COBOL85
Reference Manual

FUJITSU

Third Edition: November 1996

Preface

Purpose

This COBOL85 Reference Manual covers the rules for writing programs in COBOL (COmmon Business Oriented Language) using FUJITSU COBOL85.

Audience

This manual is for users who develop COBOL programs using COBOL85. This manual assumes users possess basic programming knowledge.

Organization

The table below shows how this manual is organized.

Chapter /Appendix	Description
Chapter 1. General Rules	Covers the general rules including the language elements of COBOL, unique reference, writing of literals, and reference format
Chapter 2. COBOL Modules	Lists and explains the facilities of COBOL
Chapter 3. Identification Division and End Program Header	Explains the syntax of the identification division and end program header
Chapter 4 . Environment Division	Explains the syntax of the environment division
Chapter 5. Data Division	Explains the syntax of the data division
Chapter 6. Procedure Division	Explains the syntax of the procedure division
Chapter 7. Source Text Manipulation	Explains the syntax of statements used in the source text manipulation function
Chapter 8. Database(SQL)	Explains the syntax of the database function(SQL)
Chapter 9. Communication Database	Explains the syntax of the communication database function
Chapter 10. Micro Focus Native Functions	Explains the syntax of the Micro Focus native functions
Appendix A. List of Reserved Words	Lists the reserved words in COBOL
Appendix B. System Quantitative Restrictions	Lists the quantity restrictions of the system
Appendix C. Code Tables	Lists the character sets and their large-to-small sequence
Appendix D. Intermediate Results	Explains the attributes and accuracy of intermediate results
Appendix E. Functional Differences	Lists the functional differences for each system
Appendix F. Control Record	Explains the formats of the control records

Scope of FUJITSU COBOL85 Functions

The FUJITSU COBOL85 functions consist of JIS standard COBOL functions (JIS X 8002-1992) and FUJITSU COBOL85 extended functions. The FUJITSU COBOL85 functions are as follows:

- FUJITSU COBOL85 functions
  - JIS standard COBOL functions
    - Required functions
      - Nucleus (2, nucleus 1, 2)
      - Sequential I-O function (2, sequential 1, 2)
      - Relative I-O function (2, relative 0, 2)
      - Indexed I-O function (2, indexed 0, 2)
      - Inter-program communication function (2, communication 1, 2)
      - Sort-merge function (1, sort 0, 1)
      - Source test manipulation function (2, source test 0, 2)
    - Optional functions
      - Report writer function (1, report 0, 1)
      - Built-in function function (1, function 0, 1)
      - Debug function (2, debug 0, 2)*
      - Segmentation function (2, segmentation 0, 2)*
  - FUJITSU COBOL85 extended functions
    - Presentation file function
    - Screen handling module
    - Command line argument and environment variable operation function
    - Database function
    - Communication database function
    - Micro Focus native function

*: The debug function and segmentation function are treated as comments when the program is run.

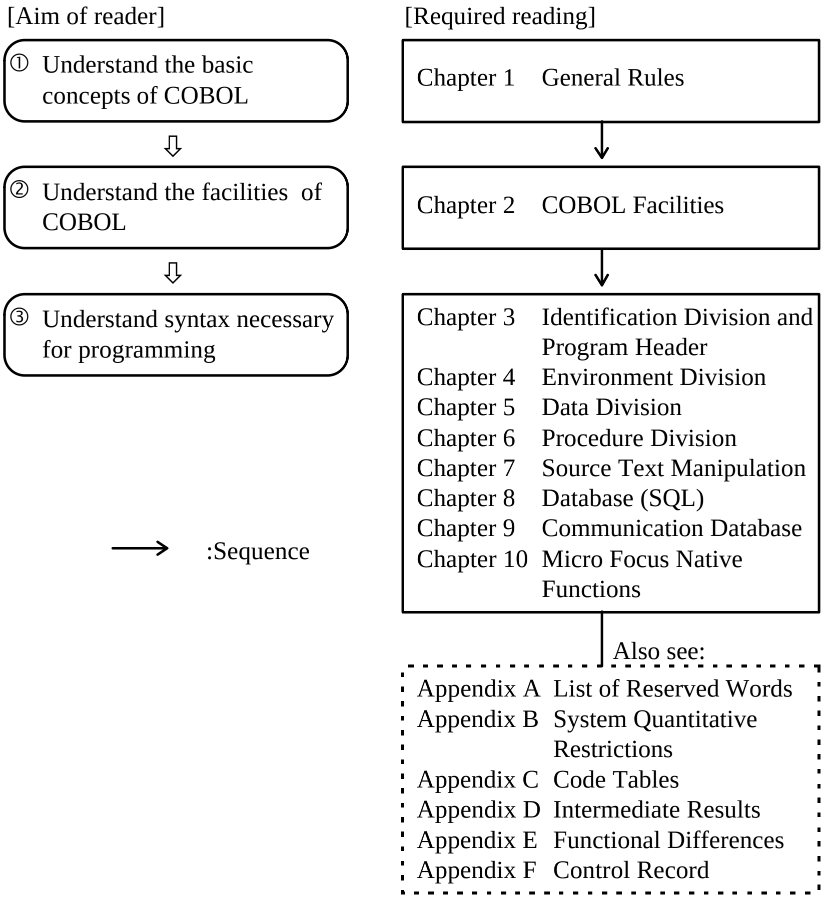
Note: In the above, standard mnemonic symbols for each JIS standard COBOL function is shown in parentheses. From left to right the symbols show the position of the function in the hierarchy, the function abbreviation, and the maximum and minimum levels of the function to which the level belongs.

In this reference manual, FUJITSU COBOL85 is called simply COBOL.

How to Use this Manual

Users who wish to gain an understanding of the basic concepts of COBOL and its facilities should first read Chapters 1 and 2 in that order. Users who already have an understanding of COBOL may skip the first two chapters and read any of the other chapters or appendixes as required.

The following diagram shows the sequence in which the chapters and appendixes of this reference manual should be read.



[Aim of reader]

① Understand the basic concepts of COBOL

② Understand the facilities of COBOL

③ Understand syntax necessary for programming

→ :Sequence

[Required reading]

Chapter 1 General Rules

Chapter 2 COBOL Facilities

Chapter 3 Identification Division and Program Header
Chapter 4 Environment Division
Chapter 5 Data Division
Chapter 6 Procedure Division
Chapter 7 Source Text Manipulation
Chapter 8 Database (SQL)
Chapter 9 Communication Database
Chapter 10 Micro Focus Native Functions

Also see:

Appendix A List of Reserved Words
Appendix B System Quantitative Restrictions
Appendix C Code Tables
Appendix D Intermediate Results
Appendix E Functional Differences
Appendix F Control Record

Conventions Used in This Manual

Symbols Used in [Format]

In each chapter, [Format] shows how to write COBOL language elements, such as statements and clauses. The words shown in [Format] must be written in the sequence shown unless otherwise specified in the syntax rules or general rules.

The following table lists the meaning of symbols appearing in [Format].

Example of Convention	**Description**
<u>IF</u>	Underlined text indicates that the character string is a keyword and cannot be omitted. Keywords not underlines can be omitted.
VALUE IS constant-1	Uppercase characters indicate a COBOL reserved word. These character strings must be written exactly as they appear in [Format].
{ identifier-1 }	Brackets indicate that one of the values given can be selected.
[data-name-1]	Square parenthesis indicate that one of the values can be selected or the value can be omitted.
{| <u>COMMON</u> |}	Choice indicators mean that at least one of the values given can be written. Each character string must be unique.
…	Ellipses indicate the item immediately preceding can be specified repeatedly.

Example of Convention	**Description**
WORKING-STORAGE SECTION.	Periods must be written in the same position.
=, -, >, <, =, >=, <=, ->	Special characters are key words. They are not underlined, but must not be omitted.
CHECK WITH PASCAL LINKAGE ALL PARAGRAPH-ID COBOL	Commands, statements, clauses, and options you enter or select appear in uppercase. Program section names, and some proper names also appear in uppercase.
"COBOL85 User's Guide" See "Data Division" in Chapter 5.	References to other publications or sections within publications are in quotation marks.

Shading

FUJITSU COBOL85 extended functions are highlighted in the text by shading. A shaded section title indicates that the entire title is a FUJITSU COBOL85 extended function.

Syntax Rules and General Rules

The explanation of COBOL language elements such as statements and clauses in each chapter is arranged into [Format], syntax rules, and general rules. [Format] shows the arrangement of elements making up a statement or clause. Syntax rules explain the arrangement of elements in [Format] and restrictions on their arrangement.

General rules explain the results of execution and compilation where a statement or clause was written. They also explain the meaning of elements in [Format] and the relationship between the elements.

Syntax rules or general rules are omitted when there are no rules relating to the elements in [Format].

Obsolete Elements

Elements marked "Obsolete elements" in the text are given in the 1985 issue of the ANSI COBOL standard but may not be included in the next edition. Fujitsu recommends that obsolete elements not be used when creating new programs.

System-specific Functions

Some parts of the COBOL85 common syntax described in this manual depend on system functions, and differ among systems.

Such parts are indicated by the following system names:

Indication	Corresponding system
DS	UXP / DS COBOL85 V20L1X
HP	HP-UX COBOL85 V20L10
Sun	Solaris COBOL85 V20L20
Win	COBOL85 V30L10 for Windows® 95, COBOL85 V30L10 for Windows NT® & COBOL85 V20L10 for Windows® 3.1
Win32	COBOL85 V30L10 for Windows® 95 & COBOL85 V30L10 for Windows NT®
Win16	COBOL85 V20L10 for Windows® 3.1

The followings are positions of the indications and the area corresponded the indications:

Position of indication	**Correspond area of the indication**
top of a heading	the chapter, the section ,etc.
head of a paragraph	the paragraph
in table	the row

See Appendix E for a list of functional differences.

Acknowledgment

COBOL language specifications are based on the original specifications developed by the Conference on Data Systems Languages (CODASYL), and specifications in this manual are also derived from these specifications. The chapters listed below were included on request from CODASYL.

COBOL is not owned by any particular company, organization, or group; it has been designed for general use in industry. CODASYL does not guarantee and bears no responsibility in relation to the programming method, the accuracy of language, and functions.

The following copyright owners permitted partial use of the following documents when the original specifications were put together. This permission also extends to the use of the original specifications in other COBOL specifications.

- FLOW-MATIC (trademark of Sperryland Inc.), "Programming for the Univac (R) I and II, Data Automation Systems", copyright Sperryland Inc. 1958, 1959
- IBM Commercial Translator, library number F28-8013, copyright IBM Inc. 1959
- FACT, library number 27A5260-2760, copyright Minneapolis Honeywell Inc. 1960

Trademarks

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Motif is a trademark of Open Software Foundation, Inc.

MS-DOS is a registered trademark of Microsoft Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

Windows is a registered trademark of Microsoft Corporation.

Windows NT is a registered trademark of Microsoft Corporation

HP and HP-UX are trademarks of Hewlett-Packard Company

Micro Focus is a registered trademark of Micro Focus Ltd., U.K.

Sun and Solaris are trademarks of Sun Microsystems, Inc.

Contents

Chapter 1. General Rules

This chapter explains the concepts of the COBOL language and covers the general rules.

Characters and Character Sets

Though there are several types of character sets that can be used on a computer. COBOL programs use specific character sets for writing. These character sets are called “COBOL character sets.” There are four types of COBOL character sets:

- Alphabetic character set
- Numeric character set
- Special character set
- National character set

Alphabetic Character Set

The alphabetic characters are listed below. Each alphabetic character uses 1 byte of storage area.

- Uppercase letters (26 letters from A to Z)
- Lowercase letters (26 letters from a to z)
- Space

Except in a nonnumeric literal, lowercase letters are equivalent to their corresponding uppercase letters.

Numeric Character Set

There are ten numeric characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Each numeric character uses a 1 byte storage area.

Special Character Set

The table below lists the special characters. Each special character uses a 1 byte storage area.

Character	Meaning
+	Plus sign
-	Minus sign or hyphen
*	Asterisk
/	Slash
=	Equal sign
$	Dollar sign
,	Comma
;	Semicolon
.	Period or decimal point
“	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater-than sign
<	Less-than sign
:	Colon
&	Ampersand

National Characters

National characters are contained in the Japanese character set. Each character in the character set is expressed by a two-byte internal code.

Unless otherwise specified, alphabetic or numeric characters do not contain national characters.

Computer Character Set

Character sets used on a computer are called “computer character sets.” The COBOL character sets make up a part of the computer character set.

Characters from COBOL character sets and from other computer character sets can be used in the following items:

- Nonnumeric literal
- National nonnumeric literal
- Comment entry
- Comment line
- In-line comment

Basic Overview of Language

A COBOL program is made up of separators, COBOL words, literals, character-strings in PICTURE clauses, comment entries, comment lines, and in-line comments. The COBOL character sets are used for writing all program elements except for the literal values, comment entries, comment lines, and in-line comments. These elements are written using characters from a computer character set.

A COBOL program can also contain a line consisting of all spaces (blank line).

Separator

A separator is a character delimiting a character-string. Use any of the following characters as a separator:

- A sequence of one or more spaces. This is called a “separator space.”
- A sequence of one comma followed by one or more spaces. This is called a “separator comma.”
- A sequence of one semicolon followed by one or more spaces. This is called a “separator semicolon.”
- A sequence of one period followed by one or more spaces. This is called a “separator period.”
- A left parenthesis
- A right parenthesis

- A quotation mark indicating the start of a literal value (“, X”, NC”, N”, NX” or B”)
- A quotation mark indicating the end of the literal value
- The == pseudo-text delimiter
- A colon
- The -> pointer qualification symbol
- The & concatenation operator

Separator Space

The separator space delimits a COBOL word, literal, or character string in a PICTURE clause.

Separator Comma and Separator Semicolon

The separator comma and separator semicolon facilitate program reading. Either one can be used wherever a separator space can be used.

A separator space can also be included immediately before or after these separators.

Separator Period

A separator period indicates the end of an identification division for a division, section, or paragraph or the end of a division, section, paragraph, entry, or sentence. A period can only be written where indicated by “.” in [Format].

A separator space can be included immediately before or after a period.

Left and Right Parentheses

Use left and right parentheses to enclose a subscript, reference modifier, arithmetic expression, Boolean expression, conditional expression, or argument list. Parentheses are always used in pairs.

A separator space can be included immediately before or after the left or right parenthesis.

Quotation Mark

Quotation marks are used to enclose a nonnumeric literal, hexadecimal nonnumeric literal, National nonnumeric literal, National hexadecimal nonnumeric literal, or a character string indicating the Boolean literal value.

Quotation marks are always used in pairs, one mark indicating the start of a literal value and the other mark indicating the end of the literal value. Rules for describing quotation marks are the following:

1. Enclose a nonnumeric literal with a pair of quotation marks.
2. Enclose a hexadecimal nonnumeric literal with X” and a quotation mark.
3. Enclose a National nonnumeric literal with NC” and a quotation mark or N” and a quotation mark.
4. Enclose a National hexadecimal nonnumeric literal with NX” and a quotation mark.
5. Enclose a Boolean literal with B” and a quotation mark.

In all cases a separator space or a left parenthesis must immediately precede the quotation mark indicating the start of a literal value. Also, a separator space, separator comma, separator semicolon, separator period, or right parenthesis must immediately follow the quotation mark indicating the end of the literal value.

Pseudo-text Delimiter

Pseudo-text delimiters enclose pseudo-text. They are used in pairs. A separator space must immediately precede the pseudo-text delimiter indicating the start of pseudo-text. A separator space, separator comma, separator semicolon, or separator period must immediately follow the pseudo-text delimiter indicating the end of pseudo-text.

Colon

A colon is used to write a reference modifier. A separator space can be included immediately before or after the colon.

Pointer Qualification Symbol

A pointer qualification symbol is used to write a pointer qualifier. A separator space can be included immediately before or after the colon.

Concatenation Operator

A concatenation operator is used to link a literal. A separator space must be written immediately before and after the concatenation operator.

Character-strings Not Regarded as Separators

The rules for separators do not apply to character-strings written in the locations listed below. A character-string written in any of the following locations is not regarded as a separator even if the character-string has the same format as a separator:

- Numeric literal
- A portion enclosed by a separator indicating the start of a literal value and a separator indicating the end of a literal value in a nonnumeric literal, hexadecimal nonnumeric literal, National nonnumeric literal, or Boolean literal
- A character string in a PICTURE clause
- A comment entry
- A comment line
- An in-line comment

COBOL Word

A COBOL word is a single character string consisting of COBOL character set characters. There are four types of COBOL words:

- User-defined words
- System-name
- Reserved word
- Function-name

Each COBOL word has its own restrictions in addition to the following general restrictions:

1. A COBOL word must not exceed 30 characters in length.
2. A COBOL word must be made up of alphabetic characters (A to Z and a to z), numeric characters (0 to 9), or hyphens (-). It must not contain any spaces. Lowercase letters are regarded as being equivalent to their corresponding uppercase letters. A user-defined word may consist of national characters.
3. A hyphen must not be used as the first or last character of a COBOL word.

User-defined Words

A user-defined word is any of the following 19 words named by the user:

- Positioning unit name
- Print mode name
- Symbolic-constant
- Symbolic-character
- Text-name
- Index-name
- Class-name
- Condition-name
- Section-name
- Paragraph-name
- Data-name
- Library-name
- File-name
- Alphabet-name
- Program-name
- Report-name
- Mnemonic-name
- Record-name
- Level-number

The names of all user-defined words except the level-number must be unique. However, the condition-name, data-name, record-name, and index-name may be the same provided the qualifiers allow reference uniqueness. The same level-number can be written multiple times in one program.

Rules for Describing User-defined Words

A user-defined word is made up of alphabetic characters, numeric characters and hyphens, or national characters. User defined words except level-numbers, library-names, program-names, and text-names can be written in national characters. A user-defined word made up of national characters is called a “National user-defined words.”

