Returned values

1. The returned value is the ordinal position of argument-1 in the collating sequence for the program.

See also: CHAR

Example 12-29

```
...
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.
```

Result: 234
The ordinal position of the letter Z in EBCDIC is 234.

ORD-MAX - Ordinal position of maximum argument

The ORD-MAX function returns an integer value that indicates which of the specified arguments, seen from left to right, contains the maximum value.
The type of this function is integer.

Format

```
FUNCTION ORD-MAX ({argument-1}...)
```

Arguments

1. If more than one argument is specified, all arguments must be of the same class.

Returned values

1. The returned value is the ordinal number that indicates the position of the argument having the greatest value in the argument series.
2. The comparisons used to determine the greatest valued argument are made according to the rules for simple conditions (see section 3.9.4).
3. If more than one argument has the same greatest value, the number returned corresponds to the position of the leftmost argument having that value.
4. The error default value is 0.

See also: ORD-MIN, MAX, MIN

Example 12-30

```
...
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.
```

Result: 004
The fourth argument (18) has the greatest value.

ORD-MIN - Ordinal position of minimum argument

The ORD-MIN function returns an integer value that indicates which of the specified arguments, seen from left to right, contains the minimum value.
The type of this function is integer.

Format

```
FUNCTION ORD-MIN ({argument-1}...)
```

Arguments

1. If more than one argument is specified, all arguments must be of the same class.

Returned values

1. The returned value is the ordinal number that indicates the position of the argument having the last value in the argument series.
2. The comparisons used to determine the least valued argument are made according to the rules for simple conditions (see section 3.9.4).
3. If more than one argument has the same least value, the number returned corresponds to the position of the leftmost argument having that value.
4. The error default value is 0.

See also: ORD-MAX, MAX, MIN

Example 12-31

```
...
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.
```

Result: 004
The fourth argument ("?") has the least value.

PRESENT-VALUE - Present value (period-end amount)

The PRESENT-VALUE function returns the principal amount repaid by a series of deferred payments. The deferred payments are given in argument-2; the interest rate at which the payments are calculated is specified in argument-1.
The type of this function is numeric.

Format

```
FUNCTION PRESENT-VALUE (argument-1 {argument-2}...)
```

Arguments

1. Argument-1 and argument-2 must be class numeric.
2. The value of argument-1 must be greater than −1.

Returned values

1. The returned value is the approximation of a summation of a series of calculations with each term in the following form:

   argument-2 / (1 + argument-1) ** n

   The exponent n is incremented from one by one for each term in the series.
2. The error default value is −999’999’999’999’999’999.

See also: ANNUITY

Example 12-32

```
 ...
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*** Interest rate 10% **********
 01  INTEREST PIC 9V99 VALUE 0.10.
*** Four payments *********
 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.
*** Principal amount repaid ***
 01  PAR PIC 9(6).
 PROCEDURE DIVISION.
 P1 SECTION.
 MAIN.
     COMPUTE PAR = FUNCTION PRESENT-VALUE (INTEREST B-1 B-2 B-3 B-4).
     DISPLAY "Amount repaid: " PAR UPON T.
     STOP RUN.
```

Result: Amount repaid: 003996
This is the principal amount repaid by 4 payments.

RANDOM - Random number

The RANDOM function returns a numeric value that is a pseudo-random number from a rectangular distribution.
The type of this function is numeric.

Format

```
FUNCTION RANDOM [(argument-1)]
```

Arguments

1. If argument-1 is specified, it must be zero or a positive integer. It is used as the seed value to generate a sequence of pseudo-random numbers.
2. If a subsequent reference specifies argument-1, a new sequence of pseudo-random numbers is started.
3. If the first reference to this function in the run unit does not specify argument-1, the seed value is 0.
4. In each case, subsequent references without specifying argument-1 return the next number in the current sequence.

Returned values

1. The returned value is greater than or equal to zero and less than one.
2. For a given seed value, the sequence of pseudo-random numbers will always be the same.
3. The range of argument-1 values that will yield distinct sequences of pseudo-random numbers is 0 through $2^{31}-1$.
4. The error default value is −1.

Example 12-33

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. LOTTO.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     TERMINAL IS VIDEO.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  LIST.
  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-O        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.
*
* Select a seed for the random function based on the
* time, date, and weekday
*
     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
*
* Compute the first random number
*
     COMPUTE RAN-VAL = FUNCTION RANDOM (INIT-ARG)
*
* Traverse the loop until 6 elements have been entered
* into the list
*
     PERFORM VARYING LI FROM 1 BY 1 UNTIL LI > 6
*
* Traverse the loop until a unique number is found
* and entered into the current list element
*
       PERFORM UNTIL Z (LI) NOT ZERO
*
* Map the return value of the RANDOM function
* to an integer between 1 and 49
*
```

```
            COMPUTE Z-O = FUNCTION INTEGER (49 * RAN-VAL) + 1
            SET SI TO 1
      *
      * Check the result for uniqueness
      *
            SEARCH ELEM
      *
      * If number not found in list -> enter the number
      *
            AT END MOVE Z-O TO Z (LI)
      *
      * If number is already in the list -> new random number
      *
              WHEN Z (SI) = Z-O CONTINUE
            END-SEARCH
      *
      * Compute next random number
      *
            COMPUTE RAN-VAL = FUNCTION RANDOM
          END-PERFORM
        END-PERFORM
        SORT ELEM ASCENDING Z
        MOVE "." TO T (6)
        DISPLAY LISTE UPON VIDEO
        STOP RUN.
```

Result: 6 numbers from 1 to 49

RANGE - Difference value

The RANGE function returns a value that is equal to the value of the maximum argument minus the value of the minimum argument.
The type of this function depends upon the argument types as follows:

Argument type	Function type
all arguments integer all arguments numeric mix of integer/numeric	integer numeric numeric

Format

```
FUNCTION RANGE ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is equal to the greatest value of argument-1 minus the least value of argument-1. The comparisons used to determine the greatest and least values are made according to the rules for simple conditions (see 3.9.4, "Simple conditions").
2. The error default value is −1.

See also: MIN, MAX, MEAN, MEDIAN, MIDRANGE, ORD-MAX, ORD-MIN, SUM

Example 12-34

```
...
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.
```

Result: 027

REM - Remainder

The REM function returns a numeric value that is the remainder of argument-1 divided by argument-2.
The type of this function is numeric.

Format

```
FUNCTION REM (argument-1 argument-2)
```

Arguments

1. Argument-1 and argument-2 must be class numeric.
2. The value of argument-2 must not be zero.
3. Argument-1 and argument-2 must be greater than -10^{18} and less than 10^{18}.

Returned values

1. The returned value is the remainder of argument-1 / argument-2. It is defined as the expression:

   argument-1 − (argument-2 * FUNCTION INTEGER-PART (argument-1 / argument-2))

2. The error default value is −999’999’999’999’999’999.

See also: MOD

Example 12-35

```
...
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.
```

Result: 004

REVERSE - Reverse order of string characters

The REVERSE function returns a character string of exactly the same length as argument-1 and whose characters are exactly the same as those of argument-1, except that they are in reverse order.
The type of this function is alphanumeric.

Format

```
FUNCTION REVERSE (argument-1)
```

Arguments

1. Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

Returned values

1. If argument-1 is a character string of length n, the returned value is a character string of length n such that for 1 <= j <= n, the character in position j of the returned value is the character from position n − j + 1 of argument-1.
2. The error default value is a space.

See also: LENGTH

Example 12-36

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  REV PIC X(14) VALUE "dog ma i,amgod".
01  RES PIC X(14).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION REVERSE (REV) TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Result: dogma,i am god

SIN - Sine

The SIN function returns the sine of the angle or arc that is specified in radians by argument-1.
The type of this function is numeric.

Format

```
FUNCTION SIN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the approximation of the sine of argument-1 and is greater than or equal to −1 and less than or equal to +1.

See also: ASIN, COS, ACOS, TAN, ATAN

Example 12-37

```
...
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.
```

Result: −0.0009073462

SQRT - Square root

The SQRT function returns a numeric value that approximates the square root of argument-1.
The type of this function is numeric.

Format

```
FUNCTION SQRT (argument-1)
```

Arguments

1. Argument-1 must be class numeric.
2. The value of argument-1 must be zero or positive.

Returned values

1. The returned value is the absolute value of the approximation of the square root of argument-1.
2. The error default value is −2.

See also: FACTORIAL, LOG, LOG10

Example 12-38

```
...
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.
```

Result: 05.744562

STANDARD-DEVIATION - Standard deviation of arguments

The STANDARD-DEVIATION function returns a numeric value that approximates the standard deviation of its arguments.
The type of this function is numeric.

Format

```
FUNCTION STANDARD-DEVIATION ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the approximation of the standard deviation of the argument series.
2. The returned value is calculated as follows:
   a) The difference between each argument value and the arithmetic mean of the argument series is calculated and squared.
   b) The values obtained are then added. This quantity is divided by the number of values in the argument series.
   c) The square root of the quotient obtained is then calculated. The returned value is the absolute value of this square root.
3. If the argument series consists of only one value, or if the argument series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.
4. The error default value is −1.

See also: VARIANCE

Example 12-39

```
...
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.
```

Result: 01.6329

SUM - Sum of arguments

The SUM function returns a value that is the sum of all the arguments.

The type of this function depends upon the argument types as follows:

Argument type	Function type
all arguments integer all arguments numeric mix of integer/numeric	integer numeric numeric

Format

```
FUNCTION SUM ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the sum of all the arguments.
2. The error default value is 0.

See also: MAX, MIN, RANGE, MEAN, MEDIAN, MIDRANGE

Example 12-40

```
...
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.
```

Result: 00000083

TAN - Tangent

The TAN function returns the tangent of the angle or arc that is specified in radians by argument-1.
The type of this function is numeric.

Format

```
FUNCTION TAN (argument-1)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the approximation of the tangent of argument-1.
2. The error default value is −999’999’999’999’999’999.

See also: ATAN, SIN, ASIN, COS, ACOS

Example 12-41

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  R PIC S9V9(10).
01  RES PIC -9.9(10).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    COMPUTE R = FUNCTION TAN (3.1425).
    MOVE R TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Result: 0.0009073466

UPPER-CASE - Uppercase letters

The UPPER-CASE function returns a character string that is the same length as argument-1 with each lowercase letter replaced by the corresponding uppercase letter. The type of this function is alphanumeric.

Format

```
FUNCTION UPPER-CASE (argument-1)
```

Arguments

1. Argument-1 must be class alphabetic or alphanumeric and must be at least one character in length.

Returned values

1. The same character string as argument-1 is returned, except that each lowercase letter is replaced by the corresponding uppercase letter.
2. The character string returned has the same length as argument-1.
3. Umlauts are not converted.
4. The error default value is a space.

See also: LOWER-CASE

Example 12-42

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RES PIC X(20).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION UPPER-CASE ("wonderwoman") TO RES.
    DISPLAY RES UPON T.
    STOP RUN.
```

Result: WONDERWOMAN

VARIANCE - Variance of arguments

The VARIANCE function returns a numeric value that approximates the variance of its arguments.
The type of this function is numeric.

Format

```
FUNCTION VARIANCE ({argument-1}...)
```

Arguments

1. Argument-1 must be class numeric.

Returned values

1. The returned value is the approximation of the variance of the argument series.
2. The returned value is defined as the square of the standard deviation of the argument series (see STANDARD-DEVIATION function).
3. If the argument series consists of only one value, or if the argument series consists of all variable occurrence data items and the total number of occurrences for all of them is one, the returned value is zero.
4. The error default value is −1.

See also: STANDARD-DEVIATION

Example 12-43

```
...
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.
```

Result: 02.6666

WHEN-COMPILED - Date and time of compilation

The WHEN-COMPILED function returns the date and time the program was compiled. The type of this function is alphanumeric.

Format

```
FUNCTION WHEN-COMPILED
```

Returned values

1. The character positions returned, numbered from left to right, are:

Character position	Contents
1-4	Four numeric digits of the year (Gregorian calendar).
5-6	Two numeric digits of the month of the year, in the range 01 through 12.
7-8	Two numeric digits of the day of the month, in the range 01 through 31.
9-10	Two numeric digits of the hours past midnight, in the range 00 through 23.
11-12	Two numeric digits of the minutes past the hour, in the range 00 through 59.
13-14	Two numeric digits of the seconds past the minute, in the range 00 through 59.
15-21	0000000

See also: CURRENT-DATE, DATE-OF-INTEGER, DAY-OF-INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY

Example 12-44

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  A-DATE PIC X(21).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
    MOVE FUNCTION WHEN-COMPILED TO A-DATE.
    DISPLAY A-DATE UPON T.
    STOP RUN.
```

Result: 199604211434270000000

YEAR-TO-YYYY year conversion

The YEAR-TO-YYYY function converts a 2-digit year into a 4-digit year. The end of the 100-year interval in which the year specified in argument-1 falls is determined by adding argument-2 to the current year (the year in which the function is executed) ("floating window"). The type of this function is integer.

Format

```
FUNCTION YEAR-TO-YYYY (argument-1 [argument-2])
```

Arguments

1. argument-1 must be a positive number less than 100.
2. argument-2, if specified, must be an integer.
3. If argument-2 is not specified, the value 50 is taken for the second argument.
4. The sum of the current year and argument-2 must be less than 10000 and greater than 1699.

Returned values

1. The returned value is the year specified in argument-1 together with the century. The returned value depends on the intermediate value

   $L_1L_2L_3L_4$ = current year +argument-2 (= last year of the 100-year interval).
   The century is calculated as follows:

$100 * L_1L_2 + YY$	if $L_3L_4 >= YY$ (YY=argument-1)
$100 * (L_1L_2 -1) + YY$	if $L_3L_4 < YY$

2. The error default value is 0.

See also: DATE-TO-YYYYMMDD, DAY-TO-YYYYDDD

Example 12-45

```
...
DATA DIVISION.
WORKING-STORAGE SECTION.
01  A-DATE            PIC 9(7).
01  CURRENT-YEAR      PIC 9(7).
01  YEAR              PIC 9(7).
PROCEDURE DIVISION.
P1 SECTION.
MAIN.
*
* Calculation of the function with a floating window:
*
* The 100-year interval in which the calculated year falls is to
* include the years (current year -35) through (current year +64):
*
    COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (59 64).
    DISPLAY A-DATE UPON T.                                          (1)
*
* Without 2nd argument (or 2nd argument =50)
* The 100-year interval includes the years (current year -49)
* through (current year +50):
*
    COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (0).
    DISPLAY A-DATE UPON T.                                          (2)
*
* The 2nd argument can also be negative
* The 100-year interval includes the years (current year -109)
* through (current year -10):
*
    COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (96 -10).
    DISPLAY A-DATE UPON T.                                          (3)
*
* Calculation of the function with a fixed window
*
* The 100-year interval in which the calculated year falls is to
* include the years 1950 through 2049:
*
* Calculation of the last year of the 100-year interval
* relative to the current year
*
    MOVE FUNCTION CURRENT-DATE(1:4) TO CURRENT-YEAR.
    COMPUTE YEAR = 2049 - CURRENT-YEAR.
* Calculation of the function values
    COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (50 YEAR).
    DISPLAY A-DATE UPON T.                                          (4)
    COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (1 YEAR).
    DISPLAY A-DATE UPON T.                                          (5)
```

```
      *
      * The 100-year interval in which the calculated year falls is to
      * include the years 1890 through 1989:
      *
      * Calculation of the last year of the 100-year interval
      * relative to the current year
      *
          MOVE FUNCTION CURRENT-DATE(1:4) TO CURRENT-YEAR.
          COMPUTE YEAR = 1989 - CURRENT-YEAR.
      * Calculation of the function values
        COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (89 YEAR).
        DISPLAY A-DATE UPON T.                                          (6)
        COMPUTE A-DATE = FUNCTION YEAR-TO-YYYY (90 YEAR).
        DISPLAY A-DATE UPON T.                                          (7)
        STOP RUN.
```

Result:

In the year 1996 the program returns the following results:
(1) 2059
(2) 2000
(3) 1896
(4) 1950
(5) 2001
(6) 1989
(7) 1890

In the year 2050 the program returns the following results:
(1) 2059
(2) 2100
(3) 1996
(4) 1950
(5) 2001
(6) 1989
(7) 1890

Related publications

[1] **COBOL85** (BS2000/OSD)
COBOL Compiler
User's Guide

Target group
COBOL users of BS2000 and POSIX

Contents
- Generation of the COBOL85 compiler and the software required for the linking, loading and debugging of COBOL programs
- File processing with COBOL programs
- Inter-program communication
- COBOL85 and POSIX
- Structure of the COBOL85 system
- Compiler messages and runtime system messages

[2] **CRTE** (BS2000/OSD)
Common RunTime Environment
User Guide

Target group
Programmers and system administrators in a BS2000 environment

Contents
Description of the common runtime environment for COBOL85, C and C++ objects and for "language mixes":
- CRTE components
- ILCS program communication interface
- linkage examples

[3] **AID** (BS2000/OSD)
Advanced Interactive Debugger
Core Manual
User Guide

Target group
Programmers in BS2000

Contents
– Overview of the AID system
– Description of facts and operands which are the same for all programming languages
– Messages
– Comparison between AID and IDA

Applications
Testing of programs in interactive or batch mode

[4] **AID** (BS2000/OSD)
Advanced Interactive Debugger
Debugging of COBOL Programs
User Guide

Target group
COBOL programmers

Contents
– Description of the AID commands for symbolic debugging of COBOL programs
– Sample application

Applications
Testing of COBOL programs in interactive or batch mode

[5] **AID** (BS2000/OSD)
Advanced Interactive Debugger
Debugging on Machine Code Level
User Guide

Target group
Programmers in BS2000

Contents
– Description of the AID commands for debugging on machine code level
– Sample application

Applications
Testing of programs in interactive or batch mode

[6] **UDS/SQL** (BS2000/OSD)
Application Programming
User Guide

Target group
Programmers

Contents
– Transaction concept
– Use of the currency table
– COBOL DML
– CALL DML
– Testing of DML functions

[7] **UTM** (TRANSDATA)
Programming Applications
User's Guide

Target group
Programmers of UTM applications

Contents
– Language-independent description of the KDCS program interface
– Structure of UTM programs
– KDCS calls
– Testing UTM applications
– All the information required by programmers of UTM applications

Applications
BS2000 transaction processing

UTM (TRANSDATA)
Supplement for COBOL
User's Guide

Target group
Programmers of UTM COBOL applications

Contents
– Translation of the KDCS program interface into the COBOL language
– All the information required by programmers of UTM COBOL applications

Applications
BS2000 transaction processing

[8] **SORT** (BS2000/OSD)
SDF Format

Target group
– BS2000 users
– Programmers

Contents
This manual describes the principles, functions and statements for sorting and merging data records (SDF format), calling the subroutine interface and the SORTZM access method. A chapter with examples instructs newcomers how to use SORT.

[9] **BS2000/OSD-BC V2.0**
Introductory Guide to DMS
User Guide

Target group
All BS2000/OSD users

Contents
File processing in BS2000: files, file and catalog management, file protection, files and data media, file and data protection, OPEN, CLOSE and EOV processing, DMS access methods (SAM, ISAM).

[10] **EDT V16.5A** (BS2000/OSD)
Statements
User Guide

Target group
EDT newcomers and EDT users

Contents
Processing of SAM and ISAM files and elements from program libraries and POSIX files.

[11] **COB85** (SINIX)
COBOL Compiler
Language Reference Manual

Target group
COBOL users in a SINIX environment

Contents
The manual contains a description of the COBOL language for the COB85 compiler. This compiler supports all the required modules of the ANS85 standard, as well as the optional modules "Report Writer", "Segmentation" and "Screen Handling".

[12] **COBOL85** (BS2000/OSD)
COBOL Compiler
Ready Reference

Target group
COBOL users in BS2000

Contents
Tabular overviews of the language set and control of the COBOL85 compiler.