The following rules must be satisfied when creating a user-defined word from alphabetic characters or numeric characters and hyphens:

1. The word must not exceed 30 characters in length.
2. The word must be made up of alphabetic characters (A to Z and a to z), numeric characters (0 to 9), or hyphens. It must not contain any spaces. Lowercase letters are regarded as equivalent to their corresponding uppercase letters.
3. A hyphen must not be used as the first or last character of the word.
4. A character string making up a user-defined word must not be identical to the reserved word, but can be the same as the function-name or system-name.
5. All user-defined words except a paragraph-name, section-name, and level-number must include at least one alphabetic character.

The following rules must be satisfied when creating a user-defined word from national characters:

1. The word must not exceed 30 characters in length.
2. The word must be made up of national characters and contain no spaces. Lowercase letters in JIS non-kanji characters are not regarded as being equivalent to their corresponding uppercase letters in JIS non-kanji characters.
3. A JIS non-kanji hyphen or JIS non-kanji minus sign must not be used as the first or last character of the word.
4. The word must include at least one JIS level-1 kanji character, JIS level-2 kanji character, JIS non-kanji hiragana character, JIS non-kanji katakana character, or JIS non-kanji macron.
5. Level-numbers, library-names, program-names, and text-names cannot be written in national characters.

Application of User-defined Words

The application for user-defined words and the locations where they can be written are explained below.

POSITIONING UNIT NAME

A positioning unit name is the name given to the value unit indicating the column to be printed. It is defined in the environment division special names paragraph. It can be written in the “positioning unit name-n” indicated in [Format].

PRINT MODE NAME

A print mode name is the name given to the characters presentation format to be printed, such as character size, pitch, typeface, rotation, and configuration. It is defined in the environment division special names paragraph. It can be written in the “print mode name-n” indicated in [Format].

SYMBOLIC-CONSTANT

A symbolic-constant is the name given to a literal for referencing the literal by name. It is defined in the environment division special names paragraph. It can be written in the “literal-n” or “integer-n” indicated in [Format].

SYMBOLIC-CHARACTER

A symbolic-character is the name given to a character for expressing a figurative constant using specific characters. It is defined in the environment division special names paragraph. It can be written in the “literal-n” indicated in [Format].

TEXT-NAME

A text name is the name given to library text for referencing in the COBOL library. It can be written in the “text-name-n” indicated in [Format].

INDEX-NAME

An index-name is the name given to the table index. It is defined in the data division description entry . It can be written in the “index-name-n” indicated in [Format].

CLASS-NAME

A class name is the name given to a character set specified by the user for verifying the class of the data item contents. It is defined in the environment division special names paragraph. It can be written in the “class-name-n” indicated in [Format].

CONDITION-NAME

There are two types of condition names:

1. The name given to an assumed value by a data item
2. The name given to the external ON or OFF status switch

The first type of condition-name is defined in the condition-name data division data description entry . A data item associated to a condition-name is called a “conditional variable.” This type of condition-name can be used in a condition-name condition or in a SET statement. A condition-name condition is an abbreviated format of a relation condition. It indicates whether the value of the conditional variable is equal to the condition-name value. This condition-name can be used in a SET statement for setting the conditional variable value .

The second type of condition-name is defined in the environment division special names paragraph . It can represent the switch-status condition. The switch-status condition indicates whether the status of the external switch is specified correctly in the special names paragraph. Use the SET statement to set the status of the external switch. Specify the mnemonic-name corresponding to the condition-name in the SET statement.

The condition-name can be written in the “condition-name-n” indicated in [Format].

SECTION-NAME

A section-name is the name given to a procedure division section. It can be written in “section-name-n” or “procedure-name-n” indicated in [Format].

PARAGRAPH-NAME

A paragraph-name is the name given to a procedure division paragraph. It can be written in the “paragraph-name-n” or “procedure-name-n” indicated in [Format].

DATA-NAME

A data-name is the name given to a data item. It is defined in the data description entry . It can be written where “data-name-n” or “identifier-n” is indicated in [Format]. When “data-name-n” is indicated in [Format], do not use reference modification, subscripting, qualification, or an explicit pointer except where specifically permitted. When “identifier-n” is indicated in [Format], reference modification, subscripting, qualification, or an explicit pointer may be required for use referencing a particular data-name.

LIBRARY-NAME

A library-name is the name given to a COBOL library for referencing the COBOL library.

FILE-NAME

A file-name is the name given to a file connector. It is defined in the environment division file control entry. It can be written in the “file-name-n” indicated in [Format].

ALPHABET-NAME

An alphabet-name is either the name given to a particular character pair or indicates the large-to-small sequence of characters. It is defined in the environment division special names paragraph . It can be written in the “alphabet-name-n” is indicated in [Format].

PROGRAM-NAME

A program-name is the name given to a COBOL source program. It is defined in the identification division program-name paragraph. It can be written in the “program-name-n” indicated in [Format].

REPORT-NAME

A report-name is the name given to a report. It is defined in the report description entry of the data division. It can be written where “report-name-n” is indicated in [Format].

MNEMONIC-NAME

A mnemonic-name is a name which sets up an association with a module name (a name which identifies a module) to enable using a module defined outside the program. It is defined in the environment division special names paragraph . It can be written in the “mnemonic-name-n” indicated in [Format].

RECORD-NAME

A record-name is the name given to a record. It is defined in the data division record description entry . It can be written in the “record-name-n” indicated in [Format].

LEVEL-NUMBER

A level-number is the number given to indicate the data item, report group item, or screen item hierarchy. A level-number between 01 and 49 indicates the item location in the hierarchical structure of a record. Use level-number 66, 77, or 88 when writing a data description entry having special characteristics. The high-order zero in level-numbers 01 to 09 may be omitted. The level-number can be written in “level-number” indicated in [Format].

System-name

A system-name links the program with the operating system. A system-name can be either a user-specified word or a specific word.

The following system-names are used:

- Function-name
- Computer-name
- Language-name
- File identifier name

FUNCTION-NAME

A function-name identifies a module defined outside the program such as a logical device on the system, printed pages layout, and the external switch. When using a module defined externally, the function-name must be associated with the mnemonic-name in the environment division special names paragraph.

The function-name is different for each module. See the section titled “Special Names Paragraph,” for the rules pertaining to description of a function-name.

COMPUTER-NAME

A computer-name identifies the computer compiling or running the program.

The user specifies a computer-name by writing a character string conforming to the COBOL words rules.

LANGUAGE-NAME

A language-name identifies a particular programming language.

FILE IDENTIFIER NAME

A file identifier name is a name for identifying a file on a storage medium. The file identifier name must be associated with the file-name in the environment division file control entry before the file can be used.

The user specifies the file identifier name. See the sections beginning with the title “ASSIGN clause (sequential file, relative file, indexed file),” and ending with the title “ASSIGN clause (presentation file),” for rules on describing a file identifier name.

Reserved Word

A reserved word is a specific word for process writing in a program. There are several types of reserved words as shown below.

- Reserved word
  - Required word
    - Key word
    - Special-character word
  - Optional word
  - Special use word
    - Special register
    - Word identifying a figurative constant

A character-string identical to a reserved word cannot be used as a user-defined word or system-name. See Appendix A, “List of Reserved Words,” for the reserved words list.

REQUIRED WORD

A required word is a word that must be included. There are two types of required words:

- Key word
- Special-character word

A key word is written in uppercase letters and underlined in [Format].

A special-character word is one of the twelve characters shown below. It is not underlined in [Format] in order to avoid confusion with other symbols.

+	-	*	/	**	>	<	=	>=	<=	&	->

OPTIONAL WORD

An optional word may be written or omitted as required. It appears in uppercase letters in [Format] and is not underlined.

SPECIAL USE WORD

There are two types of special use words:

- Special register
- Figurative constant

SPECIAL REGISTER

A special register is a storage area where the compiler automatically generates code. It saves information generated when using a specific module. It can also be used to report information to the program (system) running a specific module.

There are several types of special registers:

- A special register for use with the sequential file function

  LINAGE-COUNTER

- A special register for use with an inter-program communication module

  PROGRAM-STATUS

  RETURN-CODE (synonym for PROGRAM-STATUS)

- A special register for use with the sort-merge module

  SORT-STATUS

- A special register for use with a presentation file module

  EDIT-MODE

  EDIT-OPTION

  EDIT-COLOR

  EDIT-STATUS

  EDIT-CURSOR

- A special register for use with a report writer module

  LINE-COUNTER

  PAGE-COUNTER

A special register acts as a data item having a specific category. The special register can be written in the "data-name-n" or "identifier-n" indicated in [Format] and the category held by the special register is permitted.

WORD IDENTIFYING A FIGURATIVE CONSTANT

A figurative constant is expressed using reserved words. For example, ALL and SPACE in the figurative constant ALL SPACE are reserved words.

Function-name

A function-name is a word indicating the name of a function. It can be written in a function-identifier. The function type determines the location where the function-identifier can be written. See the section titled "General Rules for Functions," for more information on the location where the function-identifier can be written.

Literal

A literal is data having a value written into a program. There are several literal types as shown below.

- Literal
  - Numeric literal
    - Fixed-point numeric literal
    - Floating-point literal
  - Nonnumeric literal
  - Hexadecimal nonnumeric literal
  - National nonnumeric literal
    - National nonnumeric literal
    - National hexadecimal nonnumeric literal
  - Boolean literal

Numeric Literal

A numeric literal is a literal whose numerals represent a value. There are two types of numeric literal:

- Fixed-point numeric literal
- Floating-point literal

Fixed-point Numeric Literal

A fixed-point numeric literal is expressed by a combination of signs, numeric characters, and a decimal point.

[Format]

$$\begin{bmatrix} + \\ \\ - \end{bmatrix} [\underbrace{\text{numeric-character-string-1}}_{\text{Integer part}}][.\underbrace{\text{numeric-character-string-2}}_{\text{Decimal part}}]$$

1. A fixed-point numeric literal consists of a sign, integer part, decimal-point, and decimal part.
2. The sign “+” represents the plus sign, “-” represents the minus sign, and “.” represents the decimal-point.
3. Numeric character string-1 and numeric character string-2 must contain numeric characters between 0 and 9 only.
4. The total number of digits in the integer part and decimal part must be between 1 and 18.
5. The fixed-point numeric literal value is algebraic.
6. The fixed-point numeric literal category is numeric character.
7. A fixed-point numeric literal can be written in the following locations:

- A fixed-point numeric literal having a decimal-point can be written where “literal-n” is indicated in [Format], and a numeric literal is permitted.
- A fixed-point numeric literal not containing a decimal-point can be written in “literal-n” indicated in [Format] and in a permitted numeric literal , or it can be written in “integer-n” indicated in [Format]. Only integers of 1 or more without any sign may be written where “integer-n” is indicated in [Format], unless otherwise permitted in the syntax rules.

Floating-point Literal

A floating-point literal is expressed in the format "mantissa*(10**exponent)."

[Format]

$$\underbrace{\begin{bmatrix} + \\ - \end{bmatrix} \underbrace{\text{[numeric-character-string-1]}}_{\text{Integer part of mantissa part}} \; \underbrace{\text{[.numeric-character-string-2]}}_{\text{Decimal part of mantissa part}}}_{\text{Mantissa part}} \text{ E } \underbrace{\begin{bmatrix} + \\ - \end{bmatrix} \underbrace{\text{[numeric-character-string-3]}}_{\text{Exponent}}}_{\text{Exponent part}}$$

1. A floating-point literal consists of a mantissa part, E, and an exponent part. The mantissa part consists of a sign, an integer part, a decimal-point and a decimal part. The exponent part consists of a sign and an exponent.

2. The sign "+" represents the plus sign, "-" represents the minus sign, and "." represents the decimal-point.

3. Numeric character string-1 to numeric character string-3 must contain numeric characters between 0 and 9 only.

4. The total number of digits in the integer part and decimal parts of the mantissa part must be between 1 and 15.

5. The number of digits in the exponent must be 1 or 2.

6. The value of the floating-point literal must be an algebraic value represented by the following formula:

   Mantissa part * (10 ** exponent part)

7. A single-precision floating-point literal can be 0 or an absolute value number greater than approximately 1.18 x 10 to the -38th power and is less than about 3.4 x 10 to the +38th power. A double-precision floating-point literal can be 0 or an absolute value number greater than approximately 2.23 x 10 to the -308th power and is less than about 1.79 x 10 to the +308th power.

8. The floating-point literal category is numeric character.

9. A floating-point literal can be written where “literal-n” is indicated in [Format], and a numeric literal is permitted.

10. The internal format of floating-point items the following:

    a. The sign of the mantissa is indicated by the leftmost bit.

    b. For a single-precision floating-point number, the exponent occupies the eight bits following the leftmost bit. For a double-precision floating-point number, the exponent occupies the 11 bits following the leftmost bit.

    c. For a single-precision floating-point number, the mantissa occupies the 23 bits following the exponent bits. For a double-precision floating-point number, the mantissa occupies the 52 bits following the exponent bits.

Nonnumeric Literal

A nonnumeric literal is a literal whose value is represented by a computer character set character.

[Format]

“{ character-1 } ...”

1. The string in character-1 must be delimited at both ends with separator quotation marks.
2. Any character belonging to a computer character set may be written as character-1.
3. Use double quotation marks to delimit character-1 at each end by writing a single quotation mark twice.
4. The character-1 count must be between 1 and 160. The character length -1 must be between 1 and 160 bytes.
5. The value of a nonnumeric literal is the character written in character-1. The separator quotation marks enclosing the nonnumeric literal are not part of the value of the nonnumeric literal.
6. The category of a nonnumeric literal is alphanumeric character.
7. A nonnumeric literal can be written where “literal-n” is indicated in [Format], and a nonnumeric literal is permitted.

Hexadecimal Nonnumeric Literal

A hexadecimal nonnumeric literal is a literal whose value is represented by a computer character set character. A hexadecimal nonnumeric literal represents the literal value as a hexadecimal character.

[Format]

```
X”{ hexadecimal-character-1 } ...”
```

1. The list in hexadecimal-character-1 must be delimited by the separator X” at the left and by double separator quotation marks at the right.
2. Hexadecimal-character-1 must consist of characters between 0 and 9 or A and F. Two consecutive characters in hexadecimal-character-1 represent a single character code or part of a character code from a computer character set.
3. The number of characters in hexadecimal-character-1 must be between 2 and 320.
4. The value of a hexadecimal nonnumeric literal is the value (a character belonging to a computer character set) indicated in the string of hexadecimal-character-1. A hexadecimal nonnumeric literal is regarded as the equivalent to a nonnumeric literal.
5. The hexadecimal nonnumeric literal category is alphanumeric character.

A hexadecimal nonnumeric literal can be written in “literal-n” indicated in [Format], and a nonnumeric literal is permitted. In this reference manual the term “nonnumeric literal” also covers a hexadecimal nonnumeric literal.

6. The table below shows the internal bit configuration for a hexadecimal character.

Hexadecimal Character	**Internal Bit Configuration**
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

National Nonnumeric Literal

A national nonnumeric literal is a literal whose value is represented by a national character. There are two types of national nonnumeric literal:

- National nonnumeric literal
- National hexadecimal nonnumeric literal

National Nonnumeric Literal

A National nonnumeric literal represents the literal value as a computer character set national character.

[Format 1]

NC”{ national-character-1 } ...”

[Format 2]

N”{ national-character-1 } ...”

1. Format 1 and format 2 are equivalent.
2. The list in national-character-1 must be delimited by the separator NC” or the separator N” at the left and by double separator quotation marks at the right.
3. Any national character belonging to a computer character set can be written to national-character-1.
4. The national-character-1 count must be between 1 and 80.
5. The National nonnumeric literal value is the national character written in the national-character-1 list.
6. The National nonnumeric literal category is national language.

7. A National nonnumeric literal can be written in “literal-n” indicated in [Format] and a National nonnumeric literal is permitted.

National Hexadecimal Nonnumeric Literal

A national hexadecimal nonnumeric literal represents the literal value as a hexadecimal character.

[Format]

NX”{ hexadecimal-character-1 } ...”

1. The list in hexadecimal-character-1 must be delimited by the separator NX” at the left and by double separator quotation marks at the right.
2. Hexadecimal-character-1 must consist of characters between 0 and 9 or A and F. Two consecutive characters in hexadecimal-character-1 represent a single national character code from a computer character set.
3. The hexadecimal-character-1 count must be between 4 and 320.
4. The national hexadecimal nonnumeric literal value is the value (a national character belonging to a computer character set) indicated in the hexadecimal-character-1 list. A national hexadecimal nonnumeric literal is regarded as equivalent to a National nonnumeric literal.
5. The category of a national hexadecimal nonnumeric literal is national language.
6. A national hexadecimal nonnumeric literal can be written in “literal-n” indicated in [Format] and a national nonnumeric literal is permitted.

Boolean Literal

A Boolean literal is a literal value represented by a Boolean character.

[Format]

B”{ Boolean-character-1 } ...”

1. The list in Boolean-character-1 must be delimited by the separator B” at the left and by double separator quotation marks at the right.
2. The Boolean-character-1 must consist of characters “0” and “1” only.
3. The Boolean-character-1 count must be between 1 and 160.
4. The Boolean literal value is the value written in the Boolean-character-1 list.
5. The Boolean literal category is Boolean.
6. A Boolean literal can be written in “literal-n” indicated in [Format] and a Boolean literal is permitted.

Figurative Constant

A figurative constant repeats a literal having a specific value or a literal. There are seven types of figurative constants:

- ZERO
- SPACE
- HIGH-VALUE
- LOW-VALUE
- QUOTE
- ALL literal

- Symbolic-character

The following table shows the format and value for each figurative constant.