[13] **System Interfaces for Applications**
C, COBOL, FORTRAN
Language Overview
User Guide

Target group
Software developers

Contents
– Tables of SIA language elements for the programming languages C, COBOL and FORTRAN
– Comparison with SAA, language standards and compilers in BS2000 and SINIX

Ordering manuals

The manuals listed above and the corresponding order numbers can be found in the Siemens Nixdorf *List of Publications*. New publications are described in the *Druckschriften-Neuerscheinungen (New Publications)*.

You can arrange to have both of these sent to you regularly by having your name placed on the appropriate mailing list. Please apply to your local office, where you can also order the manuals.

Index

- (minus)
    arithmetic operator 215
    PICTURE symbol 168

$, PICTURE symbol 168
* (asterisk)
    arithmetic operator 215
    comment line 105
    PICTURE symbol 168
**, arithmetic operator 215
+ (plus)
    arithmetic operator 215
    PICTURE symbol 168
/ (slash)
    arithmetic operator 215
    comment line 105
    PICTURE symbol 168

0, PICTURE symbol 168
66 level description entry (RENAMES) 145
77 level description entry 144
88 level description entry (condition name) 145
9, PICTURE symbol 167

A
A, PICTURE symbol 167
ACCEPT statement 246
    X/OPEN formats 249, 261
access mode 7
ACCESS MODE clause 364
    indexed file organization 488
    relative file organization 438
    sequential organization 364
actual decimal point 7

ADD CORRESPONDING statement 253
ADD statement 251
    arithmetic statements 236
    CORRESPONDING phrase 239
ADVANCING phrase 420
AFTER phrase 286
    INSPECT statement 286
    PERFORM statement 315
    USE statement 411, 417, 473, 522
    WRITE statement 420
algebraic signs 71
alignment of data 72, 73
    ACCEPT statement 246
    MOVE statement 294
    RECORD clause 390
ALL 286
    INSPECT statement 286
    SEARCH statement 330
ALL literal 53
    DISPLAY statement 259
    INSPECT statement 286
    STOP statement 340
    STRING statement 341
    UNSTRING statement 348
ALPHABET clause 128
ALPHABETIC 220
alphabetic character 7
alphabetic data items 68, 169
ALPHABETIC-LOWER 220
ALPHABETIC-UPPER 220
alphabet-name 7, 48
    CODE-SET clause 382
    MERGE statement 655
    OBJECT-COMPUTER paragraph 124
    SORT statement 663
    SPECIAL-NAMES paragraph 125
alphanumeric character 7
alphanumeric data items 169
alphanumeric edited items 71, 172
alphanumeric function 8
alphanumeric items 70
alphanumeric move 299
ALSO phrase 125, 267

ALTER statement 254
GO TO statement 278
segmentation 637
alternate record key 8
ALTERNATE RECORD KEY clause 489
ALTERNATE RECORD KEY clause 489
READ statement 515
RECORD KEY clause 493
REWRITE statement 517
START statement 520
WRITE statement 524
American National Standard 1
AND operator 230
ANSI 1
ANY phrase 267
area A 8, 102
area B 8, 102
argument 8
ARGUMENT-NUMBER 125, 249, 261
ARGUMENT-VALUE 125, 250
arithmetic expression 8, 215
COMPUTE statement 255
for subscripting 93
relation condition 223
sign condition 229
valid symbol combinations 216
arithmetic operator 8
arithmetic operators 215
arithmetic statements 100, 236
ascending key 9
ASCENDING KEY phrase 155
MERGE statement 655
OCCURS clause 155
SEARCH statement 330
SORT statement 663
ASCENDING phrase, OCCURS clause 158
ASSIGN clause 362
indexed file organization 487
relative file organization 437
sequential organization 362
sorting and merging 652
assumed decimal point 9

asterisk (*)
comment line 105
PICTURE symbol 168
at end condition 9
AT END phrase 97, 406
conditional statements 97
READ statement 406, 463, 512
RETURN statement 661
SEARCH statement 325, 330
AT END-OF-PAGE phrase, WRITE statement 420

B
B, PICTURE symbol 167
BEFORE phrase 286
INSPECT statement 286
USE statement 411
WRITE statement 420
BEGINNING phrase 411
binary arithmetic operator 215
BINARY phrase 192, 195
binary search 9
blank lines 9, 104
BLANK WHEN ZERO clause 148
PICTURE clause 164
USAGE IS INDEX clause 201
VALUE clause 204
block 9, 65
BLOCK CONTAINS clause 379
indexed file organization 501
relative file organization 449
sequential organization 379
block size 379, 449, 501
body group 9
BOTTOM 386
braces 44
brackets 44
BY CONTENT phrase 551
BY phrase 265
COPY statement 686
DIVIDE statement 265
INITIALIZE statement 283
INSPECT statement 286
MULTIPLY statement 301

PERFORM statement 315
REPLACE statement 692, 695
BY REFERENCE phrase 551
BY VALUE operand 554

C

C01 through C08 126
C10 through C11 126
CALL statement 551
CANCEL statement 561
ENTRY statement 564
EXIT PROGRAM statement 566
LINKAGE SECTION 542
Procedure Division header 549
segmentation 638
called program 10, 638
calling program 10
CANCEL statement 561
CALL statement 551
EXIT PROGRAM statement 566
CBL-CTR special register 52, 631
CF 618
CH 618
character 10
character set 11, 46
CHARACTERS 124
BLOCK CONTAINS clause 379, 449, 501
INSPECT statement 286
OBJECT-COMPUTER paragraph 124
RECORD clause 390, 452, 504, 653
SPECIAL-NAMES paragraph 125
CHARACTERS BY 286
character-string 10
CLASS clause 135
class condition 10, 220
class-name 10, 48, 125
clause 10
CLOCK-UNITS, RERUN clause 372, 444, 496
CLOSE statement 397
FILE STATUS clause 365, 440, 491
indexed file organization 508
I-O status 356, 431, 480
OPEN statement 402, 460, 511
READ statement 406, 463, 512

relative file organization 457
sequential organization 397
TERMINATE statement 625
COBOL character set 11, 46
COBOL language set 1
COBOL program 112
processing 107
structure 112
COBOL special register 51
COBOL statement 210, 212
COBOL word 11
COBOL words 43, 48
CODE 578
CODE-SET clause 382
sequential organization 382
collating sequence 11
COLLATING SEQUENCE clause 124
COLLATING SEQUENCE phrase 655
MERGE statement 655
SORT statement 663
column 11
COLUMN clause 593
combined condition 12, 230
comma
DECIMAL-POINT IS COMMA clause 125, 137
subscript 92
comma (,)
PICTURE symbol 168
comment entry 12, 64, 117
comment lines 12, 105
debugging lines 353
Common program 12
COMP(UTATIONAL) phrase 192, 195
COMP(UTATIONAL)-1 phrase 192, 196
COMP(UTATIONAL)-2 phrase 192, 197
COMP(UTATIONAL)-3 phrase 192, 198
COMP(UTATIONAL)-5 phrase 192, 195
compile time 12
compile time switch 353
compiler directing statements 12, 98, 101
compiler directive 695
compiler-directing sentences 98
COMPILER-INFO 126
complex condition 12, 218

COMPUTE statement 255
arithmetic statements 236
computer-independent data description 64
computer-name 12, 50
condition 13, 218
class condition 220
complex condition 218, 230
condition-name condition 219
IF statement 280
implied subjects and relational operators 233
PERFORM UNTIL statement 311, 315
relation condition 223
SEARCH statement 325, 330
sign condition 229
simple condition 218
switch-status condition 222
valid symbol combinations 231
condition name 87
conditional expression 13
conditional sentences 97
conditional statements 13, 97, 100
conditional variable 13
condition-name 13, 48
data description entry 145, 204
indexed 82
level-number 88 145
qualification 77
REDEFINES clause 179
RERUN clause 372, 444, 496
SEARCH statement 330
SET statement 338
SPECIAL-NAMES paragraph 125
subscripted 80
VALUE clause 204
condition-name condition 13, 219
CONFIGURATION SECTION 122
connective 13
CONSOLE 126
contained program 527
containing program 527
contiguous items 14
continuation line 103, 104
continuation of lines 104
comment-entry 64

CONTINUE statement 257
control break 14, 572
    CONTROL clause 579
    GENERATE statement 622
    GROUP INDICATE clause 595
    TYPE clause 618
control break level 14
CONTROL clause 579
    TYPE clause 618
control data items 14, 572, 579
control data-name 14
control footing 14, 573
CONTROL FOOTING (CF) 618
control group 14
control heading 14, 573
CONTROL HEADING (CH) 618
control hierarchy 14
conversion 15
CONVERTING phrase, INSPECT statement 286
COPY statement 686
    compiler directing statement 98
CORR(ESPONDING) phrase 239
    ADD statement 253
    MOVE statement 295
    SIZE ERROR phrase 243
    SUBTRACT statement 347
COUNT phrase 348
counter 15
CPU-TIME 126, 246
CR, PICTURE symbol 168
CURRENCY SIGN clause 136
currency symbol 15
    PICTURE symbol 168
current record 15
current record pointer 15

D
data categories 68
    editing 172
    MOVE statement 297
    nonnumeric literals 62
    numeric literals 62
    PICTURE clause 164
    VALUE clause 202

data classes 68
data clause 15
data description entry 15, 142
  clauses 148
  elementary items 144
  formats 144
  group items 145
data description, computer-independent 64
Data Division 138
  indexed file organization 498
  inter-program communication 542
  relative file organization 446
  Report Writer 572, 575
  sequential organization 376
  sorting and merging 653
data formats 73
data item 16
data items 142
  alphabetic 68, 169
  alphanumeric 169
  numeric 69, 170
data processing statements 101
DATA RECORDS clause 383
  indexed file organization 502
  relative file organization 450
  sequential organization 383
  sorting and merging 653
data, incompatible 245
data-name 16, 49, 151
  indexed 82
  qualification 77
  subscripted 93
data-name clause 151
DATE, ACCEPT statement 248
DATE-COMPILED paragraph 119
DATE-ISO4 126, 246
DATE-TO-YYYYMMDD function 714
DAY, ACCEPT statement 248
DAY-OF-WEEK, ACCEPT statement 248
DAY-TO-YYYYDDD function 717
DB, PICTURE symbol 168
DE 618
debugging aids 353
debugging lines 16, 106, 353