Name	**Format**	**Value**
ZERO	[ALL] ZERO [ALL] ZEROS [ALL] ZEROES	Has one of the following values depending on context: - Numeric value zero - Repeats character “0” from a computer character set. - Repeats Boolean character “0”.
SPACE	[ALL] SPACE [ALL] SPACES	Has one of the following values depending on context: - Repeats an alphabetic character space from a computer character set. - Repeats a national character space from a computer character set.
HIGH-VALUE	[ALL] HIGH-VALUE [ALL] HIGH-VALUES	Repeats the largest character code in the large-to-small character sequence in a program.
LOW-VALUE	[ALL] LOW-VALUE [ALL] LOW-VALUES	Repeats the smallest character code in the large-to-small character sequence in a program.
QUOTE	[ALL] QUOTE [ALL] QUOTES	Repeats the quotation mark.
ALL literal	ALL literal	Repeats the literal written after ALL.
Symbolic-character	[ALL] Symbolic-character	Repeats a symbolic-character.

1. ALL must be immediately followed by a separator space.
2. Singular and plural forms such as SPACE and SPACES are synonymous.

3. Quotation marks enclosing the value in a nonnumeric literal, hexadecimal nonnumeric literal, national nonnumeric literal or Boolean literal cannot be expressed by the figurative constant QUOTE. For example, “ABD” cannot be written as QUOTE ABD QUOTE.

4. A literal written in an ALL literal must be a nonnumeric literal, hexadecimal nonnumeric literal, national nonnumeric literal, Boolean literal, or an associated symbolic literal. A figurative constant can not be written in an ALL literal.

5. A symbolic-character is defined in the SPECIAL-NAMES paragraph SYMBOLIC CHARACTERS clause.

6. A figurative constant can be written in “literal-n” indicated in [Format], unless particular restrictions apply.

7. Any transcription or relation between an ALL literal in which ALL is immediately followed by a literal of two or more characters and a numeric data item or numeric edited data item becomes an obsolete element. Avoid using this element when generating a new program.

8. When a figurative constant containing a character-string covering several positions is written in a VALUE clause, or when a transcription or relation is positioned between such a figurative constant and a data item, the figurative constant is regarded as being the same length as the data item. That is, the character-string specified in the figurative constant is repeated until it is as long as the data item. If it becomes longer than the data item, the character-string is truncated from the right so it is equal in length to the data item. The figurative constant value is determined before the JUSTIFIED clause in the data item is processed.

Concatenation Expression

A concatenation expression is a method of expressing the value of a single literal by combining two or more literals with a concatenation operator.

[Format]

```
[ALL] { literal-1 } { & literal-2 } ...
```

1. The ampersand (&) is called a concatenation operator. The concatenation operator must be immediately preceded and followed by a separator space.
2. The combination in literal-1 and literal-2 must be one of the following two types:
   a. A combination consisting of at least one nonnumeric literal, hexadecimal nonnumeric literal, figurative constant SPACE, figurative constant HIGH-VALUE, figurative constant LOW-VALUE, figurative constant QUOTE, figurative constant ZERO, or symbolic-character. The literal category expressed in this concatenation expression is alphanumeric character.

   b. A combination of national nonnumeric literal or a combination of a national nonnumeric literal and a figurative constant SPACE. The literal category expressed in this concatenation expression is national language.
3. ALL must be affixed to the figurative constant when writing a figurative constant in literal-1 or literal-2.

4. Either of the following values can be used as the figurative constant value when writing a figurative constant in literal-1 or literal-2:

   a. When writing at least one nonnumeric literal or hexadecimal nonnumeric literal in the literal contained in a concatenation expression, or when writing only a figurative constant in the literal contained in a concatenation expression, the figurative constant value can be expressed as a single alphanumeric character specified in the figurative constant.

   b. When writing at least one national nonnumeric literal in a literal contained in a concatenation expression, the figurative constant value can be expressed as a single national character specified in the figurative constant.

5. The concatenation expression value is the value obtained when the literal values in the concatenation expression are linked in order from the left.

6. Write ALL when using a literal expressed in a concatenation expression as the figurative constant. A concatenation expression containing ALL is treated in the same manner as an ALL literal.

7. A concatenation expression can be written where “literal-n” is indicated in [Format] and a literal whose category is concatenation expression is permitted.

8. A comment entry or blank line can be written between the concatenation operator in a concatenation expression and a literal.

Literal for Special Applications

There are several types of literals for special applications:

- Program-name literal
- File-identifier literal
- Text-name literal

Program-name Literal

A program-name literal is a literal whose value is represented by a program-name.

A program-name literal must be written in accordance with the rules for nonnumeric literals. The program-name literal value must conform to the rules for program-names determined by the system. For more information on the rules for program-name literals, refer to the "COBOL85 User's Guide".

A program-name literal can be written in “program-name-literal-n” indicated in [Format].

File-identifier Literal

A File-identifier literal is a literal whose value is represented by the File-identifier of a sequential file, relative file, or indexed file.

The format of the File-identifier literal is shown below.

“name”

Specify the file name on the storage medium in “name.”

A file-identifier literal must be written in accordance with the rules for nonnumeric literals. The value and length of a file-identifier literal must conform to the rules determined by the system. For more information on the rules for file-identifier literals, refer to the “COBOL85 User's Guide.”

A file-identifier literal can be written where “file-identifier literal-n” is indicated in [Format].

Text-name Literal

A text-name literal is a literal whose value is represented by a text-name.

A text-name literal must be written in accordance with the rules for text-name literals. The value and length of the text-name literal must conform to the rules determined by the system. For more information on the rules for text-name literals, refer to the ”COBOL85 User's Guide.”

A text-name literal can be written where “text-name-literal-n” is indicated in [Format].

Picture Character-string

A PICTURE character-string is a character-string written in a PICTURE clause. A PICTURE character-string consists of alphabetic characters, numeric characters, and special characters belonging to a COBOL character set. The section titled “PICTURE clause,” lists the rules for describing a PICTURE character-string.

Comment Entry

A comment entry is an identification division entry. It consists of any characters belonging to a computer character set.

A comment entry is an obsolete element. Avoid using this element when generating a new program.

Concept of Data Description

This section explains the data description concept.

Concept of Levels

The most comprehensive collection of data consisting of at least one piece of data is called a “record.”

A record is configured according to the concept of levels. This concept arose from the need to subdivide a record for referencing data. The record can be subdivided twice for referencing data.

The most basic portion of a record, that is, a portion which cannot be split further, is called an “elementary item.” Several elementary items can be grouped so they can be referenced at one time. Furthermore, several groups can be grouped. An elementary item can therefore be a subordinate to several groups. A group to an elementary item subordinate is called a “group item.”

Concept of Class

Elementary items other than index data items, pointer data items, and internal floating-point data items all belong to a category and class. The data class for a group item is regarded as alphanumeric character when the program is activated, regardless of the elementary items category belonging to it.

The relationship between category and data class is shown in the following table.

Level of Item	Type	Category	Class
Elementary item	Alphabetic data item	Alphabetic character	Alphabetic character
	Numeric data item	Numeric	Numeric
	Numeric edited data item	Numeric edited	Alphanumeric item character
	Alphanumeric data item	Alphanumeric character	
	Alphanumeric edited data item	Alphanumeric edited	
	National data item	National	National
	National edited data item	National edit	
	Boolean item	Boolean	Boolean
	External floating-point item	External floating item	Alphanumeric character
Group item	—	Alphabet character Numeric Numeric edited Alphanumeric character Alphanumeric edited National National edit Boolean External floating item	Alphanumeric character

Standard Alignment Rule

The standard alignment rule when saving data in an elementary item is determined by the receiving area category. Data from the sending area is aligned in accordance with the following rules.

1. When the receiving area is a numeric data item, the data is aligned in accordance with the following rules:

   a. When the assumed decimal-point in the receiving area is explicit, the decimal-point in the sending area is aligned with the decimal-point position in the receiving area before the data is saved. At this time, the end is padded with zeroes or truncated, whichever is applicable.

   b. When the assumed decimal-point in the receiving area is not explicit, the assumed decimal-point is regarded as being on the right end of the receiving area and data is aligned in according with the rule described in (a) above.

2. When the receiving side is a numeric edited data item, the sending area aligns data with the decimal-point position in the receiving area and edits and transcribes it.

   At this time, the end is padded with zeroes or truncated, whichever is applicable. However, no zeroes are added in positions where the leading zero string in the digit place is replaced with spaces.

3. When the receiving area is an alphanumeric data item, alphanumeric edited data item, alphabetic data item, National data item, or national edited data item, the sending area is aligned with the left side of the receiving area and transcribed. At this time, the right end is either padded with spaces or truncated, whichever is applicable. However, if a JUSTIFIED clause was specified in the receiving area, data is aligned in accordance with the rules for JUSTIFIED clauses.

4. When the receiving area is a Boolean item, the Boolean position at the sending area low order end is aligned with the Boolean position at the receiving area low order end before data is saved. At this time, the end is padded with Boolean character 0 or truncated, whichever is applicable.

Adjustment of Data Boundary

When a SYNCHRONIZED clause has been written in a data description entry, an unused character position or Boolean position may be inserted by the compiler for allocating data items to a native boundary. The character position and Boolean position inserted by the compiler to align data with a native boundary are called the “slack byte” and “slack bit”, respectively.

Slack Byte

When a SYNCHRONIZED clause is specified in a binary item or internal floating-point data item, the data item is allocated to a native boundary in accordance with the SYNCHRONIZED clauses rules. At this time, a slack byte may be inserted into the record.

Rules for Insertion of a Slack Byte

The compiler allocates a relative address to each data item in the record. The leading address in the record is 0 and the value obtained by adding the data item size to the relative address of the data item becomes the relative address of the next data item.

When a SYNCHRONIZED clause has been specified in a binary item or internal floating-point data item, the data item digit positions must be padded. When a record contains a data item where the digit positions must be padded and the relative address of the data item is not a multiple of the byte count in the data item native boundary, a slack byte is inserted. A slack byte is an implicit FILLER of the minimum length required for padding.

A slack byte can be inserted in the following positions:

1. When the data item immediately preceding the data item requiring padding is an elementary item, a FILLER having the same level-number as the level-number of the data item requiring padding is inserted immediately before the data item requiring padding.
2. When the data item immediately preceding the data item requiring padding is a group item, the program searches for the item immediately following an elementary item having the highest level-number among the series of group items subordinate to the data item requiring padding. It inserts a FILLER item having the same level-number as the level-number of the group item immediately before the group item.

3. When the group item specifying an OCCURS clause has a subordinate data item requiring padding, no slack byte is inserted if the size of one occurrence of the group data is not a multiple of the maximum value of the byte count for the data item native boundary subordinate to the group item. A slack byte is inserted after each occurrence of the group item.

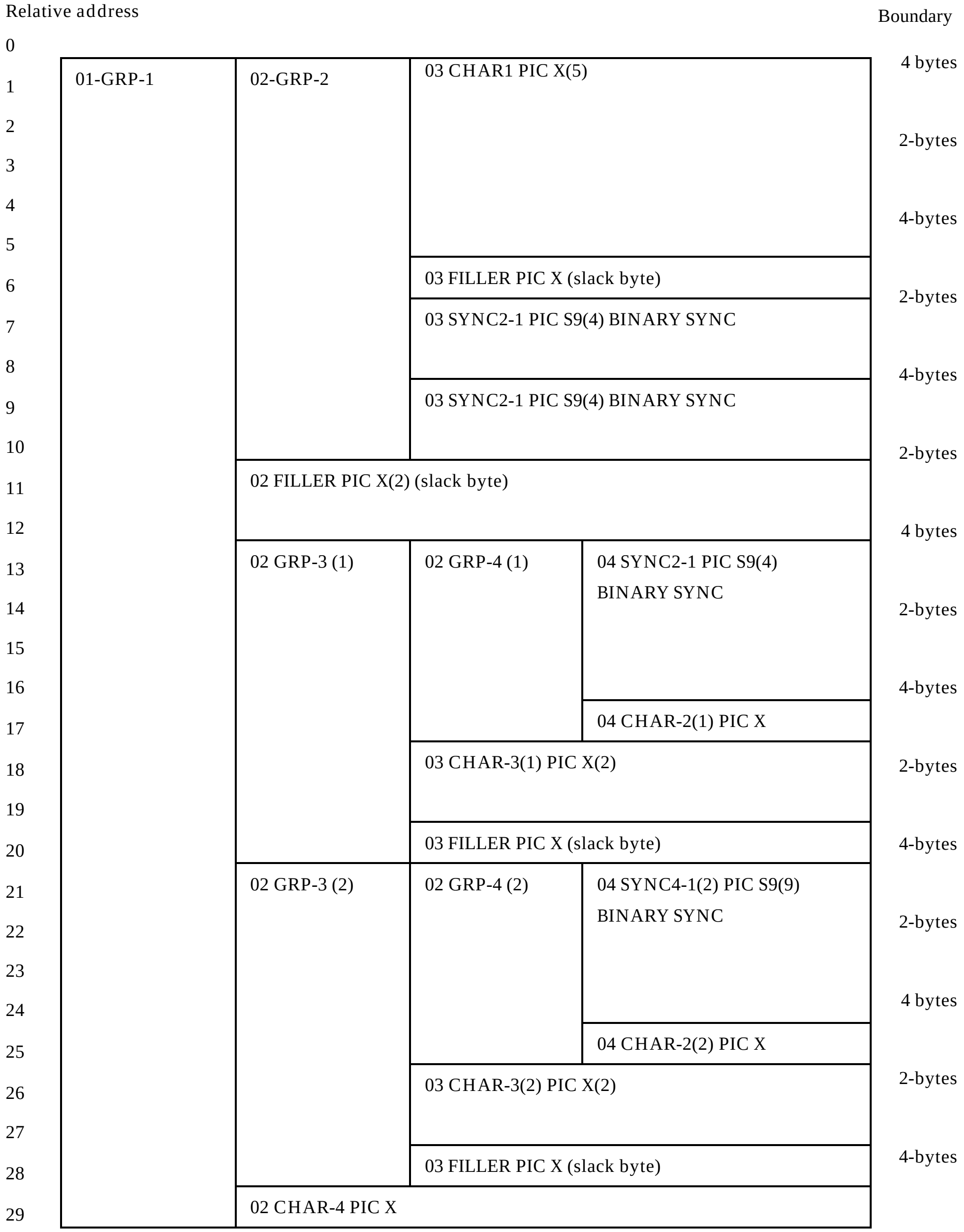
When a slack byte is inserted, the size of the group item is expanded by the size of the implicit FILLER inserted in the group item.

Example of Insertion of a Slack Byte

Below is an example of slack byte insertion for the following record description entry.

```
01  GRP-1.
   02  GRP-2.
      03  CHAR-1   PIC X(5).
      03  SYNC2-1  PIC S9(4)  BINARY SYNC.
      03  SYNC2-2  PIC S9(4)  BINARY SYNC.
   02  GRP-3    OCCURS 2.
      03  GRP-4.
         04  SYNC4-1  PIC S9(9)  BINARY SYNC.
         04  CHAR-2   PIC X.
      03  CHAR-3     PIC X(2).
   02  CHAR-4      PIC X.
```

The diagram below shows area allocation of the above record description entry.



Relative address

Boundary

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

4 bytes
2-bytes
4-bytes
2-bytes
4-bytes
2-bytes
4 bytes
2-bytes
4-bytes
2-bytes
4-bytes
2-bytes
4 bytes
2-bytes
4-bytes

01-GRP-1

02-GRP-2

03 CHAR1 PIC X(5)

03 FILLER PIC X (slack byte)

03 SYNC2-1 PIC S9(4) BINARY SYNC

03 SYNC2-1 PIC S9(4) BINARY SYNC

02 FILLER PIC X(2) (slack byte)

02 GRP-3 (1)

02 GRP-4 (1)

04 SYNC2-1 PIC S9(4) BINARY SYNC

04 CHAR-2(1) PIC X

03 CHAR-3(1) PIC X(2)

03 FILLER PIC X (slack byte)

02 GRP-3 (2)

02 GRP-4 (2)

04 SYNC4-1(2) PIC S9(9) BINARY SYNC

04 CHAR-2(2) PIC X

03 CHAR-3(2) PIC X(2)

03 FILLER PIC X (slack byte)

02 CHAR-4 PIC X

Slack Bit

When a SYNCHRONIZED clause has been specified in an internal Boolean item, an area is divided into units of one byte and begins with a one-byte boundary allocated to the internal Boolean item. A slack bit may be inserted in the record of this item type.

Rules for Insertion of a Slack Bit

A slack bit is inserted in accordance with the following rules:

1. The slack bit is inserted in one of the following positions:

   a. When the record contains an internal Boolean item in which a SYNCHRONIZED clause has been specified, between the right end of the padded internal Boolean item and the next byte boundary.

   b. When a data item (a data item other than an internal Boolean item and no SYNCHRONIZED clause has been specified) allocated to the byte boundary immediately follows an internal Boolean item where no SYNCHRONIZED clause record has been specified, between the right end of the internal Boolean item and the next byte boundary.

   c. When the last data item subordinate to a group item is an internal Boolean item, between the right end of the internal Boolean item and the next byte boundary.

2. The slack bit size is determined by the following procedure:

   a. First, the number of Boolean positions containing internal Boolean items is determined. If contiguous internal Boolean items in which no SYNCHRONIZED clause has been specified in condition (1)(b) above, the program finds the total number of Boolean positions containing internal Boolean items in the sequence. If the contiguous internal Boolean items in which no SYNCHRONIZED clause has been specified before the last internal Boolean item in condition (1)(c) above, the program finds the total number of Boolean positions containing internal Boolean items in the sequence.

   b. The number of Boolean positions found at (a) is divided by 8.

   c. If the remainder from the division at (b) is zero, no slack bit is required. If the remainder is not zero, a slack bit is inserted. Assuming the remainder from the division at (b) is r, then the slack bit size is 8 - r bit(s).

3. The slack bit is a FILLER having the same level-number as the data item level-number immediately preceding the slack bit. The slack bit size is included when counting the group item size where the slack bit was inserted.

4. When a group item specifys an OCCURS clause has a subordinate internal Boolean item, a slack bit is inserted. Each occurrence of the table elements is regarded as a group item and the procedure in 1.c above is effected.

5. A slack bit is automatically inserted in an output file, working-storage section, or constant section. An input file and linkage section are regarded as having a slack bit, so the slack bit must be taken into account when writing a record description entry.

Example of Insertion of a Slack Bit (1)

Below is an example of a slack bit insertion for the following record description entry.

```
01  RECORD-A.
    02  DATA-A1.
      03  DATA-A11  PIC 1(4) BIT.
      03  DATA-A12  PIC 1(3) BIT SYNC.
      03  DATA-A13  PIC 1(5) BIT.
    02  DATA-A2  PIC 1(4) BIT.
```

DATA-A12 is allocated to a one-byte boundary, so four slack bits are inserted between DATA-A11 and DATA-A12.

A one-byte area is allocated to DATA-A12, so five slack bits are inserted between DATA-A12 and DATA-A13.

DATA-A13 is the last internal Boolean item subordinate to DATA-A1, so three slack bits are inserted after DATA-A13.

The above record description entry is equivalent to the example shown below.

```
01  RECORD-A.
    02   DATA-A1.
      03  DATA-A11  PIC 1(4) BIT.
      03  FILLER    PIC 1(4) BIT.     <-- Slack bit
      03  DATA-A12  PIC 1(3) BIT.
      03  FILLER    PIC 1(5) BIT.     <-- Slack bit
      03  DATA-A13  PIC 1(5) BIT.
      03  FILLER    PIC 1(3) BIT.     <-- Slack bit
    02  DATA-A2  PIC 1(4) BIT.
```

Example of Insertion of a Slack Bit (2)

Below is an example of a slack bit insertion in a group item where an OCCURS clause has been specified for the following record description entry.

```
01  RECORD-B.
    02  DATA-B1 OCCURS 10 TIMES.
      03  DATA-B11  PIC 1(5) BIT.
      03  DATA-B12  PIC 1(5) BIT.
```

DATA-B1 is allocated to begin on a one-byte boundary where DATA-B1 is repeated. Therefore, six slack bits are inserted after DATA-B12. The slack bits inserting method is the same as the repetition of DATA-B1.

When an OCCURS clause has been specified in an internal Boolean item containing no SYNCHRONIZED clause, no slack bit is inserted between each table element occurrence. For example, no slack bit is inserted between each occurrence of DATA-C1 in the case shown below.

```
02  DATA-C1 PIC 1(3) BIT OCCURS 6.
```

Uniqueness of Reference

All user-defined words assigned in a program must be unique. The method for making user-defined words unique is called "uniqueness of reference."

Qualification

If two or more user-defined words having identical spelling are in the same name scope, the user-defined words must be qualified. This is done by adding at least one high-order name according to the name hierarchy. The section titled "Scope of a name," explains the scope of a name.

[Format 1] Making a data-name, condition-name, or index-name unique:

$$\left\{\begin{array}{l}\text{data-name-1}\\ \text{condition-name-1}\\ \text{index-name-1}\end{array}\right\}\left\{\begin{array}{l}\left\{\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\text{data-name-2}\right\}\ldots\left[\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\text{file-name-1}\right]\\ \left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\text{file-name-1}\end{array}\right\}$$

[Format 2] Making a paragraph-name unique:

$$\text{paragraph-name-1}\left\{\begin{array}{c}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\}\text{section-name-1}$$

[Format 3] Making a text-name unique:

$$\text{text-name-1} \quad \left\{\begin{array}{l}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\} \quad \text{library-name-1}$$

[Format 4] Making the special register used for the sequential I-O module unique:

$$\underline{\text{LINAGE-COUNTER}} \left\{\begin{array}{l}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\} \text{file-name-2}$$

[Format 5] Making the special register used with the report writer module unique:

$$\left\{\begin{array}{l}\underline{\text{PAGE-COUNTER}}\\ \underline{\text{LINE-COUNTER}}\end{array}\right\} \quad \left\{\begin{array}{l}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\} \quad \text{report-name-1}$$

[Format 6] Making a data-name in a report unique:

$$\text{data-name-3}\left\{\begin{array}{l} \begin{Bmatrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{Bmatrix}\ \text{data-name-4}\ \left[\begin{Bmatrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{Bmatrix}\ \text{report-name-2}\right] \\ \begin{Bmatrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{Bmatrix}\ \text{report-name-2} \end{array}\right\}$$

[Format 7] Making the special register used for the presentation f module unique:

$$\begin{Bmatrix}\underline{\text{EDIT-MODE}}\\ \underline{\text{EDIT-OPTION}}\\ \underline{\text{EDIT-COLOR}}\\ \underline{\text{EDIT-STATUS}}\\ \underline{\text{EDIT-CURSOR}}\end{Bmatrix}\ \begin{Bmatrix}\underline{\text{IN}}\\ \underline{\text{OF}}\end{Bmatrix}\ \text{identifier-1}$$

Rules Common to Format 1 to Format 7

1. IN or OF and the word following are together referred to as a “qualifier.” User-defined words not unique must have a sequence of qualifiers before the user-defined words can be referenced.

2. Qualifiers must continue to be added to the user-defined words until it becomes unique. However, not all qualifiers for making the word unique must be added.

3. A user-defined word not needing qualification may be qualified.

4. IN and OF are synonymous.

Rules for Format 1

1. The qualifier must be one of the following:

   a. When qualifying data-name-1 with data-name-2, data-name-2 must be the data-name of a group item subordinate to data-name-1.

   b. When qualifying condition-name-1 with data-name-2, data-name-2 must be the data-name of a conditional variable associated with condition-name-1.

   c. When qualifying index-name-1 with data-name-2, data-name-2 must be the data description entry data-name where index-name-1 was written in the INDEXED BY phrase.

   d. When qualifying data-name-1, condition-name-1, or index-name-1 with file-name-1, file-name-1 must be the file-name of a file description entry or sort-merge file description entry associated with the item (data-name-1, condition-name-1, or index-name-1) to be qualified.

2. Specify qualifiers in order from low-order qualifiers to high-order qualifiers according to the hierarchy going from left to right.

3. A record-name can be specified for data-name-1 or data-name-2.

4. When qualifying a condition-name, use the conditional variable hierarchy associated with the condition-name as the qualifier.

Rules for Format 2

1. When explicitly referencing a paragraph-name, the paragraph-name must not be defined twice in one section.
2. When referencing a defined paragraph-name in the same section as the paragraph-name, the paragraph-name need not be qualified.
3. SECTION must not be added to section-name-1.
4. A paragraph-name or section-name cannot be referenced from another program.

Rules for Format 3

When using two or more COBOL libraries, a text-name must be qualified by a library-name before being referenced.

Rules for Format 4

When a LINAGE clause has been written to two or more file description entries, LINAGE-COUNTER must be qualified with a file-name before the LINAGE-COUNTER (counts the number of lines) special register can be referenced.

Rules for Format 5

1. When two or more report description entries have been written, LINE-COUNTER must be qualified with a report-name before the LINE-COUNTER (indicates the line number) special register can be referenced in the procedure division.

2. When LINE-COUNTER has been referenced without first being qualified in the report section, LINE-COUNTER is implicitly qualified by the report-name. Before referencing LINE-COUNTER for another report in the report section, LINE-COUNTER must be explicitly qualified by the report-name.

3. When two or more report description entries have been written, PAGE-COUNTER must be qualified with a report-name before the PAGE-COUNTER (counts the pages) special register can be referenced in the procedure division.

4. When PAGE-COUNTER has been referenced without first being qualified in the report section, PAGE-COUNTER is implicitly qualified by the report-name. Before referencing PAGE-COUNTER for another report in the report section, PAGE-COUNTER must be explicitly qualified by the report-name.

Rules for Format 6

1. The qualifier must be one of the following:

   a. When qualifying data-name-3 with data-name-4, data-name-4 must be the group item subordinate data-name to data-name-3.

   b. When qualifying data-name-3 with report-name-2, report-name-2 must be the report description entry report-name associated with data-name-3.

2. Specify qualifiers in hierarchy order from low-order qualifiers to high-order qualifiers moving from left to right.

Rules for Format 7

1. The EDIT-MODE, EDIT-OPTION, EDIT-COLOR, EDIT-STATUS, and EDIT-CURSOR special registers must be qualified before being referenced.
2. Identifier-1 must be a data item defined as having an item control field. See “COBOL85 User's Guide” for more information on the item control field.
3. Identifier-1 must not be reference modified.
4. The necessary qualifiers and subscripts must be added to identifier-1.

Subscripting

Subscripting is used for table element unique reference. A subscript is added to data-name or condition-name before the table elements is referenced.

[Format]

```
                              ┌ integer-1                                    ┐
                              │                   ┌ + ┐                      │
┌ data-name-1         ┐       │ data-name-2     [ {   } integer-2]           │
│                     │    (  │                   └ - ┘                      │   ...)
│                     │       │                   ┌ + ┐                      │
└ condition-name-1    ┘       │ index-name-1    [ {   } integer-3]           │
                              └ ALL                   └ - ┘                  ┘
```

Note: In the above format, data-name-1 and condition-name-1 are shown for clarity; they are not part of the subscript.

Syntax Rules

1. Data-name-1 must be a data item subordinate to a data item where an OCCURS clause has been specified or a similar data item.
2. The data item (conditional variable) associated with condition-name-1 must be a data item where an OCCURS clause was specified or a similar data item. When subscripting is required to reference the conditional variable, subscripting having an identical combination as the conditional variable must be added to the condition-name.
3. A table element must be subscripted before being referenced except in the following cases:
    a. Subject in a SEARCH statement
    b. REDEFINES clause
    c. KEY IS phrase in an OCCURS clause
4. The number of subscripts must be the same as the number of OCCURS clauses written in the data description entry including the table elements. When two or more subscripts are required, write the subscripts in order from high-order dimensions to low-order dimensions in the table starting from left to right.
5. Index-name-1 must be the index-name written in the OCCURS clause INDEXED BY phrase of the data description entry including the table elements.
6. Data-name-2 must be an integer item.
7. Data-name-2 can be qualified.
8. The plus sign can be added to integer-1.

9. The subscript ALL can be used only when writing an function argument. ALL cannot be written as the subscript for condition-name-1.
10. The sum-counter, line-counter, and page-counter cannot be used as a subscript in a report section.
11. Data-name-1 and condition-name-1 can be qualified.

General Rules

1. One dimension corresponds to one OCCURS clause. The occurrence number indicates the element of the table elements in one dimension is being referenced. The maximum occurrence count specified in the OCCURS clause may be represented by n. In this case, the first table element in the dimension is represented by occurrence number 1 and the second table element is represented by occurrence number 2. The final table element in the dimension is represented by occurrence number n.
2. The maximum number of subscripts (the maximum number of dimensions) is 7.
3. When integer-1 or data-name-2 has been written in a subscript, the subscript value represents the table elements occurrence number.
4. When index-name-1 has been written in a subscript, the subscript value represents the value corresponding to the table elements occurrence number.
5. The index-name-1 value must be initialized before being used as a subscript. The initial value of the index-name-1 can be set by a PERFORM statement containing a VARYING phrase, a SEARCH statement containing an ALL phrase, or a SET statement. The index-name value can be modified by the PERFORM statement, SEARCH statement, or SET statement.

6. When integer-2 has been written, the subscript has one of two values. If the operator is “+”, the value is obtained by adding the value of integer-2 to the value of data-name-2 (occurrence number). If the operator is “-”, the value is obtained by subtracting the value of integer-2 from the value of data-name-2.
7. When integer-3 has been written, the subscript has one of two values. If the operator is “+”, the value is obtained by adding the value of integer-3 to the index-name-1 value (the value corresponding to the occurrence number). If the operator is “-”, the value is obtained by subtracting the integer-3 value from the index-name-1 value.
8. When ALL has been written in the subscript, the program references all table elements in the dimension corresponding to the subscript.

Reference Modification

Reference modification references part of a data item or function value. It adds a reference modifier to the data-name or function-identifier.

[Format 1] Reference modification of data item:

```
data-name-1 (high-order-end-character-position: [length])
```

[Format 2] Reference modification of function value:

```
FUNCTION  function-name-1 [({ argument-1 } ...)]
          (high-order-end-character-position: [length])
```

In the above formats, “data-name-1” and “FUNCTION function-name-1[({argument-1} ...)]” are shown for clarity; they are not part of the reference modifier.

Syntax Rules

1. The application of data-name-1 must be an elementary item or group item for presentation.
2. The high-order-end-character-position and length must be an arithmetic expression.
3. Data-name-1 can be qualified or subscripted.
4. Function-name-1 must be an alphanumeric function.

HP Win The name of a national function can be specified.

General Rules

1. Reference modification redefines part of the data item in data-name-1 or a temporary data item having a function value (the source data item for reference modification) as a unique data item (the data item resulting from reference modification).
2. Set the character position starting reference modification in high-order-end-character-position with 1 as the high order end source data item character position for reference modification. The high-order-end-character-position value must be an integer in the following range:

$$1 \leq \begin{pmatrix} \text{value of} \\ \text{high-order-end-} \\ \text{character-position} \end{pmatrix} \leq \begin{pmatrix} \text{number of character positions} \\ \text{for source data item for} \\ \text{reference modification} \end{pmatrix}$$

3. Set the reference modification range in “length.” The value of “length” must be an integer in the following range:

$$1 \leq \text{(length)}$$

$$\leq \left(\begin{array}{l}\text{number of character positions}\\\text{for source data item for}\\\text{reference modification}\end{array}\right) - \left(\begin{array}{l}\text{value of}\\\text{high-order-end-}\\\text{character-position}\end{array}\right) + 1$$

4. When “length” is omitted, the portion from the high order end character position to the beginning low order end character position of the source data item for reference modification becomes the data item resulting from reference modification.

5. When a subscript is added to data-name-1, the reference modifier is evaluated immediately after the subscript evaluation. When ALL has been specified in the data-name-1 subscript, reference modifiers are applied for all table elements in the dimension corresponding to the subscript.

6. The data item resulting from reference modification is regarded as an elementary item having no JUSTIFIED clause.

7. After the function value has been determined, the reference modifier in Format 2 is applied to a temporary data item having a function value.

8. In Format 1 the data item resulting from reference modification is regarded as having the following category and class:

   a. When the category of data-name-1 is alphabetic character, numeric character, numeric edited, alphanumeric character, or alphanumeric edited, the data item category and class resulting from reference modification are regarded as alphanumeric characters.

   b. When the category of data-name-1 is National or National edited, the data item category and class resulting from reference modification are regarded as national language.

   c. When the category of data-name-1 is Boolean, the data item category and class resulting from reference modification are regarded as alphanumeric characters.

   d. When data-name-1 is a group item, the data item category and class resulting from reference modification are regarded as alphanumeric characters.

9. In Format 2, the data item category and class resulting from reference modification are regarded as alphanumeric characters or national characters.

Pointer

A pointer is used for referencing the data-name or condition-name defined in the BASED-STORAGE SECTION. A pointer involves a pointer qualifier to the data-name or condition-name. The pointer qualifier indicates the data item address storage area defined in the BASED-STORAGE SECTION.

[Format]

```
         ⎧ data-name-1                             ⎫
[(] ...  ⎨ (data-name-2 ( { subscript } ... ) )    ⎬ ->
         ⎩ ADDR-function                           ⎭

   ⎡⎧ data-name-1                             ⎫       ⎤       ⎧ data-name-3      ⎫
   ⎢⎨                                         ⎬ ) ->  ⎥ ...   ⎨                  ⎬
   ⎣⎩ (data-name-2 ( { subscript } ... ) )    ⎭       ⎦       ⎩ condition-name-1 ⎭
```

In the above format, data-name-3 and condition-name-1 are shown for clarity; they are not part of the pointer qualifier. The sign “->“ is a key word, but is not underlined to avoid confusion with other symbols.

Syntax Rules

1. Data-name-1 and data-name-2 must be pointer data items.
2. The symbol “->“ is called the pointer qualifier symbol. A space can be written immediately before or after the pointer qualifier symbol. The pointer qualifier symbol must be contained on a single line.

3. When writing two or more pointer qualifier symbols, parentheses must enclose the portion preceding the second and subsequent pointer qualifier symbols.

   ```
   Examples: PTR1->X
             (PTR1->PTR2)->X
            ((PTR1->PTR2)->PTR3)->X
   ```

4. Data-name-1 and data-name-2 may be qualified.

5. Data-name-3 and condition-name-1 must be defined in the BASED-STORAGE SECTION.

6. Data-name-3 and condition-name-1 can be qualified or subscripted. Data-name-3 can be reference modified.

General Rules

1. When data-name-3 is a data item where the BASED ON clause has been omitted or is a data item subordinate to such a data item, a pointer qualifier must be added to data-name-3 before being referenced.

2. When the condition variable associated with condition-name-1 is a data item where the BASED ON clause was omitted or is a data item subordinate to such a data item, a pointer qualifier must be added to condition-name-1 before being referenced.

3. An implicit pointer qualifier can be defined by writing a BASED ON clause in the BASED-STORAGE SECTION data description entry. A pointer using an implicit pointer qualifier without writing a pointer qualifier is called “An implicit pointer.” Writing a pointer qualifier is called “An explicit pointer.”

4. A data item where a BASED ON clause has been specified or a data item subordinate to such a data item allows either an

implicit or an explicit pointer qualifier to be added. When the conditional variable associated with the condition-name is a data item where a BASED ON clause has been specified or is a data item subordinate to such a data item, either an implicit or an explicit pointer qualifier can be added to the condition-name.

Identifier

A data-name made unique by the addition of a qualifier, by subscripting, by reference modification, or by a pointer addition is called an “identifier.” When creating a data-name uniqueness of reference, the data-name must be written in the identifier format.