DEBUGGING MODE clause 123
    debugging lines 353
DECIMAL-POINT IS COMMA clause 137
declarative sentence 16
DECLARATIVES 210, 213
    USE statement 413, 418, 474
declaratives 16, 105
de-editing 16, 299
DELETE statement 458
    FILE STATUS clause 440, 491
    indexed file organization 509
    I-O status 431, 480
    OPEN statement 402, 460, 511
    relative file organization 458
DELIMITED phrase 341
    STRING statement 341
    UNSTRING statement 348
delimited scope statement 16, 99
delimiter 16
DELIMITER phrase 348
DEPENDING ON phrase 279
    GO TO statement 279
DEPENDING phrase
    OCCURS clause 158
    RECORD clause 390, 452, 504, 653
descendant 528
descending key 17
DESCENDING KEY phrase 155
    MERGE statement 655
    OCCURS clause 155
    SEARCH statement 330
    SORT statement 663
DESCENDING phrase, OCCURS clause 158
DETAIL (DE) 618
detail report groups 573
device 362
device names 362
DIN 1
direct indexing 17, 94
direct subscripting 17, 93
directly contained program 527
DISC 652
DISPLAY phrase 192, 193

DISPLAY statement 259
figurative constants 53
maximum logical record size 260
DIVIDE statement 263
arithmetic statements 236
division 17, 105
division header 17
dollar sign ($) 168
currency symbol 125, 136
PICTURE symbol 168
DOWN BY 337
DUPLICATES phrase 489, 663
ALTERNATE RECORD KEY clause 489
DYNAMIC 438
ACCESS MODE clause 438, 488
dynamic access 17
indexed file organization 480
relative file organization 430
DYNAMIC clause 150

E

EBCDIC table 108
editing 172
sign control symbols 174
editing character 18
elementary items 18, 65, 142
categories (MOVE) 297
elementary moves 297
elements 42
ellipsis 44
ELSE phrase, IF statement 280
END DECLARATIVES 211, 213
end program header 18, 115, 538
nested source programs 113
end statement 101
ENDING phrase 411
end-of-file 397
CLOSE statement 397, 457, 508
END-OF-PAGE phrase 97
conditional statements 97
WRITE statement 420
end-of-volume 398
CLOSE statement 398
entry 18

ENTRY statement 564
CALL statement 551
Environment Division 120
debugging aids 353
indexed file organization 484
relative file organization 434
segmentation 640
sequential organization 359
sorting and merging 649
ENVIRONMENT-NAME 125
ENVIRONMENT-VALUE 125, 250, 262
EQUAL TO comparison 223
START statement 471, 520
ERROR phrase 417, 473, 522
EVALUATE statement 267
EXCEPTION phrase 97, 417, 473, 522
conditional statements 97
execution time 18
EXIT PERFORM statement 276
EXIT PROGRAM statement 566
CALL statement 551
CANCEL statement 561
EXIT statement 274
explicit scope terminator 18, 99
exponentiation 237
extend mode 18, 460
EXTEND phrase 402
OPEN statement 402, 460, 511
USE statement 417, 473, 522
extended access 19
EXTERNAL clause 544, 546
external data 534
external data item 19
external floating-point items 70
external record 19

F

FALSE phrase 267
FD entry 377
indexed file organization 499
relative file organization 447
Report Writer 575
sequential organization 377
sorting and merging 647, 649

FD level indicator
indexed file organization 499
relative file organization 447
Report Writer 575
sequential organization 377
figurative constants 19, 53
DISPLAY statement 259
INSPECT statement 287
STOP statement 340
STRING statement 341
UNSTRING statement 348
VALUE clause 204
file 19
attributes 64
file clause 19
file concepts 355
indexed file organization 479
relative file organization 429
sequential organization 355
file connector 19
file control entry (see FILE-CONTROL paragraph) 360
FILE CONTROL paragraph
relative file organization 435
sorting and merging 647, 652
file description entry 19
indexed file organization 499
relative file organization 447
Report Writer 575
sequential organization 377
file description, EXTERNAL clause 544
file lock 397
CLOSE statement 397, 457, 508
file organization 20
indexed 479
relative 429
sequential 355
FILE phrase, USE statement 411
file position indicator 20
FILE SECTION 376
indexed file organization 498
relative file organization 446
Report Writer 575
sequential organization 376

FILE STATUS clause 365
CLOSE statement 397, 457, 508
DELETE statement 458, 509
indexed file organization 491
I-O status 356, 431, 480
OPEN statement 402, 460, 511
READ statement 406, 463, 512
relative file organization 440
REWRITE statement 409, 468, 517
sequential organization 365
START statement 471, 520
WRITE statement 420, 476, 524
FILE-CONTROL paragraph 360
indexed file organization 485
sequential organization 360
file-name 20, 49
FILLER clause 151
FINAL 579
CONTROL clause 579
SUM clause 610
TYPE clause 618
FIRST DETAIL phrase 582
FIRST, INSPECT statement 286
fixed overlayable segments 636
fixed-point items 69, 170
fixed-point literals 62
floating-point items 69, 171
external 70
internal 70
floating-point literals 63
FOOTING 386
FOOTING phrase 582
FOR 286
format 2, 20
FROM phrase 246
ACCEPT statement 246
PERFORM VARYING statement 315
RELEASE statement 660
REWRITE statement 409, 468, 517
SUBTRACT statement 345
WRITE statement 420, 476, 524
function 20
function-identifier 20
function-name 20

G

general rules 2
GENERATE statement 622
CONTROL clause 579
INITIATE statement 624
REPORT SECTION 576
SUM clause 610
TERMINATE statement 625
TYPE clause 618
GIVING phrase 241
ADD statement 252
DIVIDE statement 264
MERGE statement 655
MULTIPLY statement 302
SORT statement 663
SUBTRACT statement 346
GLOBAL clause 547
global name 21
GLOBAL phrase, USE statement 417, 568
glossary 7
GO TO statement 278
ALTER statement 254
MERGE statement 655
PERFORM statement 304
SEARCH statement 327, 331
segmentation 636
SORT statement 663
GOBACK statement 567
GREATER THAN (OR EQUAL) comparison 223
START statement 471, 520
GROUP INDICATE clause 595
VALUE clause 204
group items 21, 65, 142, 145
group moves 297
groups 142

H

handling of input/output errors 213
HEADING phrase 582
high order end 21
HIGH-VALUE(S) 53

I

ID Division 116
Identification Division 116
IDENTIFICATION DIVISION, inter-program communication 539
identifier 21, 87
IF statement 280
imperative sentences 98
imperative statements 21, 98
implementor-name 22, 50, 126
    ASSIGN clause 362, 652
    RERUN clause 372, 444, 496, 649
    SPECIAL-NAMES paragraph 125
implied subjects and relational operators 233
IN 87
IN qualifier connective 77
    indexing 82
    subscripting 80
independent segments 636
index 22, 91, 95
    changing 96
    permissible value ranges 95
index data item 22
    MOVE statement 294
INDEX phrase 192, 201
INDEXED 492
    ORGANIZATION clause 492
INDEXED BY phrase 155, 158
indexed data-name 22
indexed file 22
indexed file organization 479
indexed organization 22, 479
indexing
    comparison with subscripting 96
    condition-name 48
    direct 94
    formats 82
    qualification 77
    relative 95
    subscripting 92
index-name 22, 49
    OCCURS clause 155
    PERFORM statement 315
    relation condition 223
    SEARCH statement 325

SET statement 334
indicator area 23, 102
indirectly contained program 527
INITIAL clause 539
INITIAL phrase 286
INSPECT statement 286
initial program 23
initial state 23, 532
CANCEL statement 561
EXIT PROGRAM statement 566
INITIAL clause 539
INITIALIZE statement 283
INITIATE statement 624
GENERATE statement 622
REPORT SECTION 576
SUM clause 610
TERMINATE statement 625
input file 23
input mode 23, 403, 460
INPUT phrase 402
OPEN statement 402, 460, 511
USE statement 417, 473, 522
input procedure 23
INPUT PROCEDURE phrase 663
input/output label handling 213
input/output statements 101
sequential organization 396
input-output area 369, 442, 494
input-output control entry (see I-O-CONTROL paragraph) 370
input-output error, USE statement 417, 473, 522
input-output file 23
input-output procedures 645
INPUT-OUTPUT SECTION
indexed file organization 484
relative file organization 434
sequential organization 359
sorting and merging 647
input-output statements
indexed file organization 507
relative file organization 456
INSPECT statement 286
integer 23
integer function 23
internal data 24, 534

internal data item 24
internal file 24
internal floating-point items 70
inter-program communication 527
    concepts 527
    runtime control 529
INTO 263, 406
    DIVIDE statement 263
    READ statement 406, 463, 465, 512
    RETURN statement 661
    STRING statement 341
    UNSTRING statement 348
INVALID KEY condition
    DELETE statement 458
    READ statement 465
    REWRITE statement 468
    START statement 471
    WRITE statement 476
invalid key condition 24, 455
    DELETE statement 509
    indexed file organization 506
    READ statement 514
    relative file organization 455
    REWRITE statement 517
    START statement 520
    WRITE statement 524
INVALID KEY phrase 97, 455
    conditional statements 97
I-O mode 21, 403, 462, 511
I-O phrase 402
    OPEN statement 402, 460, 511
    USE statement 417, 473, 522
I-O status 21, 356, 480
    FILE STATUS clause 365, 440, 491
    indexed file organization 480
    relative file organization 431
    sequential organization 356
I-O-CONTROL paragraph 370, 647
    indexed file organization 495
    relative file organization 443
    sequential organization 370
ISO 1
item alignment 72

items
    alignment of 72
    alphanumeric 70
    alphanumeric edited 71, 172
    fixed-point 170
    floating-point 171
    numeric edited 70, 172

J
job variable 126
    ACCEPT statement 246
    DISPLAY statement 259
    SPECIAL-NAMES paragraph 126
job variable name 126
    ACCEPT statement 246
    DISPLAY statement 259
    SPECIAL-NAMES paragraph 126
Journal of Development (JOD) 1
JUST(IFIED) clause 153
    condition-name 48
    figurative constants 53
    USAGE IS INDEX clause 201
    VALUE clause 203, 206
JV-job-variable-name 126