[Format]

[pointed qualifier] data-name-1

$$\left[\left\{\begin{array}{l}\underline{\text{IN}}\\ \\ \underline{\text{OF}}\end{array}\right\}\text{data- name-2}\right]\ldots\left[\left\{\begin{array}{l}\underline{\text{IN}}\\ \\ \underline{\text{OF}}\end{array}\right\}\left\{\begin{array}{l}\text{file-name-1}\\ \\ \text{report-name-1}\end{array}\right\}\right]$$

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

1. IN and OF are synonyms.

2. An identifier can be written where “identifier-n” is indicated in [Format]. A data-name where a qualifier, subscript, reference modifier, or pointer has been added to provide uniqueness of reference must be written in “identifier-n” indicated in [Format].

3. A pointer qualifier can be written only when data-name-1 has been defined in the BASED-STORAGE SECTION.

Uniqueness of Reference of Condition-name

A condition-name must be made unique by the qualifier addition, subscript, or pointer except where uniqueness of reference can be guaranteed by the rules for scope of a name.

[Format]

[pointed qualifier] condition-name-1

$$\left[\begin{Bmatrix}\underline{\text{IN}}\\ \\ \underline{\text{OF}}\end{Bmatrix}\ \text{data- name-1}\right]\ldots\ \left[\begin{Bmatrix}\underline{\text{IN}}\\ \\ \underline{\text{OF}}\end{Bmatrix}\ \text{file-name-1}\right]$$

[({subscript} ...)]

1. IN and OF are synonyms.
2. A condition-name can be written where “condition-name-n” is indicated in [Format]. A condition-name where a qualifier, subscript, or pointer has been added to provide uniqueness of reference must be written where “condition-name-n” is indicated in [Format].
3. A pointer qualifier can be written only when condition-name-1 is defined in the BASED-STORAGE SECTION.

Function-identifier

A function-identifier is a format used for a function. The function-identifier is treated as a temporary data item having a function value.

[Format]

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

1. Argument-1 must be an identifier, literal, or arithmetic expression.
2. Reference modifier can be written for an alphanumeric character function or a national function. It is written to provide function value reference modification.
3. The section titled “General Rules for Functions,” explains the locations where a function-identifier can be written.
4. The function argument is evaluated according to the sequence position where argument-1 was written.
5. A function-identifier or an expression including a function-identifier can be written in argument-1. Function-name-1 can also be written in argument-1.

Reference Format

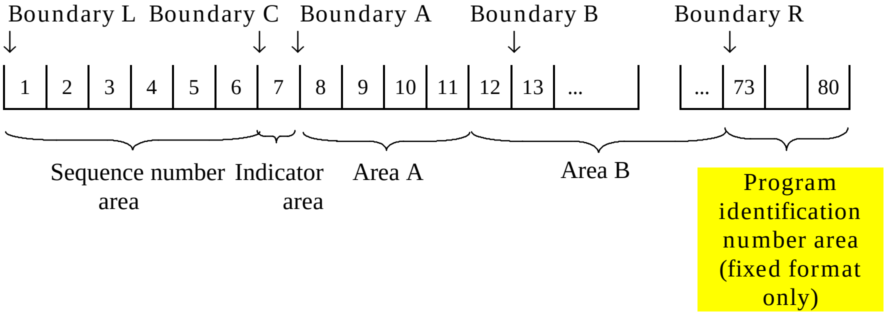
The format used when writing a COBOL source program is called the “reference format.” A COBOL source program must be written according to the reference format. When using the source text manipulation module, the COBOL library must also be written according to the reference format.

Configuration of a Line

The reference format is a rule stating which element is being placed in which position in a line. There are three types of reference formats:

- Fixed format
- Variable format
- Free format Win32

The diagram below shows the configuration of a line, if either fixed or variable reference format is selected.



- Boundary L is immediately to the left of the high order end character position in the line.
- Boundary C is between character positions 6 and 7 in the line.
- Boundary A is between character positions 7 and 8 in the line.
- Boundary B is between character positions 11 and 12 in the line.
- Boundary R is one of the following positions:
- In the fixed format, immediately to the right of character position 72 in the line.
- In the variable format, immediately to the right of the last character position in the line. See "Appendix B, System Quantity Restrictions of the System," for more information on the last character position in the line.

Sequence Number Area

"Sequence number area" refers to the six character positions between boundary L and boundary C. It identifies a line in the source program. Any character belonging to a computer character set can be written in the sequence number area. The contents of the sequence number area need not be in a specified sequence or be unique.

Indicator Area

- "Indicator area" refers to the character position between boundary C and boundary A. An asterisk "*", slash "/", hyphen "-", or "D" can be written in the indicator area.
- A line where an asterisk or slash has been written in the indicator area is called a "comment line." The contents of area A and area B in a comment line are regarded as annotation.
- A hyphen in the indicator area indicates the first character in the line is joined to the last character of the previous line.
- A line where "D" has been written in the indicator area is called a "debugging line." A debugging line leaves debugging information in the source program.

Area A

"Area A" refers to the four character positions between boundary and boundary B.

Area B

"Area B" refers to the area between boundary B and boundary R.

Program Identification Number Area

Only the fixed format has a program identification number area. "Program identification number area" refers to the eight character positions from the position immediately to the right of boundary R (character position 73) to character position 80.

Reference Format for Area A and Area B

Some COBOL words must start in area A and others in area B.

Words to be Started in Area A

The following words must start in area A:

- Division header
- Section header
- Paragraph header
- Paragraph-name
- Level indicator (FD, SD or RD)
- Level-numbers 01 and 77
- Start of declarative (DECLARATIVES.) and end of declarative (END DECLARATIVES.)
- End program header

Words to be Started in Area B

The following words must start in area B:

- Statement (except the COPY and REPLACE statements, which can originally be written in area A)
- Entry (except entries starting with a level indicator or data description entries whose level-number is 01 or 77, which must be originally written in area A. Data description entries whose level-number is not 01 or 77 can also be originally written in area A.)
- Comment entry

Rules for Each Division

In addition to the rules stated above, the following rules also apply to the various divisions:

1. In the identification division, a paragraph header and entry or comment entry can be written in the same line. Two or more comment entries can be written in the same line. A single comment entry can be written over two or more lines.
2. In the environment division, a paragraph header and entry can be written in the same line. Two or more entries can be written in the same line. A single entry can be written over two or more lines.
3. In the data division, two or more entries can be written in the same line. A single entry can be written over two or more lines. When the level-number is a single digit, it can be written with a single numeric character. For clarity, the starting position of the level-number can be changed as additional levels are added.
4. In the procedure division, a paragraph-name and sentence can be written in the same line. Two or more sentences can be written in the same line. A single sentence can be written over two or more lines.

Blank Line

A line whose indicator area (areas A and B) is all spaces is called a “blank line.” A blank line can be written anywhere in a source program and in library text.

Comment Line

A line where an asterisk (*) or slash (/) has been written in the indicator area is called a “comment line.” A comment line can be written in any position after the identification division header or in any library text position.

Any characters belonging to a computer character set can be written in area A and B of a comment line. A comment line is regarded a comment, but is printed in the compile list.

A comment line where an asterisk has been written in the indicator area is printed immediately below the previous line. A comment line where a slash has been written in the indicator area is printed after a form feed (on a new page).

Joining Lines

A character-string consisting of sentences, entries, and clauses can be continued from one line to area B in the next line. A line to where a character-string is continued is called a “continuation line.” A line from where a character-string was continued is called a “continued line.”

The following character-strings can be continued to a continuation line.

- COBOL words excluding National user-defined words
- Literal
- Character-string in a PICTURE clause

When continuing the above character strings to a continuation line, the indicator area in the continuation line must be a hyphen (-). Area A in the continuation line must be spaces. A hyphen in the indicator area indicates the first character not being a space in area B of the continuation line continues on from the last character not being a space in the continued line (excluding a comment line and blank line).

Continuation of a nonnumeric literal, hexadecimal nonnumeric literal, national nonnumeric literal, or Boolean literal to the continuation line must also conform to the following rules:

1. The first character not being a space in area B of the continuation line must be a quotation mark. The continuation of the literal must start at the character position immediately to the right of the quotation mark.
2. Any spaces to the end of area B in the continued line are regarded as part of the literal. The literal must be written so the last character in area B of the continued line continues to the character immediately following the first quotation mark in the continuation line.

To continue a line when the character-string fits into one line and is not to be continued to the next line, a hyphen must not be written in the indicator area. When no hyphen exists in the indicator area, the position immediately preceding a character not being a space in the continuation line is regarded as a space.

A separator (i.e., one of the following character-strings) must be written in a single line:

- The == pseudo-text delimiter
- A separator indicating the start of a literal value (X”, N”, NC”, NX” or B”)
- The -> pointer qualification symbol

The diagram below shows an example of continuing a line.

```
Indicator area

       Boundary A                                                   Boundary R
 ↓      ↓                                                                    ↓
  77    XX  PIC X(60) VALUE "ABCDE12345ABCDE12345ABCDE12345ABCDE12345
 -                           "ABCDE12345ABCDE12345".                       ...1
        MOVE X
 -                X TO YY.                                                 ...2
        MOVE XX TO
                     YY.                                                   ...3
```

1. The nonnumeric literal is continued.
2. The user-defined word XX is continued.
3. The line is continued between two words. A hyphen must not be written in the indicator area.

Debugging Line

A line where D has been written in the indicator area is called a “debugging line.” A debugging line leaves debugging information in the source program. The debugging line can be written after the OBJECT-COMPUTER paragraph in the IDENTIFICATION DIVISION.

Area A and area B in the debugging line must be written according to COBOL syntax. When both area A and area B in the debugging line are spaces, the debugging line is regarded as a blank line.

Inclusion of the WITH DEBUGGING MODE clause in the identification division source computer paragraph specifies the debugging line is to be compiled. That is, the debugging line is compiled only when the WITH DEBUGGING MODE clause was written. When this clause was omitted, the debugging line is regarded as a comment line.

At compilation, the compiler determines whether the WITH DEBUGGING MODE clause was included after processing the COPY statement and REPLACE statement.

In-line Comment

To write a comment in the same line as a statement or entry, use an in-line comment to write the comment at the right side of the line. An in-line comment is the portion starting with the two characters *> and ending at the character position immediately to the left of boundary R in the same line. An in-line comment can be written in the identification division header line or any line after the header line or in any library text line.

A separator space must be written immediately before the *> symbol indicating the start of an in-line comment.

Any character belonging to a computer character set can be written after the *> symbol. The in-line comment is regarded as a comment.

WIN32

Reference Format for Free Format

This section explains reference format for free format.

Free format is specific to [Win32].

Configuration of a Line

In the reference format for free format, the source program text can be written anywhere in a line with the exception of the particular rules for comment, debugging line, and joining lines. The number of character positions in each line can be written from 0 to 251. (A national character indicates two character positions.)

Blank Line

A line where all spaces have been written is called a “blank line.” A blank line can be written anywhere in a source program and library text.

Comment

Combining two COBOL characters “*>” indicates comment.

A line called a “comment line” consists of either none or only one or more blanks before the comment .

A comment called a “in-line comment” consists of one or more COBOL words immediately followed by only one or more blanks. An in-line comment can be written in the header line of the identification division or any line after the header line or in any library text line.

A separator space must be written immediately before the “*>” symbol indicating the start of an in-line comment. Any character belonging to a computer character set can be written after the “*>” symbol. The in-line comment is regarded as a comment.

Joining Lines

A character-string consisting of sentences, entries, and clause can be continued from one line to the next line. A line where a character-string is continued is called a “continuation line.” A line where a character-string was continued is called a “continued line.”

Continuation of nonnumeric literal, Boolean literal, or national nonnumeric literal can be written separately to the continuation line. When these literals have been separated at the end of a line, the separated literal must be ended in the quotation mark immediately following the hyphen. None or one or more separator space must be continued after the hyphen.

The first character not being a space in the continuation line must be a quotation mark. The continuation of the literal must start at the character position immediately to the right of the quotation mark. At this point, one or more characters of the literal must be written both in the continued line and in the continuation line.

A separator composed of two or more characters must be written in the same line. A set of quotation marks in the literal must be described in the same line. A comment line and a blank line can be written anywhere between lines including part of a literal.

Debugging Line

Combining three COBOL characters “>>D” immediately followed by a single-byte space indicates debugging indicator. A line with none or only one or more blanka before the debugging indicator is called “debugging line.” The debugging line leaves debugging information in the source program. The debugging line can be written after the OBJECT-COMPUTER paragraph in the IDENTIFICATION DIVISION. The debugging line ends at the end of the line.

The debugging line must be written according to COBOL syntax. If none or only one or more blanks have been written in the debugging line, the debugging line is regarded as a blank line.

Inclusion of the WITH DEBUGGING MODE clause in the SOURCE-COMPUTER paragraph of the identification division specifies the debugging line is to be compiled. Therefore, the debugging line is compiled only when the WITH DEBUGGING MODE clause is written. When this clause is omitted, the debugging line is regarded as a comment line.

At compilation, the compiler determines whether the WITH DEBUGGING MODE clause is included after processing the COPY statement and REPLACE statement.

Program Configuration

A program consists of the following four divisions and an end program header:

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- PROCEDURE DIVISION

A program starts with the identification division and ends with the end program header or the last line of the program.

Other programs can be written between the start and end of the program. A program contained within another program is called an “internal program.” An internal program can be written provided an inter-program communication module is used.

The scope between the outermost start and end of a program is called the “compilation unit.” One or more internal programs can be written in one compilation unit. The configuration where one program contains another program is called “program nesting.”

An internal program can itself contain an internal program. When one program (program A) contains another program (program B) immediately within it and program B also contains another program (program C) immediately within it, program B is said to be “contained directly” within program A and program C is said to be “contained indirectly” within program A.

The formats of programs in a compilation unit are shown below:

[Format 1] Compilation unit:

```
IDENTIFICATION DIVISION
[ENVIRONMENT DIVISION]
[DATA DIVISION]
[PROCEDURE DIVISION]
[Internal program] ...
[End program header]
```

[Format 2] Internal program:

```
IDENTIFICATION DIVISION
[ENVIRONMENT DIVISION]
[DATA DIVISION]
[PROCEDURE DIVISION]
[Internal program] ...
End program header
```

1. When writing internal programs in a compilation unit, an end program header must be written for each program. For example, when program A contains program B and program B in turn contains program C, an end program header must be written separately for each of the programs A, B, and C.

2. The outermost program in a compilation unit must end with an end program header. However, this header can be omitted from the last program in a series of compilation unit programs.

3. The start of each division is indicated by a division header. The end of each division is indicated by one of the following:

   a. The next division header in the same program

   b. The identification division header indicating the start of another internal program

   c. The end program header

   d. A physical position where there is no other line.

Chapter 2. COBOL Modules

COBOL has the following ten modules:

- Nucleus module
- Input-output module (sequential, relative and indexed)
- Inter-program communication module
- Sort-merge module
- Source text manipulation module
- Presentation file module
- Built-in function module
- Screen handling module
- Command line argument and environment variable operation module
- Report writer module

This chapter gives an overview of each module.

Nucleus

The nucleus is an elementary module for converting, comparing, and performing operations on data. The nucleus consists of the following modules:

- Data transcription: MOVE statement
- Arithmetic operation: COMPUTE statement, ADD statement, DIVIDE statement, MULTIPLY statement, and SUBTRACT statement
- Optional processing: IF statement, EVALUATE statement, and CONTINUE statement
- Branch: GO TO statement and ALTER statement
- Repeat processing: PERFORM statement, EXIT statement, and EXIT PERFORM statement
- Table handling: SEARCH statement and SET statement
- Initialization of data item: INITIALIZE statement
- Character-string handling: INSPECT statement, STRING statement, and UNSTRING statement
- Simple input-output: ACCEPT statement and DISPLAY statement
- End of program run: STOP RUN statement
- Pointer handling: No statement
- Floating-point handling: No statement

Include the following descriptions as required when using a nucleus in a program.

Program Description	Main Facility
IDENTIFICATION DIVISION	Specifies the name of the program
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SOURCE-COMPUTER paragraph	Specifies the debugging mode
OBJECT-COMPUTER paragraph	Specifies the large-to-small sequence of characters
SPECIAL-NAMES paragraph	Defines the mnemonic-name, alphabet-name, symbolic-character, symbolic-constant, class-name, currency symbol, decimal point, positioning unit name, and print mode name
DATA DIVISION	
BASED-STORAGE SECTION	
77-level description entry and record description entry	Defines the data item to which a pointed is to be added and referenced
WORKING-STORAGE SECTION	
77-level description entry and record description entry	Defines the data item
CONSTANT SECTION	
77-level description entry and record description entry	Defines a data item with a constant

Program Description	Main Facility	
PROCEDURE DIVISION		
Procedure portion		
MOVE	Performs transcription	
COMPUTE	Performs arithmetic operation	
ADD	Performs addition	Arithmetic operations
DIVIDE	Performs division	
MULTIPLY	Performs multiplication	
SUBTRACT	Performs subtraction	
IF	Distributes processing according to the truth value of a condition	Optional processing
EVALUATE	Distributes processing according to the truth value of several conditions	
CONTINUE	No-operation statement	
GO TO	Shifts control to another location in the PROCEDURE DIVISION	Branch
ALTER	Changes the branch destination of a GO TO statement	
PERFORM	Repeats a procedure	Repeat processing
EXIT	Specifies the common exit for outer PERFORM statements	
EXIT PERFORM	Specifies the common exit for inner PERFORM statements	
SEARCH	Retrieves a table element	Table handling
SET	Sets the coordinates of a table element	
INITIALIZE	Initializes a data item	
INSPECT	Verifies a character-string	Character-string handling
STRING	Links a character-string	
UNSTRING	Decomposes a character-string	
ACCEPT	Inputs data from the hardware unit	Simple input-output
DISPLAY	Displays data on the hardware unit	
STOP RUN	Ends a program	
End program header	Specifies the end of a program	

Transcription of Data

The MOVE statement transcribes the data item or literal contents to another data item. It also provides tailoring to the data item attributes in the receiving area, data alignment , data type conversion, and editing.