K
key 24
    of reference 516
key of reference 24
KEY phrase 471, 514
    MERGE statement 655
    OCCURS clause 155, 158
    READ statement 514
    SORT statement 663
    START statement 471, 520
key words 24, 51

L
LABEL PROCEDURE phrase 411
LABEL RECORDS clause 384
    indexed file organization 503
    relative file organization 451
    sequential organization 384
    sorting and merging 653

labels 384, 411
language elements 353
  debugging aids 353
  indexed file organization 479
  relative file organization 429
  Report Writer 572, 575
  segmentation 639
  sequential organization 359
  sorting and merging 647
  source text manipulation 685
  table handling 88
LAST DETAIL phrase 582
LEADING 185, 605
  INSPECT statement 286
  SIGN clause 185, 605
LEFT phrase, SYNCHRONIZED clause 189
LESS THAN (OR EQUAL) comparison 223
  START statement 471, 520
level concept 65
level indication 44
level indicator 24
level-number 24, 44, 49, 65, 143, 146
  data description entry 144
  format 146
  qualifier 77
level-numbers
  summary 143
library-name 25, 49
  COPY statement 686
library-text 25
LIMIT(S) 582
LINAGE clause 386
LINAGE-COUNTER special register 52, 388
LINE 582
LINE clause 597
line counter 628
line number 25
LINE NUMBER clause 597
line sequential organization 355
LINE-COUNTER special register 52, 628
LINES 386, 420, 582
link name 362, 437, 487
LINKAGE SECTION 542
literals 25, 62, 125

CURRENCY SIGN clause 125, 136
STOP statement 340
LOCK phrase 397, 457, 471, 508
logical operator 25, 230
logical page 386
logical record 25, 65
low order end 25
LOW-VALUE(S) 53

M
margin conventions 105
MARK-NEW-KEYWORDS 60
mass storage 26
mass storage file 26
MEMORY SIZE clause 124
merge file 26
merge processing 645
MERGE statement 655
OPEN statement 402, 460
segmentation 638
merging
examples 675
mnemonic-name 26, 49
ACCEPT statement 246
DISPLAY statement 259
SET statement 338
SPECIAL-NAMES paragraph 125
WRITE statement 420
MODULES, OBJECT COMPUTER paragraph 124
MOVE CORR(ESPONDING) statement 295
MOVE statement 294
CORRESPONDING phrase 239
USAGE IS INDEX clause 201
multi-branch structure 267
multi-dimensional table 89
multi-join structure 267
MULTIPLE FILE TAPE clause 371
MULTIPLY statement 301
arithmetic statements 236
multivolume file 398

N

native character set 26
native collating sequence 26
NATIVE phrase 125
negated combined condition 26
negated simple condition 26
NEGATIVE phrase, sign condition 229
nested program 527
  structure 113
nested source program 26
next executable sentence 27
next executable statement 27
NEXT GROUP clause 602
  PAGE FOOTING 582
  PAGE LIMIT clause 586
  REPORT HEADING 573
NEXT PAGE phrase 597
  LINE NUMBER clause 597
  NEXT GROUP clause 602
NEXT phrase 406
  READ statement 512
  relative file organization 463
  sequential organization 406
next record 27
NEXT SENTENCE phrase 280
  IF statement 280
  SEARCH statement 325, 330
NO LOCK phrase 463, 465, 471, 512, 514, 520
NO REWIND 397, 402
noncontiguous items 27
nonnumeric comparison 224
nonnumeric item 27
nonnumeric literals 27, 62
  continuation of lines 104
nonnumeric operands 224
NOT AT END phrase 97
  conditional statements 97
  READ statement 406, 463, 512
  RETURN statement 661
NOT AT END-OF-PAGE phrase, WRITE statement 420
NOT END-OF-PAGE phrase 97
  conditional statements 97
NOT INVALID KEY phrase 97
  conditional statements 97

DELETE statement 458, 509
READ statement 465, 514
REWRITE statement 468, 517
START statement 471, 520
WRITE statement 476, 524
NOT ON EXCEPTION phrase 97, 554
conditional statements 97
NOT ON OVERFLOW phrase 97
conditional statements 97
STRING statement 341
UNSTRING statement 348
NOT ON SIZE ERROR phrase 97, 243
ADD statement 251
COMPUTE statement 255
conditional statements 97
DIVIDE statement 263
MULTIPLY statement 301
SUBTRACT statement 345
NOT operator 230
notation
example 45
for COBOL 42
NUMERIC 220
numeric character 27
numeric comparison 224
numeric data categories 68
numeric data classes 68
numeric data items 27, 69, 170
internal representation 199
numeric edited items 70, 172
numeric fixed-point literals 62
numeric floating-point literals 63
numeric function 27
numeric literals 28, 62
continuation of lines 104
numeric move 299
numeric operands 224

O

object 267
object program 28
object time 28
OBJECT-COMPUTER paragraph 124
SEGMENT-LIMIT clause 640

occurrence number 91, 94, 96
OCCURS clause 155
    CORRESPONDING phrase 239
    RECORD clause 391
    REDEFINES clause 179
    SEARCH statement 325
    subscripting 92
    SYNCHRONIZED clause 190
OF 87
OF qualifier connective 77
    indexing 82
    subscripting 80
OFF STATUS 125, 338
OMITTED 384, 653
ON EXCEPTION phrase 97, 554
    conditional statements 97
ON OVERFLOW operand, CALL statement 551
ON OVERFLOW phrase 97
    conditional statements 97
    STRING statement 341
    UNSTRING statement 348
ON SIZE ERROR phrase 97, 243
    ADD statement 251
    COMPUTE statement 255
    conditional statements 97
    DIVIDE statement 263
    MULTIPLY statement 301
    SUBTRACT statement 345
ON STATUS 125, 338
one-dimensional table 89
OPEN mode 403, 511
open mode 28, 462
    OPEN statement 403, 462, 511
OPEN statement 402
    CLOSE statement 397, 457, 508
    DELETE statement 458, 509
    FILE STATUS clause 365, 440, 491
    indexed file organization 511
    INITIATE statement 624
    I-O status 356, 431, 480
    LINAGE clause 386
    READ statement 406, 463, 512
    relative file organization 460
    REPORT clause 575

REWRITE statement 409, 468, 517
sequential organisation 402
START statement 471, 520
WRITE statement 420, 476, 524
operands 28
overlapping 245
valid comparisons 228
operational sign 28, 185, 605
operator
arithmetic 8, 215
logical 230
relational 223, 233
optional file 28
OPTIONAL phrase 361, 436, 486
optional words 29, 51
options in arithmetic statements 239
OR operator 230
OR phrase 348
ORDER 663
ORGANIZATION clause 366
indexed file organization 492
relative file organization 441
sequential organization 366
outermost containing program 527
output file 29
output mode 29, 403, 460
OUTPUT phrase 402
OPEN statement 402, 460, 511
USE statement 417, 473, 522
output procedure 29
OUTPUT PROCEDURE phrase 655
MERGE statement 655
SORT statement 663
OVERFLOW phrase 97
conditional statements 97
STRING statement 341
UNSTRING statement 348
OVERFLOW phrase, CALL statement 551
overlapping operands 245

P

P, PICTURE symbol 167
PACKED-DECIMAL phrase 192, 198
padding character 29
PADDING CHARACTER clause 367
page 29
page body 29
page counter 629
page footing 29, 573
PAGE FOOTING (PF) 618
page heading 29
PAGE HEADING (PH) 618
PAGE LIMIT clause 582
PAGE phrase 420
PAGE-COUNTER special register 52, 629
paragraph 30, 105, 210, 212
paragraph header 30
paragraph-name 30, 49
  qualification 77
parameter transfer to C programs 552
parentheses 44
  conditions 231
  indices 82
  PICTURE clause 164
  subscripts 80
PERFORM statement 303
  segmentation 638
  USE statement 411, 473, 522
period (.)
  PICTURE symbol 168
permanent segments 636
PF 618
PH 618
phrase 30
physical record 30, 65
PIC(TURE) clause 164
  BLANK WHEN ZERO clause 148
  COMPUTATIONAL phrase 193
  CURRENCY SIGN clause 136
  DECIMAL-POINT IS COMMA clause 137
  LINKAGE SECTION 542
  SYNCHRONIZED clause 189
  USAGE IS INDEX clause 201
PICTURE character-string 64, 164

PICTURE symbols 166
plus (+)
PICTURE symbol 168
PLUS phrase 597
LINE NUMBER clause 597
NEXT GROUP clause 602
POINTER phrase 341
STRING statement 341
UNSTRING statement 348
POSITIVE phrase
sign condition 229
precedence of symbols, PICTURE symbols 166
primary record key, RECORD KEY clause 493
prime record key 31
printable group 31
printable item 31
PRINTER 126, 426
PRINTER01 427
PRINTER01 through PRINTER99 126, 362
PRINT-SWITCH 630
PRINT-SWITCH special register 52, 630
procedure 31, 210
procedure control statements 101
Procedure Division 210
debugging aids 353
indexed file organization 506
inter-program communication 549
relative file organization 455
Report Writer 574, 622
segmentation 642
sequential organization 396
sorting and merging 655
Procedure Division header 549
PROCEDURE phrase 417, 473, 522
procedure-name 31, 210
qualification 77
PROCEED TO 254
PROCESS-INFO 126, 246
program 527
nested 527
separately compiled 527
PROGRAM COLLATING SEQUENCE clause 124
program communication statements 101
program identification area 31, 104

program names
rules 530
valid 530
PROGRAM-ID paragraph 118
INITIAL clause 539
program-name 31, 49
CANCEL statement 561
program-name, CALL statement 551
pseudo-text 31, 106
COPY statement 686
REPLACE statement 692, 695
pseudo-text-delimiter 31
punctuation character 32

Q
qualification 77
CONTROL clause 579
indexing 82
LINE-COUNTER special register 628
LINKAGE SECTION 542
MERGE statement 655
PAGE-COUNTER special register 629
SORT statement 663
subscripting 80
qualified data-name 32
qualifier 32
QUOTE(S) 53

R
random access 32
indexed file organization 480
relative file organization 430
RANDOM, ACCESS MODE clause 438, 488
RD entry 572, 577
RD level indicator 577
READ statement 406
ALTERNATE RECORD KEY clause 489
CLOSE statement 397, 457, 508
DELETE statement 458, 509
FILE STATUS clause 365, 440, 491
indexed file organization 512
I-O status 356, 431, 480
OPEN statement 402, 460, 511
RECORD clause 390

RECORD KEY clause 493
relative file organization 463
REWRITE statement 409, 468, 517
sequential organization 406
WRITE statement 420, 476, 524
RECORD 406
DELETE statement 458, 509
READ statement 406
RETURN statement 661
SAME AREA clause 445, 497
record 32, 65
logical 65
physical 65
record area 32
RECORD clause 390
indexed file organization 504
relative file organization 452
sequential organization 390
RECORD CONTAINS clause 390
indexed file organization 504
relative file organization 452
REPORT clause 575
REPORT SECTION 576
sequential organization 390
sorting and merging 653
RECORD DELIMITER clause 368
record description
EXTERNAL clause 546
record description entry 33, 142
indexed file organization 498
relative file organization 446
sequential file organization 376
record format 394
RECORD IS VARYING IN SIZE 390
RECORD IS VARYING IN SIZE clause, sorting and merging 653
record key 33
RECORD KEY clause 493
ALTERNATE RECORD KEY clause 489
READ statement 515
REWRITE statement 517
START statement 520
WRITE statement 524
record length 390, 452, 504
calculating 393

record number 33
record sorting 643
RECORDING MODE clause 394
    RECORD clause 390, 394
    REPORT clause 575
    REPORT SECTION 576
    sequential organization 394
    sorting and merging 653
record-name 33, 49
RECORDS 372
    BLOCK CONTAINS clause 379, 449, 501
    RERUN clause 372, 444, 496
REDEFINES clause 179
    GLOBAL clause 548
REEL phrase 397
    CLOSE statement 397
    RERUN clause 372
    USE statement 411
reference format 33, 102
    rules 102
reference modification 33, 85
    example 86
reference-modifier 33
regions, page partitioning 586
relation 33
relation character 34
relation condition 34, 223
    complex condition 230
    index data item 22
    index-name 49
    nonnumeric operands 224
    numeric operands 224
    SORT statement 663
    valid comparisons of various operands 228
relational operators 34, 223, 233
RELATIVE 441
    ORGANIZATION clause 441
relative file 34
relative file organization 429
relative indexing 34, 95
relative key 35
RELATIVE KEY phrase 438
    ACCESS MODE clause 438
    READ statement 465

REWRITE statement 468
START statement 471
WRITE statement 476
relative organization 35, 429
relative record number 35, 429
relative subscripting 35, 93
RELEASE statement 660
RECORD clause 390
REMAINDER phrase 265
REMOVAL 397
RENAMES clause 183
level-number 66 145
REPEATED phrase, VALUE clause 206
REPLACE statement 692
compiler directing statement 98
REPLACING phrase 283
COPY statement 686
INITIALIZE statement 283
INSPECT statement 286
report clause 35
report description entry 35, 572, 577
clauses 578
report file 35
report footing 35, 573
REPORT FOOTING (RF) 618
report group 35
report group description entry 36, 142, 572, 589
clauses 591
report group types 573
report heading 36, 573
REPORT HEADING (RH) 618
report line 36
REPORT SECTION 576
GENERATE statement 622
Report Writer
functions 213
language elements 572, 575
statements 101
report writer 571
Report Writer logical record 36
REPORT(S) clause 575
report-name 36, 49, 575, 576
required words 50

RERUN clause 372
indexed file organization 496
relative file organization 444
sequential organization 372
sorting and merging 649
RESERVE clause 369
indexed file organization 494
relative file organization 442
sequential organization 369
reserved words 36, 50
list 57
optionally 60
RESET phrase 610
RETURN statement 661
RECORD clause 390
RETURN-CODE special register 52
REVERSED phrase 402
REWRITE statement 409
ALTERNATE RECORD KEY clause 489
FILE STATUS clause 365, 440, 491
indexed file organization 517
I-O status 356, 431, 480
OPEN statement 402, 460, 511
RECORD clause 390
RECORD KEY clause 493
relative file organization 468
sequential organization 409
RF 618
RH 618
RIGHT phrase
SYNCHRONIZED clause 189
ROUNDED phrase 242
ADD statement 251
COMPUTE statement 255
DIVIDE statement 263
MULTIPLY statement 301
SUBTRACT statement 345
RUN phrase 340
run unit 36, 528

S

S, PICTURE symbol 167
SAME AREA clause 374
  EXTERNAL clause 544
  GLOBAL clause 547
  indexed file organization 497
  relative file organization 445
  sequential organization 374
SAME RECORD AREA clause 374
  indexed file organization 497
  relative file organization 445
  sequential organization 374
  sorting and merging 650
SAME SORT AREA clause 650
SAME SORT-MERGE AREA clause 650
scope terminators 99
SD entry 653
SD level indicator 653
SEARCH ALL statement 330
SEARCH statement 325
SECTION 211, 642
section 36, 105, 210, 212
  segmentation 642
section header 36, 212
section-name 37, 49, 210
  segmentation 642
SEGMENT LIMIT clause 640
segmentation 635
  language elements 639
  MERGE statement 659
  SORT statement 668
segment-number 37, 642
segments 635
  fixed overlayable 636
  independent 636
  permanent 636
SELECT clause 361
  indexed file organization 486
  relative file organization 436
  sequential organization 361
  sorting and merging 652
selection object 267
selection subject 267
sentence 37

sentences 97, 210, 212
  compiler-directing 98
  conditional 97
  imperative 98
SEPARATE CHARACTER 185, 605
separately compiled program 527
separator 37
separators 43, 46
sequence
  of programs 114
  of report groups 621
sequence number area 37, 102
sequence of programs 37
SEQUENTIAL 364
  ACCESS MODE clause 364, 438, 488
  relative file organization 438
sequential access 38
  indexed file organization 479
  relative file organization 429
sequential file 38
sequential organization 38, 355
set
  of objects 267
  of subjects 267
SET statement 334
  conditional variable 338
  condition-name condition 219
  SEARCH statement 325
  SPECIAL-NAMES paragraph 127
  switch-status condition 222
sibling 528
SIGN clause 185
  class condition 220
  MOVE statement 294
  operational sign 185, 605
  PICTURE clause 164
  reference modification 85
  Report Writer 605
sign condition 38, 229
simple condition 38, 218, 219
single-volume file 398
SIZE ERROR phrase 97, 243
  ADD statement 251
  COMPUTE statement 255

conditional statements 97
DIVIDE statement 263
MULTIPLY statement 301
SUBTRACT statement 345
SIZE phrase 341
slack bytes 73
sort file 38
sort file declaration 653
sort key 644, 655, 664
sort procedures 645
sort processing 644
sort register 52
sort special register 670
SORT statement 663
OPEN statement 402, 460
segmentation 638
sorting tables 678
sort statements 101
SORT-CORE-SIZE special register 52, 670
SORT-EOW 682
sort-file-name 652, 653, 655, 661, 663
SORT-FILE-SIZE special register 52, 670
sorting
examples 672
record 643
tables 678
with century window 682
sorting and merging 644
language elements 647
SORT-MERGE 650
sort-merge file description entry 38
SORT-MODE-SIZE special register 52, 670
sort-record-name 660
SORT-RETURN special register 52
SORT-RETURN-SIZE special register 670
SORTWK 652
SOURCE clause 608
SOURCE information 573
source program 39
general structure 112
source text manipulation 685
SOURCE-COMPUTER paragraph 123
SOURCE-FIXED 695
space 44

SPACE(S) 53
space, with operators 215
special character 39
    in formats 44
special character word 39
special purpose words 51
special register 39, 51
    Report Writer 628
    sorting 670
SPECIAL-NAMES paragraph 125
    ACCEPT statement 246
    alphabet-name 48
    DISPLAY statement 259
    mnemonic-name 49
    switch-status condition 222
standard close file 399
standard close volume 400
STANDARD COBOL 1
standard file lock 399
standard labels 411
STANDARD phrase 384
    LABEL RECORDS clause 384, 451, 503, 653
    USE statement 411, 417, 473, 522
STANDARD-1 phrase 125
STANDARD-2 phrase 125
START statement 471
    ALTERNATE RECORD KEY clause 489
    FILE STATUS clause 440, 491
    indexed file organization 520
    I-O status 431, 480
    OPEN statement 460, 511
    READ statement 463, 512
    RECORD KEY clause 493
    relative file organization 471
statement 39
statements
    categories 97
    compiler-directing 98
    conditional 97
    delimited scope 99
    imperative 98
    summary of categories 100
STOP statement 340
    figurative constants 53

STRING statement 341
figurative constants 53
stroke (virgule, slash) (/)
comment line 12, 105
PICTURE symbol 168
subject 267
subprogram 39
SUB-SCHEMA SECTION 139
subscript 39
subscripted data-name 40
subscripting 92
comparison with indexing 96
condition-name 48
direct 93
formats 80
relative 93
using an arithmetic expression 93
SUBTRACT CORRESPONDING statement 347
SUBTRACT statement 345
arithmetic statements 236
CORRESPONDING phrase 239
SUM clause 610
sum counter 40, 610
SUM information 573
SUPPRESS
COPY statement 686
switch-status condition 40, 222
symbolic character 40, 134
SYMBOLIC CHARACTERS clause 134
SYNC(HRONIZED) clause 189
USAGE IS INDEX clause 201
VALUE clause 206
syntax rules 2
SYSDTA 247
SYSIPT 126, 362
SYSLST01 through SYSLST99 362
SYSnnn 372
SYSOPT 126, 362
SYSOUT 259
system files 362
system name 40
system-names 50, 126