Examples of the MOVE statement are given below.

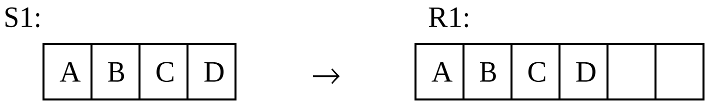
[DATA DIVISION]

```
77 S1 PIC X(4).
77 R1 PIC X(6).
77 S2 PIC 9(2)V99.
77 R2 PIC 9(4)V9(4).
77 S3 PIC 9(6).
77 R3 PIC ZZZ, ZZ9.
```

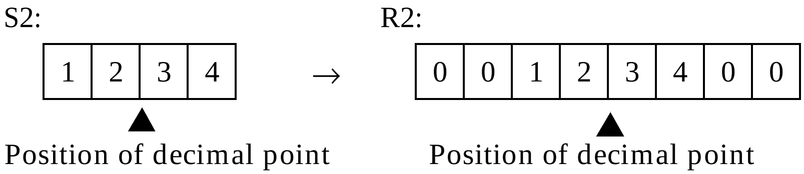
[PROCEDURE DIVISION]

```
MOVE S1 TO R1.        ...1
MOVE S2 TO R2.        ...2
MOVE S3 TO R3.        ...3
```

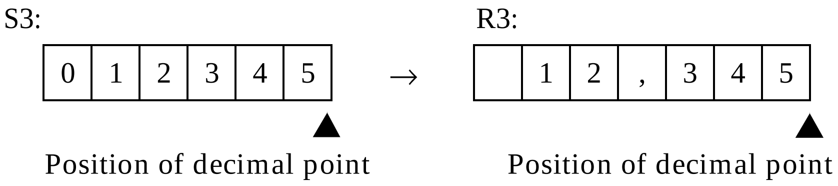
The MOVE statement at 1 transcribes S1 to R1. It positions the data from the left end and fills the remaining positions with spaces. (transcribing of nonnumeric items)



The MOVE statement at 2 transcribes S2 to R2 with decimal alignment. (transcribing of numeric data items)



The MOVE statement at 3 transcribes S3 to R3 with decimal alignment and editing. (transcribing of numeric edited data items) It replaces the leading zero with a space and inserts a comma.



Arithmetic Operation

Use the ADD statement to perform addition, the SUBTRACT statement to perform subtraction, the MULTIPLY statement to perform multiplication, the DIVIDE statement to perform division, and the COMPUTE statement to perform an arithmetic operation. The COMPUTE statement can also be used to perform a Boolean operation.

The ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE statements are generically called "arithmetic statements." An arithmetic statement can round off the result of an arithmetic operation and also perform an overflow check. To round off a value, write the ROUNDED phrase. To run an overflow check, write the ON SIZE ERROR phrase.

Examples of arithmetic statements are given below.

ADD C TO A.	**Stores the result of A + C in A.**
DIVIDE C BY D GIVING A ROUNDED.	**Rounds off the quotient of C/D and stores the result in A.**
COMPUTE A = C / D + E * 100.	**Stores the result of C/D + E * 100 in A.**
COMPUTE Z = X AND Y.	**Stores the Boolean product of X and Y in Z.**

Optional Processing and Branching

The IF statement and EVALUATE statement start optional processing. The IF statement checks a condition and selects the next executable statement depending on the condition truth value . The EVALUATE statement checks a condition and selects one process from among several.

Use the GO TO statement to branch unconditionally from one location to another location.

Example of IF Statement

In an IF statement, write the THEN phrase to indicate processing if a condition is true and write the ELSE phrase to indicate processing if the condition is false.

An example of the IF statement is given below.

```
    IF X = Y THEN
      MOVE 0 TO Z     ...1
    ELSE
      GO TO P1        ...2
    END-IF.
      :
P1.
      :
```

1. If X=Y is true, the MOVE statement is run.
2. If X=Y is false, the GO TO statement is run and the program branches to P1.

Example of EVALUATE Statement

In an EVALUATE statement, write the WHEN phrase to indicate processing that should be selected. The processing object written immediately after EVALUATE and the processing object written immediately after WHEN are compared. If the conditions are met, the processing object written in the WHEN phrase is run. An example of the EVALUATE statement is given below.

```
EVALUATE MARKS
  WHEN 85 THRU 100 MOVE “A” TO RESULT              ...1
  WHEN 70 THRU  84 MOVE “B” TO RESULT              ...2
  WHEN 55 THRU  69 MOVE “C” TO RESULT              ...3
  WHEN OTHER       MOVE “D” TO RESULT     ...4
END-EVALUATE.
```

1. When the value of MARKS is between 85 and 100, MOVE “A” TO RESULT is run.

2. When the value of MARKS is between 70 and 84, MOVE “B” TO RESULT is run.

3. When the value of MARKS is between 55 and 69, MOVE “C” TO RESULT is run.

4. When MARKS does not meet the conditions in (1), (2) or (3), MOVE “D” TO RESULT is run.

The above EVALUATE statements can be written using a condition-name condition as follows.

```
[DATA DIVISION]
    01 MARKS PIC 9(3).
       88 RESULT-A VALUE 85 THRU 100.
       88 RESULT-B VALUE 70 THRU 84.
       88 RESULT-C VALUE 55 THRU 69.
[PROCEDURE DIVISION]
    EVALUATE MARKS
      WHEN RESULT-A MOVE “A” TO RESULT
       WHEN RESULT-B MOVE “B” TO RESULT
       WHEN RESULT-C MOVE “C” TO RESULT
       WHEN OTHER    MOVE “D” TO RESULT
    END-EVALUATE.
```

Repetition Processing

Use the PERFORM statement to repeat a processing sequence. There are two kinds of PERFORM statements:

- Outer PERFORM statement
- Inner PERFORM statement

Outer PERFORM Statement

The outer PERFORM statement specifies the statement to be repeated by the procedure-name. The outer PERFORM statement not only repeats a procedure but can also control the program at a single point.

An example of the outer PERFORM statement is given below.

```
    MOVE 0 TO TBL-SUM.
    PERFORM P1 VARYING I FROM 1 BY 1 UNTIL I > 5
            ALTER J FROM 1 BY 1 UNTIL J > 10.              ...1
      :
    PERFORM P2 THRU PX.                                    ...2
      :
 P1.
    ADD TBL-X (I, J) TO TBL-SUM.
 P2.                                  Scope of the PERFORM
       :                              statement at 1
 PX.
    EXIT.                             Scope of the PERFORM
                                      statement at 2
```

1. The PERFORM statement repeats the statement in paragraph-name P1. It sets the initial value of J to 1 and increments the value by 1 each time the statement in P1 is run. When the value of J exceeds 10, the PERFORM statement increments the value of I by 1 from its initial value of 1. The PERFORM statement repeats this process until the value of I exceeds 5.

2. The PERFORM statement runs the statements between paragraph-name P2 and PX once only. The EXIT statement indicates the procedure exit.

Inner PERFORM Statement

When using the inner PERFORM statement, write the statement to be repeated in the PERFORM statement.

An example of the inner PERFORM statement is given below. The result of the inner PERFORM statement is the same as the result explained at 1 for the outer PERFORM statement.

```
MOVE 0 TO TBL-SUM.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5           ...1
  PERFORM VARYING J FROM 1 BY 1 UNTIL J > 10        ...2
    ADD TBL-X (I, J) TO TBL-SUM
  END-PERFORM
END-PERFORM.
```

1. The PERFORM statement repeats the unconditional statement (PERFORM statement) written after the UNTIL phrase. It sets the initial value of I to 1 and increments the value by 1 each time the unconditional statement is run. When the value of I exceeds 5, the PERFORM statement stops repetition processing.
2. The PERFORM statement repeats the unconditional statement (ADD statement) written after the UNTIL phrase. It sets the initial value of J to 1 and increments the value by 1 each time the unconditional statement is run. When the value of J exceeds 10, the PERFORM statement stops repetition processing.

Table Handling

When defining data items repetition having the same attributes, the data items can be collected and defined as a table. To define a table, write the OCCURS clause in the data description entry. An element in the table is called a “table element.” A table element is subscribed before being referenced.

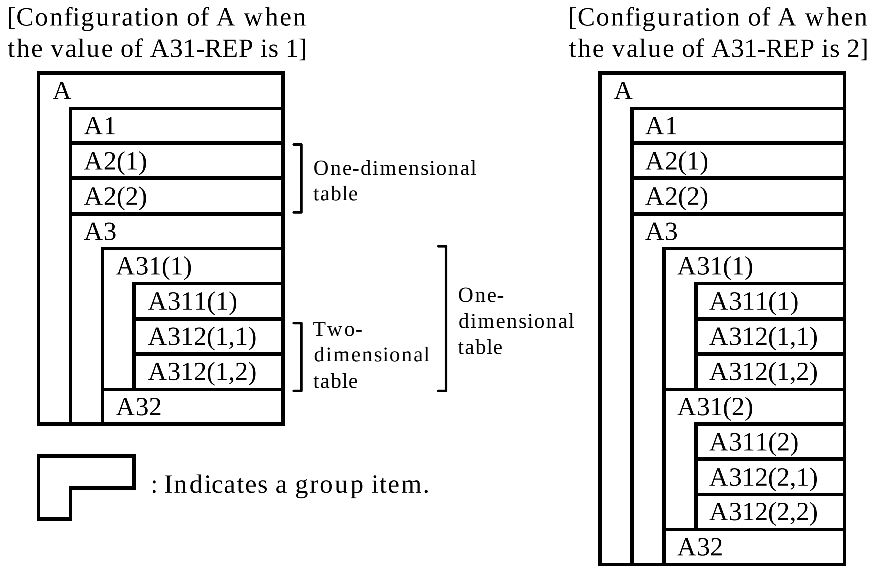
Occurrence Count for Table Elements

The occurrence count for a table element can be fixed or variable. When the occurrence count is fixed, the DEPENDING phrase in the OCCURS clause can be omitted. When the occurrence count is variable, write the DEPENDING phrase in the OCCURS clause. Specify the minimum value and maximum value for the occurrence count in the OCCURS clause. Specify the occurrence count for table elements by setting a value in the data item written in the DEPENDING phrase. The data item specified in the DEPENDING phrase is called a “variable occurrence data item.”

The variable occurrence data item in a record can be followed by another data item. The area allocated after the variable occurrence data item in a record is called the “variable position in the record.”

An example of defining a table is given below.

```
01  A.
  02  A1 PIC X.
  02  A2 PIC X(2) OCCURS 2.         ...Table with fixed occurrence count
  02  A3.
   03  A31 OCCURS 1 TO 3            ...Table with variable occurrence count
        DEPENDING ON A31-REP.
    04  A311 PIC X(4).
    04  A312 PIC X(5) OCCURS 2.     ...Table with fixed occurrence count
   03  A32 PIC X(3).                ...Variable position in record
01  A31-REP PIC 9.                  ...Data item for setting occurrence count
```



The values (n) and (n, m) are subscripts. In the subscript A312 (n, m), n corresponds to the OCCURS clause written in data description entry A31 and m corresponds to the OCCURS clause written in data description entry A312.

Reference Area of Variable Occurrence Data Item

A statement referring to a group item having a subordinate variable occurrence data item processes the following areas within the variable occurrence data item area. In the following explanation, X represents the data-name written in the DEPENDING phrase in the OCCURS clause and n represents the value of X at the processing start.

1. When the group item does not have a subordinate X, the first to the nth table elements are processed.
2. When the group item has a subordinate data-name-1, the first to the nth table elements are processed when using the group item as a sending item. When using the group item as a receiving item, the maximum value for the occurrence count is processed.

Index Name and Index Data Item

An integer, integer item, or index-name can be added as a subscript to a table element. When an integer or integer item is written as a subscript, its value represents the occurrence number of the table elements. When an index-name is written as the subscript, its value represents the value for identifying the storage location of the table elements.

The index-name identifies the table elements. It is specified in the INDEXED BY phrase in the OCCURS clause. The compiler creates and determines the index-name area. An index-name written as a subscript must be an index-name included in the table specifying the index-name. The index-name value is set in the SET statement.

The SET statement can save the value of an index-name in an index data item. An index data item is a data item for storing index-names without modification. The data description entry where the USAGE IS INDEX clause was written defines the index data item.

An example of application of the index-name and index data item is given below.

[DATA DIVISION]

```
77 B USAGE INDEX.
01 C.
  02 C1 OCCURS 3 INDEXED BY IDX1.
    03 C11 PIC X(2).
    03 C12 PIC X(3) OCCURS 3 INDEXED BY IDX2.
```

[PROCEDURE DIVISION]

```
SET IDX1 TO 1.                           ...1
SET B TO IDX1.                           ...2
SET IDX2 TO 2.                           ...3
MOVE C12(IDX1, IDX2) TO  ... .           ...4
```

1. Sets the value corresponding to occurrence number 1 in index-name IDX1 in table C1.
2. Saves the value of index-name IDX1 in index data item B.
3. Sets the value corresponding to occurrence number 2 in index-name IDX2 in table C12.
4. Adds the subscripts IDX1 and IDX2 to the table elements in table C12 and references it.

Retrieving a Table

Use the SEARCH statement to retrieve a table element meeting certain conditions. If the table element values are already in ascending or descending order, the SEARCH statement containing the ALL phrase can be used.

The following is an example of the SEARCH statement and added ALL phrase:

```
[DATA DIVISION]
   01 TBL-VALUE.
     02 FILLER .
       03 FILLER PIC X(02) VALUE “01”.
       03  FILLER PIC X(20) VALUE “APPLE”.
     02 FILLER.                                          1
       03 FILLER PIC X(02) VALUE “10”.
       03 FILLER PIC X(20) VALUE “ORANGE”.
     02 FILLER.
       03 FILLER PIC X(02) VALUE “12”.
       03 FILLER PIC X(20) VALUE “PEACH”.
   01 TBL-REF REDEFINES TBL-VALUE.
     02 TBL-DATA OCCURS 3 TIMES ASCENDING
     KEY IS G-CODE INDEXED BY IDX.           ...3          2
       03 G-CODE PIC X(02).                          4
       03 G-NAME PIC X(20) .
 [PROCEDURE DIVISION]
     SEARCH ALL TBL-DATA
       AT END DISPLAY “ERROR”
         WHEN G-CODE (IDX) = “10”                    5
           DISPLAY G-NAME (IDX)
     END-SEARCH.
```

1. Defines the value of the table TBL-VALUE.
2. Defines the table TBL-REF. TBL-REF occupies the same area as TBL-VALUE.
3. Defines the table TBL-DATA. Set the occurrence count to 3, associate the index-name IDX with TBL-DATA, and use the ASCENDING KEY phrase to specify that the values of G-CODE are arranged in ascending order.
4. Defines G-CODE and G-NAME as the table elements subordinate to TBL-DATA.
5. When the program retrieves and finds a table element meeting the conditions written in the WHEN phrase, it runs the unconditional statement (DISPLAY statement) following the conditions in the WHEN phrase. When the program does not find such a table element, it runs the unconditional statement (DISPLAY statement) in the AT END phrase.

Initialization of Data Items

Use the INITIALIZE statement to initialize a data item.

An example of an INITIALIZE statement is given below.

```
[DATA DIVISION]
     01 GRP1.
       02 A PIC X(8).
       02 B PIC S9(4) PACKED-DECIMAL.
       02 C PIC N(4).
   [PROCEDURE DIVISION]
       INITIALIZE GRP1.
          ...1
       INITIALIZE GRP1 REPLACING NUMERIC DATA BY 1
      ...2
```

1. Initializes the elementary item subordinate to GRP1. A space is set in A, a zero in B, and a National space in C.
2. Initializes only the numeric item of the elementary items subordinate to GRP1. A 1 is set in B. Nothing is set in A or C.

Character-string Handling

Use the INSPECT statement to count the occurrence number of a character-string or replace a character-string. Use the STRING statement to link character-strings, and use the UNSTRING statement to parse a character-string.

Example of the INSPECT Statement

An example of the INSPECT statement is given below.

```
[DATA DIVISION]
   77 A PIC X(10) VALUE “AB*DE*FGH”.
   77 B PIC 99.
[PROCEDURE DIVISION]
      INSPECT A TALLYING B ALL “*”.                    ...1
```

1. Inspects the contents of A, and counts the occurrence number of “*”. When the value of B prior to execution of the INSPECT statement is zero, this value becomes 3 after the INSPECT statement is executed.

Example of the STRING Statement

An example of the STRING statement is given below.

```
[DATA DIVISION]
   77 A PIC X(5) VALUE “ABCDE”.
   77 B PIC X(3) VALUE “123”.
   77 C PIC X(2) VALUE “+-”.
   77 D PIC X(10).
[PROCEDURE DIVISION]
      STRING  A DELIMITED BY SIZE
         B DELIMITED BY SIZE
         C DELIMITED BY SIZE
         INTO D.                                       ...1
```

1. Links the contents of A, B, and C in that order, and stores them in D. The contents of D after the STRING statement execution are “ABCDE123+-”.

Example of the UNSTRING Statement

An example of the UNSTRING statement is given below.

```
[DATA DIVISION]
   77 A PIC X(5)
   77 B PIC X(3)
   77 C PIC X(2)
   77 D PIC X(10) VALUE “ABCDE123+-”.
[PROCEDURE DIVISION]
      UNSTRING D INTO A B C.                    ...1
```

1. The contents of D are delimited in A, B, and C according to their sizes and stored in the delimited order. After the UNSTRING statement is executed, the contents “ABCDE”, “123” and “+-” are stored in each A, B, and C.