T

table 40
multi-dimensional 89
one-dimensional 89
table definition 89
table element 40
initial value 90
reference to 96
references to 91
table handling 88
table handling statements 101
table sorting 678
an example 681
TALLY special register 52
TALLYING phrase 286
INSPECT statement 286
UNSTRING statement 348
TERMINAL 126
TERMINAL-INFO 126, 246
TERMINATE statement 625
INITIATE statement 624
REPORT SECTION 576
SUM clause 610
TYPE clause 618
text-name 40, 49
COPY statement 686
text-word 41
THEN 280
THROUGH 125
EVALUATE statement 267
MERGE statement 655
PERFORM statement 303, 307, 311, 315
RENAMES clause 183
SORT statement 663
SPECIAL-NAMES paragraph 125
VALUE clause 204
THRU (see THROUGH) 125
TIME, ACCEPT statement 248
TIMES 307
OCCURS clause 155, 158
PERFORM statement 307
VALUE clause 206
TO TEST OF phrase 276
TOP 386

TRAILING 605
SIGN clause 185
TRUE phrase 267
truth value 41
TSW-0 through TSW-31 126
two-dimensional table 89
TYPE clause 618

U
unary arithmetic operator 215
unary operator 41
UNIT phrase 372
CLOSE statement 397
RERUN clause 372
USE statement 411
unit record file 398
UNIT-RECORD volume 398
UNSTRING statement 348
figurative constants 53
UNTIL phrase 311, 315
UP BY 337
update mode 403, 460
UPON phrase 259
DISPLAY statement 259
SUM clause 610
USAGE clause 192
class condition 220
INSPECT statement 286
LINKAGE SECTION 542
RECORD clause 390
relation condition 223
Report Writer 607
SIGN clause 185, 605
STRING statement 341
UNSTRING statement 348
USAGE IS DISPLAY
reference modification 85
USAGE IS INDEX clause
SEARCH statement 325
USE AFTER STANDARD 411, 417, 473, 522
USE BEFORE REPORTING statement 626
REPORT SECTION 576
USE BEFORE STANDARD 411
USE GLOBAL statement, inter-program communication 568

USE statement 411
compiler directing statement 98
DECLARATIVES 213
DELETE statement 458, 509
indexed file organization 522
input-output error 417
inter-program communication 568
labels 411
READ statement 406, 512
relative file organization 473
REWRITE statement 468, 517
sequential organization 411
START statement 471, 520
WRITE statement 420, 476, 524
user-defined words 41, 48
USING phrase 211
ENTRY statement 564
MERGE statement 655
Procedure Division header 211, 549
SORT statement 663
USING phrase, CALL statement 551
USW-0 through USW-31 126

V
V, PICTURE symbol 167
validity of names 535
VALUE clause 202
LINKAGE SECTION 542
VALUE information 573
VALUE OF clause 395, 454
variable 41
VARYING phrase 315, 390
PERFORM statement 315
RECORD clause 390, 452, 504
SEARCH statement 325
verb 41
volume 398
CLOSE statement 398

W
WHEN 267, 325, 330
WHEN OTHER phrase 267
WITH DEBUGGING MODE clause 123
debugging lines 353

WITH DUPLICATES phrase 489
ALTERNATE RECORD KEY clause 489
WITH LOCK phrase 397, 457, 508
WITH NO ADVANCING phrase 259
WITH NO LOCK phrase 463, 465, 471, 512, 514, 520
WITH NO REWIND phrase 397, 402
WITH POINTER phrase 341
STRING statement 341
UNSTRING statement 348
WITH TEST BEFORE/AFTER phrase 311, 315
word 41, 48
words 43, 48
continuation of lines 104
WORDS phrase
OBJECT COMPUTER paragraph 124
WORKING-STORAGE SECTION 141
WRITE statement 420
ALTERNATE RECORD KEY clause 489
FILE STATUS clause 365, 440, 491
indexed file organization 524
I-O status 356, 431, 480
OPEN statement 402, 460, 511
RECORD clause 390
RECORD KEY clause 493
relative file organization 476
sequential organization 420

X

X, PICTURE symbol 167
X/OPEN Portability Guide 249, 261

Y

YEAR-TO-YYYY function 759

Z

Z, PICTURE symbol 167
ZERO / ZEROS / ZEROES 53
ZERO phrase, sign condition 229

Contents

1	**Preface**	**1**
1.1	Brief product description	1
1.2	Target group and summary of contents	2
1.3	Changes since the last version of the manual	4
1.4	Acknowledgment	5
2	**Introduction to the COBOL language**	**7**
2.1	Glossary	7
2.2	COBOL notation	42
2.3	Language concepts	46
2.3.1	COBOL character set	46
2.3.2	Separators	46
2.3.3	COBOL words	48
2.3.4	Literals	62
2.3.5	PICTURE character-string	64
2.3.6	Comment-entry	64
2.3.7	Concept of computer-independent data description	64
2.3.8	Implementor-dependent representation and alignment of data	73
2.4	Uniqueness of references	77
2.4.1	Qualification	77
2.4.2	Subscripting	80
2.4.3	Indexing	82
2.4.4	Function-identifier	84
2.4.5	Reference modification	85
2.4.6	Identifier	87
2.4.7	Condition-name	87
2.5	Table handling	88
2.5.1	Table definition	89
2.5.2	Subscripting	92
2.5.3	Indexing	94
2.5.4	Indexing and subscripting compared	96
2.6	Statements and sentences	97
2.6.1	Conditional statements and conditional sentences	97
2.6.2	Compiler-directing statements and compiler-directing sentences	98
2.6.3	Imperative statements and imperative sentences	98
2.6.4	Delimited scope statements	99

2.6.5 Scope of statements (scope terminators) . . . 99
2.6.6 Categories of statements . . . 100
2.7 Reference format . . . 102
2.7.1 General description . . . 102
2.7.2 Rules for using the reference format . . . 103
2.8 Processing a COBOL program . . . 107
2.9 EBCDIC character set . . . 108

3 Basic elements of a COBOL source program . . . 111
3.1 General description . . . 111
3.2 Structure of a COBOL program . . . 112
3.3 Structure of a nested source program . . . 113
3.4 Sequence of programs . . . 114
3.5 End program header . . . 115
3.6 Identification Division . . . 116
3.6.1 General description . . . 116
3.6.2 Structure . . . 116
3.6.3 Paragraphs . . . 118
PROGRAM-ID paragraph . . . 118
DATE-COMPILED paragraph . . . 119
3.7 Environment Division . . . 120
3.7.1 General description . . . 120
3.7.2 CONFIGURATION SECTION . . . 122
SOURCE-COMPUTER paragraph . . . 123
OBJECT-COMPUTER paragraph . . . 124
SPECIAL-NAMES paragraph . . . 125
Implementor-name . . . 126
ARGUMENT-NUMBER / ARGUMENT-VALUE / ENVIRONMENT-NAME / ENVIRONMENT-VALUE . . . 127
ALPHABET clause . . . 128
SYMBOLIC CHARACTERS clause . . . 134
CLASS clause . . . 135
CURRENCY SIGN clause . . . 136
DECIMAL-POINT IS COMMA clause . . . 137
3.8 Data Division . . . 138
3.8.1 General description . . . 138
3.8.2 WORKING-STORAGE SECTION . . . 141
Data description entry . . . 142
Data description entry formats . . . 144
Level number . . . 146
3.8.3 Clauses for data description . . . 148
BLANK WHEN ZERO clause . . . 148
DYNAMIC clause . . . 150
Data-name or FILLER clause . . . 151

JUSTIFIED clause . . . 153
OCCURS clause . . . 155
PICTURE clause . . . 164
Summary of characters and symbols in a PICTURE character-string . . . 167
Alphabetic data items . . . 169
Alphanumeric data items . . . 169
Numeric data items . . . 170
Alphanumeric edited data items . . . 172
Numeric edited data items . . . 172
Simple insertion editing . . . 173
Special insertion editing . . . 173
Fixed insertion editing . . . 174
Floating insertion editing . . . 175
Zero suppression and replacement editing . . . 177
REDEFINES clause . . . 179
RENAMES clause . . . 183
SIGN clause . . . 185
SYNCHRONIZED clause . . . 189
USAGE clause . . . 192
DISPLAY phrase . . . 193
BINARY phrase or COMPUTATIONAL phrase or COMPUTATIONAL-5 phrase . . . 195
COMPUTATIONAL-1 phrase . . . 196
COMPUTATIONAL-2 phrase . . . 197
COMPUTATIONAL-3 phrase or PACKED-DECIMAL phrase . . . 198
INDEX phrase . . . 201
VALUE clause . . . 202
3.9 Procedure Division . . . 210
3.9.1 General description . . . 210
Structure . . . 211
3.9.2 DECLARATIVES . . . 213
3.9.3 Arithmetic expressions . . . 215
3.9.4 Conditions . . . 218
Condition-name condition . . . 219
Class condition . . . 220
Switch-status condition . . . 222
Relation condition . . . 223
Sign condition . . . 229
Complex conditions . . . 230
Implied subjects and relational operators . . . 233
3.9.5 Arithmetic statements . . . 236
3.9.6 Options in arithmetic statements . . . 239
CORRESPONDING phrase . . . 239
GIVING phrase . . . 241

	ROUNDED phrase	242
	ON SIZE ERROR phrase	243
3.9.7	Overlapping operands	245
3.9.8	Incompatible data	245
3.9.9	Statements	246
	ACCEPT statement	246
	ADD statement	251
	ALTER statement	254
	COMPUTE statement	255
	CONTINUE statement	257
	DISPLAY statement	259
	DIVIDE statement	263
	EVALUATE statement	267
	EXIT statement	274
	EXIT PERFORM statement	276
	GO TO statement	278
	IF statement	280
	INITIALIZE statement	283
	INSPECT statement	286
	MOVE statement	294
	MULTIPLY statement	301
	PERFORM statement	303
	SEARCH statement	325
	SET statement	334
	STOP statement	340
	STRING statement	341
	SUBTRACT statement	345
	UNSTRING statement	348
3.10	Debugging	353
4	**Sequential file organization**	**355**
4.1	File concepts	355
4.1.1	Sequential organization	355
4.1.2	Line sequential organization	355
4.1.3	I-O status	356
4.2	Language elements of the Environment Division	359
	INPUT-OUTPUT SECTION	359
	FILE-CONTROL paragraph	360
	SELECT clause	361
	ASSIGN clause	362
	ACCESS MODE clause	364
	FILE STATUS clause	365
	ORGANIZATION clause	366
	PADDING CHARACTER clause	367