Simple Input-Output

Use the ACCEPT statement to input a small amount of data from a hardware device, and use the DISPLAY statement to output a small amount of data on a hardware device. A file need not be defined when using the ACCEPT statement or DISPLAY statement. The ACCEPT statement can also be used to give the date and time.

An example of the ACCEPT statement and DISPLAY statement is given below.

```
 DISPLAY “ERROR”.                        ...1
  ACCEPT X.                          ...2
  ACCEPT C-TIME FROM TIME.               ...3
```

1. Indicates a nonnumeric literal (ERROR).

2. Inputs the contents of X.

3. Stores the current time in C-TIME.

Terminating a Program

Use the STOP RUN statement to terminate a program. This statement indicates termination of the run unit.

Pointer Handling

Use pointers for referencing the address area.

The address of an area can be found using the ADDR function and set in the pointer data item. A pointer data item defines the data description entry written in the USAGE IS POINTER clause.

You define the structure attributes of the area to be referenced using a pointer in the BASED-STORAGE SECTION of the DATA DIVISION. The data item area defined in the BASED-STORAGE SECTION cannot be reserved.

Clauses can be described both in the data description entry of the WORKING-STORAGE SECTION and also the BASED ON clause written in the data description entry of the BASED-STORAGE SECTION. Write the BASED ON clause to specify an implicit pointer.

When referencing a data item defined in the BASED-STORAGE SECTION, a pointer qualifier or an implicit pointer must be added to the data-name.

An example of handling a pointer is given below.

```
[DATA DIVISION]
   BASED-STORAGE SECTION.
   77 AA PIC X  BASED ON P1.                 …Specifies implicit pointer P1.
   77 BB PIC 9(4) BINARY.
   WORKING-STORAGE SECTION.
   77 CC PIC X(100).
   77 P3 USAGE IS POINTER.                   ... Defines pointer data item P3.
   LINKAGE SECTION.
   01 P1 USAGE IS POINTER.                   ... Defines pointer data item P1.
   01 P2 USAGE IS POINTER.                   ... Defines pointer data item P2.
[PROCEDURE DIVISION]
   PROCEDURE DIVISION USING P1
   P2.
      IF AA = “A” THEN                       ... Adds a pointer implicitly to AA and references it.
        MOVE ZERO TO P2->BB                  ... Adds a pointer explicitly to BB and references it.
         :
      MOVE FUNCTION ADDR(CC) TO P3           ... Uses the ADDR function to find the CC address.
      CALL “SUB” USING P3.
         :
```

Handling a Floating Point

A floating point is numeric data expressed in the format “mantissa*(10**exponent).” There are two types of floating points:

- Floating-point item
- Floating-point literal

A single-precision floating-point literal can be 0 or an absolute value number greater than approximately 1.18 x 10 to the -38th power and less than about 3.4 x 10 to the +38th power. A double-precision floating-point literal can be 0 or an absolute value number greater than approximately 2.23 x 10 to the -308th power and less than about 1.79 x 10 to the +308th power.

A floating-point item can be written where “data-name” or “identifier” is indicated below. A floating-point literal can be written where “literal” is indicated below.

ENVIRONMENT DIVISION

- Literal in a SYMBOLIC CONSTANT clause

DATA DIVISION

- Literal in a VALUE clause. However, a data item specified in a VALUE clause must be an internal floating-point data item.
- Data-name in a REDEFINES clause or a RENAMES clause

PROCEDURE DIVISION

- Identifier or literal in a conditional expression. The identifier or literal can be written in a relation condition only.
- Identifiers or literal in an ADD statement
- Identifiers or literal in a COMPUTE statement
- Identifier or literal in a DISPLAY statement

  A floating-point literal or external floating-point item written in the program is displayed without being converted.

  An internal floating-point data item is converted to an external floating-point data item having the format shown below before it is displayed:

  Single-precision: -.9(8)E-99

  Long precision: -.9(17)E-99
- Identifiers or literal in a DIVIDE statement not including a REMAINDER phrase
- Identifiers or literal in an INITIALIZE statement

  When the floating-point item is specified for initialization by the INITIALIZE statement, it is initialized as follows:

  – When the floating-point item has been specified for initialization by the INITIALIZE statement not containing a REPLACING phrase, it is initialized to zero.

  – When NUMERIC has been written in the REPLACING phrase, the floating-point item is also initialized.

- Identifiers or literal in a MOVE statement

  At transcription, the floating-point item is treated in the same way as the numeric data item of a non-integer.

Note: The precision of an internal floating point differs from the precision of an external floating point and floating point. When transcribing data between two types, the ranges must be common.

- Identifiers or literal in a MULTIPLY statement
- Identifiers or literal in a SUBTRACT statement
- Data-name in the USING phrase of PROCEDURE DIVISION
- Data-name in the USING phrase of an ENTRY statement
- Identifiers or literal in the USING phrase of a CALL statement. The identifiers or literal cannot be written in a USING BY VALUE phrase.
- Identifiers or literal in the REPLACING phrase of a COPY statement
- Argument of a numeric function. The argument type is numeric.

Input-Output Facility

There are three types of input-output facilities:

- Sequential I-O module
- Relative I-O module
- Indexed I-O module

The sequential I-O module processes the records in a file according to a set sequence. A file processed by the sequential I-O module is called a “sequential file.” Each record in a sequential file is identified by its position at the time the record was written.

The relative I-O module processes any selected record in a file or processes the records in a file according to a set sequence. A file processed by the relative I-O module is called a “relative file.” Each record in a relative file is identified by its relative record number.

An indexed I-O module processes any selected record in a file or processes the records in a file according to a set sequence. A file processed by the indexed I-O module is called an “indexed file.” Each record in an indexed file is identified by the value of one or several keys.

Include the following entries as required in a program using the input-output facility.

Program Description	**Main Facility**
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates the file-name with an external medium. Specifies the organization, access mode, relative key, prime record key, alternate record key, and lock mode.
I-O CONTROL paragraph	Specifies the storage area to be shared by several files.
DATA DIVISION	
FILE SECTION	
File description entry	Defines the physical structure of a file.
Record description entry	Defines the structure of a record.
PROCEDURE DIVISION	
Declarative	
USE statement and USE AFTER STANDARD EXCEPTION	Defines the input-output error procedure.
Procedure	
OPEN	Opens a file.
READ	Reads a record from a file.
WRITE	Writes a record to a file.
REWRITE	Reloads a record in a file.
DELETE (*1)	Deletes a record in a file.
START (*1)	Positions a file logically.
UNLOCK	Releases a lock on a record.
CLOSE	Closes a file.

(*1) The DELETE and START statements are not included in the sequential file facility.

File Organization

The logical structure of a file on an external medium is called its "organization." There are four types of file organization as listed below. The ORGANIZATION clause in the file control entry specifies which type of file should be used.

- Sequential organization
- Line sequential organization
- Relative organization
- Indexed organization

A sequential organization file or line sequence organization file can be used with the sequential file facility. A relative organization file can be used with the relative file facility. An indexed organization file can be used with the indexed file facility.

Sequential Organization File

A sequential organization file consists of records identified according to the positions immediately before and immediately after each record. The logical position of a record is determined when the record is written and does not change except when a record is added at the end of the file.

To use a sequential organization file, write a SEQUENTIAL phrase in the ORGANIZATION clause or omit the ORGANIZATION clause.

There are two types of sequential organization files: record sequential file and print file. The record sequential file supports input-output handling on an external storage device. The print file writes data to a printing device. The ASSIGN clause, WRITE statement, and entries in the record description entry determine whether a record sequential file or print file should be used. Refer to the “COBOL85 User's Guide” for more information on the rules for determining which file to used.

The print file can edit and output records according to the form descriptor. When using the form descriptor, specify the data item which sets the form descriptor name in the file control entry FORMAT clause. A print file specifying a FORMAT clause is called a “print file with FORMAT clause.” A print file not specifying a FORMAT clause is called a “print file without FORMAT clause.”

Line Sequential Organization File

A line sequential organization file consists of records delimited by delimiters. One record is counted as one line, and one line consists of printable characters and record delimiters. The logical position of a record is determined when the record is written and does not change except when a record is added at the end of the file.

To use a line sequential file, write the LINE SEQUENTIAL phrase in the ORGANIZATION clause.

Relative Organization File

A relative organization file consists of records identified by the relative record number. The relative record number is an integer of 1 or higher. Each record in the relative file is stored in the position indicated by its relative record number when the record is written. For example, a record written with a relative record number of 10 is stored in the area whose relative record number is 10 even if no records are stored in the positions indicated by relative record numbers 1 to 9.

To use a relative organization file, write the RELATIVE phrase in the ORGANIZATION clause. To process records randomly, the relative key item must be specified in the RELATIVE KEY phrase of the ACCESS MODE clause.

Indexed Organization File

An indexed organization file consists of records and indexes for identifying records. An indexed organization file is identified by the value of the area (key) in the records.

To use an indexed organization file, write the INDEXED phrase in the ORGANIZATION clause. Specify the key in the RECORD KEY clause or the ALTERNATE RECORD KEY clause. Specify the prime record key in the RECORD KEY clause and the alternate record key in the ALTERNATE RECORD KEY clause. The prime record key must be specified.

The prime record key and alternate record key are generically called “record keys.” The record key value can be made unique within a file or can allow duplication of the value within a file. To allow duplication of the record key value, write the DUPLICATES phrase in the RECORD KEY clause or the ALTERNATE RECORD KEY clause.

A record key calling a record in an indexed file is called a “key of reference.” A key of reference changes upon execution of an input-output statement.

File Connector

A file connector associates a file on an external medium with a program. It is a storage area which holds information about files. It can be used for the following purposes:

- To link a file-name and file on an external medium.
- To link a file-name and the record area associated with a file.

Write a SELECT clause and ASSIGN clause in the file control entry to associate the file connector with a file-name and a file on an external medium. The file connector is established when the OPEN statement is executed.

Operation of Input-Output Statement

File processing is written by combining input-output statements (OPEN statement, READ statement, WRITE statement, REWRITE statement, START statement, DELETE statement, and CLOSE statement).

Before file processing can begin, the OPEN statement must be executed to open the file. After processing is completed, the CLOSE statement closes the file. The access mode and open mode determine which input-output statements can be executed between the OPEN statement and CLOSE statement.

Access Mode

The sequence in which records in a file are processed is called the “access mode.” There are three access modes: sequential access mode, random access mode, and dynamic access mode. In sequential access mode, reading and writing of records is done in a fixed sequence. In random access mode, reading and writing records is done in a random sequence. In dynamic access mode, processing switches between the sequential and random access modes as appropriate while input-output statements are being executed.

Specify the access mode in the ACCESS MODE clause of the file control entry. When the program consists of sequential files, only the sequential access mode can be specified.

Open Mode

The mode becoming active when a file is opened is called “open mode.” There are four open modes: input mode, output mode, input-output mode, and extension mode. Specify the mode to be activated when a file is opened in the OPEN statement.

Relationship Between Open Mode and Input-Output Statement for Sequential Files

The following table shows the relationship between the open mode and input-output statement for sequential files.

Input-Output	**Open Mode**			
Statement	**Input Mode**	**Output Mode**	**I-O Mode**	**Extension Mode**
READ	Possible		Possible	
WRITE		Possible		Possible
REWRITE			Possible	

Relationship Between Open Mode and Input-Output Statement for Relative Files and Indexed Files

The table below shows the relationship between the open mode and input-output statement for relative files and indexed files.

Access Mode	**Input-**	**Open Mode**			
	Output Statement	**Input Mode**	**Output Mode**	**I-O Mode**	**Extension Mode**
Sequential access mode	READ	Possible		Possible	
	WRITE		Possible		Possible
	REWRITE			Possible	
	START	Possible		Possible	
	DELETE			Possible	
Random access mode	READ	Possible		Possible	
	WRITE		Possible	Possible	
	REWRITE			Possible	
	START				
	DELETE			Possible	
Dynamic access mode	READ	Possible		Possible	
	WRITE		Possible	Possible	
	REWRITE			Possible	
	START	Possible		Possible	
	DELETE			Possible	

File Position Indicator

A file position indicator is provided in the concept of COBOL for determining the next record to be processed in a series of input-output operations. The file position indicator has meaning only in a file opened in input mode or input-output mode.

The file position indicator status for a sequential file is influenced by execution of a CLOSE statement, OPEN statement, or READ statement.

The file position indicator status for a relative file or indexed file is influenced by execution of a CLOSE statement, OPEN statement, READ statement, or START statement.

Volume Indicator

A volume indicator is provided in the concept of COBOL for making the current physical volume unique. The volume indicator exists only in a sequential file. The status of the volume indicator is influenced by execution of a CLOSE statement, OPEN statement, READ statement, or WRITE statement.

Sharing and Exclusion of Files

COBOL programs can accommodate an attribute enabling either several run units to use a physical file in a mass storage unit simultaneously (shared mode) or a single run unit to use the physical file exclusively (exclusive mode).

Specify shared mode or exclusive mode for use of each physical file in the LOCK MODE clause and OPEN statement of the file control entry. In the LOCK MODE clause, specify whether a physical file associated with a file name is to be opened in shared mode or exclusive mode in the OPEN statement for each separately compiled program. Files for which the shared mode has been specified in the LOCK MODE clause can be opened either in shared mode or exclusive mode. Files for which the exclusive mode has been specified in the LOCK MODE clause can be opened only in exclusive mode.

Executing the OPEN statement determines whether a physical file can be used in shared mode or exclusive mode.

A file opened in shared mode has a separate file indicator associated with it each time the file is opened in shared mode. This means a single physical file can have two or more file indicators associated with it.

A file opened in exclusive mode has only one file indicator associated with it. This file is then locked and cannot be opened by any other run unit. Executing a CLOSE statement for a file opened in exclusive mode releases the lock on the file and the file is then able to be opened by another run unit.

The table below indicates the mode (shared or exclusive) in which physical files having various specifications can be opened.

Specification in LOCK MODE clause	**OPEN statement**							
	WITH LOCK specified				**WITH LOCK not specified**			
	INPUT	**I-O**	**OUTPUT**	**EXTEND**	**INPUT**	**I-O**	**OUTPUT**	**EXTEND**
No specification	Exclusive mode				Shared mode	Exclusive mode		
AUTOMATIC					Shared mode		Exclusive mode	Shared mode
MANUAL[*1]					Shared mode		Exclusive mode	Shared mode
EXCLUSIVE					Exclusive mode			

*1: MANUAL cannot be specified for a sequential file.

Locking a Record

A record read by a READ statement and located in a file opened in shared mode or input-output mode can be locked. Locking a record prevents input-output statements of other file connectors from referencing the record.

If the record is in a sequential file, only one record in one physical file can be locked at one time. If the record is in a relative file or indexed file, any number of records in a single physical file can be locked. To lock only one record, specify AUTOMATIC in the LOCK MODE clause. To lock several records, specify MANUAL in the LOCK MODE clause.

Locking a Record Where AUTOMATIC is Specified in the LOCK MODE Clause

The file indicator of a file specifying AUTOMATIC in the LOCK MODE clause and opened in input-output mode can maintain the lock on a single record. At execution of a READ statement where WITH LOCK has been specified or where the WITH [NO] LOCK has not been specified, the record read is locked and the lock information in the file indicator is updated. A READ statement specifying WITH NO LOCK does not lock the record.

Executing a READ statement specifying WITH LOCK or for not specifying the WITH [NO] LOCK prevents any other file indicator from executing a READ, WRITE, REWRITE, or DELETE statement on the locked record. However, it does not prevent the file indicator of a file opened in extension mode from executing a WRITE statement.

The lock on the record is released when any of the following statements is executed for the same file indicator executing the original READ statement:

1. A READ, WRITE, REWRITE DELETE or START statement
2. An UNLOCK statement
3. A CLOSE statement executed either explicitly or implicitly

Locking a Record Where MANUAL is Specified in the LOCK MODE Clause

The file indicator of a file specifying MANUAL in the LOCK MODE clause and opened in input-output mode can maintain the lock on several records simultaneously. At each execution of a READ statement specifying WITH LOCK, the record read is locked and relevant lock information is added to the file indicator. A READ statement specifying WITH NO LOCK or specifying WITH [NO] LOCK parameter does not lock a record.

Executing a READ statement specifying WITH LOCK prevents any other file indicator from executing a READ, WRITE, REWRITE, or DELETE statement on the set of locked records. However, it does not prevent execution of an input-output statement on a record not locked.

The lock on the records is released when any of the following statements is executed for the same file indicator executing the original READ statement:

1. An UNLOCK statement
2. A CLOSE statement executed either explicitly or implicitly. The lock on several records cannot be released for individual records.

I-O Status

The I-O status is a double character area provided in the concept of COBOL for indicating the result of an input-output statement. It is set during execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, or WRITE statement.

More detailed information on the I-O status is given for a print file specifying the FORMAT clause in the sequential file function.

The I-O status and more detailed information are set each time input or output is executed before execution of an unconditional statement written in the input-output statement or before execution of a USE AFTER STANDARD EXCEPTION procedure relating to the file.

The I-O status and more detailed information can be referenced by writing a FILE STATUS clause in the file control entry.

The first digit position of the I-O status indicates the classification of the I-O status. The table below shows the classifications.

I-O status	Classification	Meaning
0x	Succeed	The input-output statement has been executed successfully.
1x	At end condition	Execution of the sequential access READ statement failed due to the at end condition.
2x *1	Invalid key condition	Execution of the input-output statement failed due to the invalid key condition.
3x	Permanent error condition	Execution of the input-output statement failed due to an error which prevented continuation of file processing.
4x	Logical error condition	Execution of the input-output statement failed due to a sequence error in execution or because the user went beyond the set limit.
9x	Other error	Execution of the input-output statement failed due to an error not mentioned above.

*1: An invalid key condition does not occur in a sequential file.

At End Condition

The at end condition may occur at execution of a sequential access READ statement. When this condition occurs, execution of the READ statement fails. The file is not affected.