	RECORD DELIMITER clause	368
	RESERVE clause	369
	I-O-CONTROL paragraph	370
	MULTIPLE FILE TAPE clause	371
	RERUN clause	372
	SAME AREA clause	374
4.3	Language elements of the Data Division	376
	FILE SECTION	376
	File description (FD) entry	377
	BLOCK CONTAINS clause	379
	CODE-SET clause	382
	DATA RECORDS clause	383
	LABEL RECORDS clause	384
	LINAGE clause	386
	RECORD clause	390
	RECORDING MODE clause	394
	VALUE OF clause	395
4.4	Language elements of the Procedure Division	396
4.4.1	Input/output statements	396
	CLOSE statement	397
	OPEN statement	402
	READ statement	406
	REWRITE statement	409
	USE statement	411
	WRITE statement	420
5	**Relative file organization**	**429**
5.1	File concepts	429
5.1.1	Relative organization	429
5.1.2	Sequential access to records	429
5.1.3	Random access to records	430
5.1.4	Dynamic access to records	430
5.1.5	I-O status	431
5.2	Language elements of the Environment Division	434
	INPUT-OUTPUT SECTION	434
	FILE-CONTROL paragraph	435
	SELECT clause	436
	ASSIGN clause	437
	ACCESS MODE clause	438
	FILE STATUS clause	440
	ORGANIZATION clause	441
	RESERVE clause	442
	I-O-CONTROL paragraph	443
	RERUN clause	444

	SAME AREA clause	445
5.3	Language elements of the data division	446
	FILE SECTION	446
	File description (FD) entry	447
	BLOCKS CONTAINS clause	449
	DATA RECORDS clause	450
	LABEL RECORDS clause	451
	RECORD clause	452
	VALUE OF clause	454
5.4	Language elements of the Procedure Division	455
5.4.1	Invalid key condition	455
5.4.2	Input/output statements	456
	CLOSE statement	457
	DELETE statement	458
	OPEN statement	460
	READ statement	463
	REWRITE statement	468
	START statement	471
	USE statement	473
	WRITE statement	476
6	**Indexed file organization**	**479**
6.1	File concepts	479
6.1.1	Indexed organization	479
6.1.2	Sequential access to records	479
6.1.3	Random access to records	480
6.1.4	Dynamic access to records	480
6.1.5	I-O status	480
6.2	Language elements of the Environment Division	484
	INPUT-OUTPUT SECTION	484
	FILE-CONTROL paragraph	485
	SELECT clause	486
	ASSIGN clause	487
	ACCESS MODE clause	488
	ALTERNATE RECORD KEY clause	489
	FILE STATUS clause	491
	ORGANIZATION clause	492
	RECORD KEY clause	493
	RESERVE clause	494
	I-O-CONTROL paragraph	495
	RERUN clause	496
	SAME AREA clause	497
6.3	Language elements of the Data Division	498
	FILE SECTION	498

	File description (FD) entry	499
	BLOCKS CONTAINS clause	501
	DATA RECORDS clause	502
	LABEL RECORDS clause	503
	RECORD clause	504
6.4	Language elements of the Procedure Division	506
6.4.1	Invalid key condition	506
6.4.2	Input-output statements	507
	CLOSE statement	508
	DELETE statement	509
	OPEN statement	511
	READ statement	512
	REWRITE statement	517
	START statement	520
	USE statement	522
	WRITE statement	524
7	**Inter-program communication**	**527**
7.1	Concepts	527
7.2	Control of inter-program communication	529
7.2.1	Runtime control	529
7.2.2	Rules for program names	530
7.2.3	Initial state	532
7.3	Using common data	534
7.3.1	External and internal data	534
7.3.2	Local and global names	535
7.4	Language elements for inter-program communication	537
7.4.1	Overview	537
7.4.2	End program header	538
7.4.3	Language elements of the Identification Division	539
	PROGRAM-ID paragraph	539
7.4.4	Language elements of the Data Division	542
	LINKAGE SECTION	542
	EXTERNAL clause in file and data description entries	544
	GLOBAL clause	547
7.4.5	Language elements of the Procedure Division	549
	Procedure Division header	549
	CALL statement	551
	CANCEL statement	561
	ENTRY statement	564
	EXIT PROGRAM statement	566
	GOBACK statement	567
	USE statement with GLOBAL phrase	568

8	**Report Writer**	**571**
8.1	General description	571
8.1.1	General description of the Data Division	572
8.1.2	General description of the Procedure Division	574
8.2	Language elements of the Data Division	575
	REPORT clause	575
	REPORT SECTION	576
	Report description entry	577
	CODE clause	578
	CONTROL clause	579
	PAGE LIMIT clause	582
	Report group description entry	589
	COLUMN clause	593
	GROUP INDICATE clause	595
	LINE clause	597
	NEXT GROUP clause	602
	SIGN clause	605
	USAGE clause	607
	SOURCE clause	608
	SUM clause	610
	TYPE clause	618
8.3	Language elements of the Procedure Division	622
	GENERATE statement	622
	INITIATE statement	624
	TERMINATE statement	625
	USE BEFORE REPORTING statement	626
8.4	Special registers of the Report Writer	628
	LINE-COUNTER special register	628
	PAGE-COUNTER special register	629
	PRINT-SWITCH special register	630
	CBL-CTR special register	631
	Function 1 of the CBL-CTR special register	631
	Function 2 of the CBL-CTR special register	634
9	**Segmentation**	**635**
9.1	General description	635
9.2	Organization	635
9.3	Fixed portion of the program	636
9.4	Independent segments	636
9.5	General rules for segmentation	637
9.6	Language elements	639
9.6.1	Language elements of the Environment Division	640
	SEGMENT-LIMIT clause	640

9.6.2 Language elements of the Procedure Division . . . 642
Segment number . . . 642

10 Sorting of records . . . 643
10.1 Sorting and merging of files . . . 644
10.1.1 Sort processing . . . 644
10.1.2 Merge processing . . . 645
10.1.3 Sort and merge without input/output procedures . . . 645
10.1.4 Sort with input/output procedures . . . 646
10.1.5 Overview of language elements . . . 647
10.1.6 Language elements of the Environment Division . . . 649
RERUN clause . . . 649
SAME AREA clause . . . 650
SELECT clause . . . 652
10.1.7 Language elements of the Data Division . . . 653
Sort-file description . . . 653
10.1.8 Language elements of the Procedure Division . . . 655
MERGE statement . . . 655
RELEASE statement . . . 660
RETURN statement . . . 661
SORT statement . . . 663
10.1.9 Special registers for file SORT . . . 670
10.1.10 Examples of file SORT . . . 672
10.2 Sorting of tables . . . 678
SORT statement . . . 678
10.3 Sorting two-digit year numbers with a century window . . . 682

11 Source text manipulation . . . 685
COPY statement . . . 686
REPLACE statement . . . 692
SOURCE FIXED compiler directive . . . 695

12 Intrinsic functions . . . 697
12.1 General . . . 697
12.2 Overview of intrinsic functions . . . 700
ACOS - Arcosine . . . 703
ADDR - Address of an identifier . . . 704
ANNUITY - Annuity . . . 705
ASIN - Arcsine . . . 708
ATAN - Arctangent . . . 709
CHAR - Character in the collating sequence . . . 710
COS - Cosine . . . 711
CURRENT-DATE - Current date . . . 712
DATE-OF-INTEGER - Date conversion . . . 713

DATE-TO-YYYYMMDD - year conversion 714
DAY-OF-INTEGER - Date conversion 716
DAY-TO-YYYYDDD year conversion 717
FACTORIAL - Factorial 719
INTEGER - Next smaller integer 720
INTEGER-OF-DATE - Date conversion 721
INTEGER-OF-DAY - Date conversion 722
INTEGER-PART - Integer part of a floating-point value 723
LENGTH - Number of characters 724
LOG - Logarithm 725
LOG10 - Logarithm of base 10 726
LOWER-CASE - Lowercase letters 727
MAX - Value of maximum argument 728
MEAN - Arithmetic mean of arguments 730
MEDIAN - Median of arguments 731
MIDRANGE - Mean of minimum and maximum arguments 732
MIN - Value of minimum argument 733
MOD - Modulo 734
NUMVAL - Numeric value of string 735
NUMVAL-C - Numeric value of string with optional currency sign 737
ORD - Ordinal position in collating sequence 739
ORD-MAX - Ordinal position of maximum argument 740
ORD-MIN - Ordinal position of minimum argument 741
PRESENT-VALUE - Present value (period-end amount) 742
RANDOM - Random number 744
RANGE - Difference value 747
REM - Remainder 748
REVERSE - Reverse order of string characters 749
SIN - Sine 750
SQRT - Square root 751
STANDARD-DEVIATION - Standard deviation of arguments 752
SUM - Sum of arguments 754
TAN - Tangent 755
UPPER-CASE - Uppercase letters 756
VARIANCE - Variance of arguments 757
WHEN-COMPILED - Date and time of compilation 758
YEAR-TO-YYYY year conversion 759

Related publications 763

Index 769

COBOL85 V2.3A (BS2000/OSD)

COBOL Compiler
Reference Manual

Target group
COBOL users in BS2000

Contents
- COBOL glossary
- Introduction to Standard COBOL
- Description of the full language set of the COBOL85 compiler: formats, rules and examples illustrating the COBOL ANS85 language elements of the "High" language subset, and the Siemens Nixdorf-specific extensions.

Edition: May 1998

File: cb85_bs.pdf

BS2000 is a registered trademark of Siemens Nixdorf Informationssyteme AG.

Copyright © Siemens Nixdorf Informationssysteme AG, 1998.

All rights, including rights of translation, reproduction by printing, copying or similar methods, even of parts, are reserved.

Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Delivery subject to availability; right of technical modifications reserved.

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Copyright Fujitsu Technology Solutions, 2009

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.

Copyright Fujitsu Technology Solutions, 2009