Write the AT END phrase in the READ statement to enable the program to detect whether an at end condition has occurred.

Invalid Key Condition

An invalid key condition may occur at execution of a DELETE, random access READ, REWRITE, START, or WRITE statement for a relative file or indexed file. When this condition occurs, execution of the input-statement fails. The file is not affected.

Write the INVALID KEY phrase in the input-output statement to enable the program to detect whether an invalid key condition has occurred.

File Attribute Conflict Condition

A file attribute conflict condition may occur at execution of an OPEN statement. When this condition occurs, execution of the OPEN statement fails. The file is not affected.

Record Format

There are two types of file records: fixed-length records and variable-length records. The program description determines the record format regardless of the physical record transcription on the medium and information added to the record.

All fixed-length records in a file have the same number of character positions. Variable-length records in a file each have a different number of character positions.

The description in the RECORD clause and record description entry determines the record format. It also determines the record length written by the WRITE statement and the record length read by the READ statement.

The following table shows the record format, maximum record length, and minimum record length for various descriptions.

Description in RECORD clause	**Record format**	**Minimum record length**	**Maximum record length**
Format 1 RECORD integer-1	Fixed-length	Value of integer-1	Value of integer-1
Format 2 RECORD VARYING [FROM integer-2 [TO integer-3]] [DEPENDING ON data-name-1]	Variable-length	Minimum value of length of record description entry	Maximum value of length of record description entry or integer-3, whichever is greater
Format 3 RECORD integer-4 TO integer-5 or RECORD clause is not specified	Variable-length in the following cases: In all other cases, the record format is fixed-length. - Two or more record description entries of different lengths have been written. - An OCCURS DEPENDING ON clause has been written in the record description entry. - A CHARACTER TYPE clause has been written in the record description entry.	Minimum value of length of record description entry. (*1)	Maximum value of length of record description entry. (*1)

*1: The minimum value and maximum value of the length of the record description entry are the same for a fixed-length record.

Record Area

The storage area allocated for a record description entry in a FILE SECTION is called a “record area.” Several record description entries can be associated with a single file but the record area is allocated to the file, not to the record description entries. The record area size is the same as the maximum record length of the record associated with the file.

A record area can be shared by several files in a separately compiled program. To enable the record area to be shared, specify files to share the record area in the SAME RECORD AREA clause of the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION.

Special Register

Writing a LINAGE clause in the file description entry of a sequential file creates the LINAGE-COUNTER special register. LINAGE-COUNTER is treated as an integer item the same size as integer-1 or data-name-1 in the LINAGE clause and does not have symbols.

LINAGE-COUNTER counts the number of lines held and printed by the I-O-control system. Only statements in the PROCEDURE DIVISION can reference the value of LINAGE-COUNTER. The user cannot set the value of LINAGE-COUNTER.

Inter-program Communication Module

An inter-program communication module mutually links programs. Executing a CALL statement calls a program in another separately compiled program or another program in the same separately compiled program. The program executing the CALL statement is called the “calling program.” The program to which control is passed by execution of the CALL statement is called the “called program.”

The inter-program communication module not only passes control back and forth between the calling program and the called program but also passes parameters and shares data and files.

Use the following descriptions as required in a program which uses the inter-program communication module.

Program Description	**Main Facility**
IDENTIFICATION DIVISION	Specifies the name of the program and sets the initial attribute and common attribute in the program
DATA DIVISION	
FILE SECTION	
File description entry	Sets the external attribute in a file connector. Sets the global attribute in the file-name and data-name
Record description entry	Sets the external attribute in a data record. Sets the global attribute in a data-name
LINKAGE SECTION	
77-level description entry and record description entry	Defines the parameters to be accepted
WORKING STORAGE SECTION	
Record description entry	Sets the external attribute in a data record. Sets the global attribute in a data-name
BASED-STORAGE SECTION and CONSTANT SECTION	
Record description entry	Sets a global name in a data-name
Procedure name	
HEADER OF PROCEDURE DIVISION	Specifies the parameters to be accepted
Procedure portion	
CALL	Calls another program
CANCEL	Initializes another program
ENTRY	Specifies the secondary entry name. Specifies the parameters to be accepted
EXIT PROGRAM	Resumes the called program
End program header	Specifies the end of a program

Accessing and Returning to a Program

Execute a CALL statement to access a program. To return to the calling program from the called program, execute an EXIT PROGRAM statement.

Run Unit

A group of object programs mutually linked and operated at execution is called a “run unit.” A run unit consists of a single separately compiled program or a group of several separately compiled programs.

Main Program and Sub-program

The COBOL program first activated in a separately compiled program is called the “main program.” Another COBOL program called from the main COBOL program is called a “sub-program.” An internal program is a sub-program regardless of whether the outermost program in the separately compiled program is the main program or a sub-program.

Starting Execution of a Called Program

Execution of a CALL statement makes the called program ready for execution. Then, execution of the called program begins at one of the following statements depending on the description in the CALL statement.

1. When a program-name or program-name literal has been specified in the CALL statement, execution starts at the first statement in the procedure portion of the PROCEDURE DIVISION.

2. When a secondary entry name has been specified in the CALL statement, execution starts at the first statement following the ENTRY statement. “Secondary entry name” refers to the literal written in the ENTRY statement. An ENTRY statement can be written only in the outermost program of a separately compiled program.

Executing and Ending a Called Program

To end a called program, execute an EXIT PROGRAM statement, or STOP RUN statement. These statements are generically called “program end statements.”

To end a program and return to the called program, execute an EXIT PROGRAM statement. To end a program and return to the operating system, execute a STOP RUN statement.

Global Name and Local Name

Either the global attribute or the local attribute can be set in a file-name (excluding the file-name of a sort-merge file), record-name, data-name (excluding the data-name of a screen item), and condition-name.

A name having the global attribute is called a “global name,” and a name having the local attribute is called a “local name.” A global name can be referenced in a program with a defined name and all programs directly or indirectly contained in the defined program. A local name can be referenced only in a program with a defined name.

Writing a GLOBAL clause in a file description entry makes the following names global names:

- A file-name written in the file description entry
- All data-names, record-names, and condition-names in a record description entry associated with the file description entry

Writing a GLOBAL clause in a data description entry of level-number 01 makes the following names global names:

- A data-name (record-name) written in the data description entry whose level-number is 01
- All data-names and condition-names subordinate to the data description entry whose level-number is 01

File-names, record-names, data-names, and condition-names not defined as global names become local names.

External Attribute and Internal Attribute

Either the external attribute or the internal attribute can be set in a data item or file connector (excluding the file connector of a sort-merge file). Set the external attribute to enable several programs in a run unit to share an area relating to a data item or file connector.

Writing an EXTERNAL clause in a file description entry sets the external attribute in the file connector corresponding to the file-name written in the file description entry. A file connector having the external attribute is called an “external file connector.”

Writing an EXTERNAL clause in the data description entry of level-number 01 sets the external attribute in the data item defined in the data description entry and all data items subordinate to the data description entry. A record configured in a data item having the external attribute is called an “external data record.”

One storage area is allocated to all external file connectors having the same file-name in a run unit. One storage area is also allocated to all external data records having the same record-name in a run unit. External file connectors and external data records can be referenced from any program in the run unit. When referencing an external file connector or external data record in an internal program, the name referenced must be defined as a global name.

Data items and file connectors in which the external attribute has not been set are given the internal attribute. Data items and file connectors having the internal attribute can be referenced only in one separately compiled program. A file connector which does not have the external attribute is called an “internal file connector.”

The external attribute cannot be set in the following items. These items are always given the internal attribute.

- A data item defined in a BASED-STORAGE SECTION, CONSTANT SECTION, LINKAGE SECTION, or REPORT SECTION
- A data item defined in the record description entry of a sort-merge file
- A screen item

The compiler does not set the external attribute in an index-name even if an EXTERNAL clause has been written in a record description entry including a data description entry where the INDEXED BY phrase has been written.

Common Attribute in a Program

The common attribute can be set in an internal program. An internal program having the common attribute can be called not only from a program which directly or indirectly contains the internal program but also from a program which is directly or indirectly contained in a program directly containing the internal program. An internal program not having the common attribute can be called only from a program which directly or indirectly contains the internal program.

To set the common attribute in an internal program, write a COMMON phrase in the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION. An internal program having the common attribute is called a "common program."

Initial State of Program

When a program is first called in a run unit, the program is initialized to the following state. The state of the program at this stage is called the "initial state of program."

1. A data item defined in the WORKING-STORAGE SECTION is set to its initial state. A data item specifying a VALUE clause and any data item subordinate to such a data item is initialized to the value in the VALUE clause. The initial value for a data item outside the scope of the VALUE clause is not defined.

2. An internal file connector is set to a state other than to the open state.

3. The controller of the PERFORM statement is set to the initial state.

4. A GO TO statement referenced in the ALTER statement is set to the initial state.

To set the program in its initial state the second and subsequent times it is called in the run unit, use one of the following methods:

- Set the initial attribute in the program.
- Execute a CANCEL statement.

Initial Attribute in a Program

The initial attribute can be set in a program. A program having the initial attribute is always set in its initial state when called.

To set the initial attribute in a program, write an INITIAL clause in the program-name paragraph of the IDENTIFICATION DIVISION. Writing an INITIAL clause sets the initial attribute in a program where the INITIAL clause has been written and in all programs contained directly or indirectly in the program. Both the initial attribute and common attribute can be set in an internal program.

A program having the initial attribute is called an “initial program.” An initial program is always set in its initial state whenever the program or the program's secondary entry point is called by a CALL statement.

Initialization by CANCEL Statement

A CANCEL statement can set the state of a program when it is next called in its initial state. Executing the CANCEL statement sets the program specified in the CANCEL statement and any programs contained directly or indirectly in the program to the initial state the next time it is called.

Passing a Parameter

To pass a parameter between a calling program and a called program, specify the parameter as follows:

1. Specify the parameter to be passed in the USING phrase of the CALL statement in the calling program.

2. Specify the parameter to be accepted in the PROCEDURE DIVISION header of the called program or in the USING phrase of the ENTRY statement. Define the data item of the parameter to be accepted in the LINKAGE SECTION. Parameters are associated between the calling program and the called program in the sequence they were written in the USING phrase. The names of corresponding parameters need not be the same.

Specify whether the value of a parameter changed in a called program is to be returned in the USING phrase of the CALL statement as follows:

1. To return the value of a parameter changed in the called program to the accessing program, specify the parameter in the USING BY REFERENCE phrase or in a USING phrase in which a BY phrase has not been specified.

2. When the value of a parameter changed in the called program is not to be returned to the accessing program, specify the parameter in the USING BY CONTENT phrase or the USING BY VALUE phrase. Use the USING BY VALUE phrase when accessing a program written in C language or a language enabling parameter passing by value.

Scope of Names

A user-defined word determines the scope of names affixed to the user-defined words.

User-defined Words Referenced in the Defined Program Only

The following user-defined words can be referenced only in the defined program. A name can be defined in more than one program in a separately compiled program but can be referenced only in the defined programs.

- Paragraph-name
- Section-name
- Report-name
- File-name of a sort-merge file
- Data-name of a screen item

User-defined Words Referenced in All Programs in a Separately Compiled Program

The following user-defined words can be referenced in all programs in a separately compiled program. However, an entity to be referenced must be associated with the separately compiled program at compilation.

- Library-name
- Text-name

User-defined Words Referenced in the Defined Program and Programs Contained in the Defined Program

The following user-defined words can be referenced in the defined program and in any programs contained directly or indirectly in the program:

- Alphabet-name
- Class-name
- Condition-name (condition-name defined in the ENVIRONMENT DIVISION)
- Mnemonic-name
- Symbolic-character
- Symbolic constant
- Positioning unit-name
- Print mode name

User-defined Words Where the Scope of the Name Changes Depending on the Global Attribute

The scope of the name for the following user-defined words changes depending on whether the name has the global attribute:

- Condition-name (condition-name defined in DATA DIVISION)
- Data-name (excluding a data-name of a screen item)
- Record-name
- File-name (excluding the file-name of a sort-merge file)

When the global attribute has been set in these user-defined words, the user-defined words can be referenced in the defined program and in any programs contained directly or indirectly in the program. When the global attribute has not been set, the user-defined words can be referenced only in the defined program.

In a program contained directly or indirectly in a program in which a user-defined word having the global attribute has been defined, a user-defined word which has the same name as the user-defined word having the global attribute can be defined and then several items can be referenced by the same name.

Consider the case of a program (program A) directly containing another program (program B). If a user-defined word (user-defined word X) is defined in two or more programs and then user-defined word X is referenced in program B, the following rules are applied in the order shown to identify the items to be referenced:

1. When user-defined words X is defined in program B, the system references the area of user-defined words X in program B.

2. When user-defined words X is not defined in program B, the system searches the programs starting from program A and moving outward until it finds the program where user-defined words X has been defined. When it first finds the program where user-defined words X has been defined, it references the area of user-defined words X in that program.

Scope of Name of Index-name

When an index-name has been associated with a table and the data-name affixed in the table has the global attribute, the index-name is also given the global attribute. The scope of the name of an index-name is the same as the scope of the name of a data-name affixed to a corresponding table. See the above item “User-defined words for the scope of the name changes depending on the global attribute.”

Scope of the Name of a Program-name and Secondary Entry Name

The program-name and secondary entry name of a program making up a single separately compiled program must not be the same.

When the program makes up a single run unit in two or more separately compiled programs, the program-name and secondary entry name of the outermost program in each separately compiled program must not be the same. However, an internal program can be given the same program-name or secondary entry name as a program making up a different separately compiled program.

The program-name and secondary entry name can be referenced in the CALL statement or CANCEL statement in a different program. Program-names and secondary entry names referenced in the CALL statement or CANCEL statement are as follows:

1. The program-name of an internal program not having the common attribute can be referenced only from a program containing the internal program directly or indirectly.
2. The program-name of an internal program having the common attribute can be referenced from the following programs:

   A program containing the internal program directly or indirectly

   A program contained either directly or indirectly in the program that directly contains the internal program

   Note that an internal program containing the common attribute cannot be referenced from itself or from a program contained in itself.
3. The program-name and secondary entry name of the outermost program in a separately compiled program can be referenced from any program in the run unit. However, they cannot be referenced from a program contained directly or indirectly in the outermost program.

Special Register

The special register PROGRAM-STATUS (RETURN-CODE) is automatically created for a program. PROGRAM-STATUS and RETURN-CODE are synonymous. PROGRAM-STATUS passes a return code to the operating system or the called program. PROGRAM-STATUS is treated as a numeric data item defined as “PICTURE S9(9) COMPUTATIONAL-5.”

The value of PROGRAM-STATUS when the program starts running is zero. After the return code has been set in PROGRAM-STATUS, the program end statement is executed and the return code is passed as follows:

1. When control has returned to the operating system, the return code is reported to the operating system.
2. When control has returned to the called program, the return code is stored in PROGRAM-STATUS in the called program.

Sort-Merge Module

The sort-merge module consists of the sort module and merge module.

The sort module arranges several records in a file according to a specific key. The SORT statement arranges the records in sequence.

The merge module merges files which have records arranged according to a specific key. The MERGE statement merges files.

A sort-merge file must be defined before the sort-merge module can be used. The sort-merge file is created internally. The user need not associate the sort-merge file with an external medium.

Use the following descriptions as required in a program which uses the sort-merge module.

Program Description	**Main Facility**
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates a sort-merge file with an external medium.
I-O CONTROL paragraph	Specifies the storage area shared by several files containing a sort-merge file.
DATA DIVISION	
FILE SECTION	
Sort-merge file description entry	Defines the physical structure of the sort-merge file.
Record description entry	Defines the structure of a record in the sort-merge file.
PROCEDURE DIVISION	
Procedure portion	
SORT	Arranges the record in sequence.
MERGE	Merges two or more files.
Input procedure	
RELEASE	Delivers a record to the first step in a sort operation.
Output procedure	
RETURN	Accepts a record from the last step in a sort operation. Accepts a record merged by the merge operation.

Sorting Methods

There are four sorting methods:

1. A method which directly sorts the records in a file and directly writes the result to another file. To use this method, write the USING phrase and GIVING phrase in the SORT statement.
2. A method which performs special processing on records before they are sorted, then sorts the records and directly writes the result to another file. To use this method, write the INPUT PROCEDURE phrase and GIVING phrase in the SORT statement.
3. A method which directly sorts records in a file and then performs special processing on the records. To use this method, write the USING phrase and OUTPUT PROCEDURE phrase in the SORT statement.
4. A method which performs special processing on records before they are sorted, then sorts the records and performs special processing on the sorted records. To use this method, write the INPUT PROCEDURE phrase and OUTPUT PROCEDURE phrase in the SORT statement.

Merging Methods

The records in a file must be arranged according to the key used at merging before the file can be merged. There are two merging methods:

1. A method which directly merges records in a file and directly writes the result to another file. To use this method, write the USING phrase and GIVING phrase in the MERGE statement.
2. A method which directly merges records in a file and then performs special processing on the merged records. To use this method, write the USING phrase and OUTPUT PROCEDURE phrase in the MERGE statement.

Input Procedure

When writing a SORT statement containing the INPUT PROCEDURE phrase, an input procedure must be written. Write the input procedure in the procedure portion of the PROCEDURE DIVISION. The input procedure repeats the following processes:

1. Performs special processing on records before they are sorted. For example, it edits and selects the records.
2. Executes the RELEASE statement for these records. Executing the RELEASE statement causes the records to be written to the sort-merge file.

After the input procedure has ended, the records written to the sort-merge file are sorted by a series of RELEASE statements.

Output Procedure

When writing a SORT statement containing the OUTPUT PROCEDURE phrase or a MERGE statement containing the OUTPUT PROCEDURE phrase, an output procedure must be written. Write the output procedure in the procedure portion of the PROCEDURE DIVISION. The output procedure is executed after sorting or merging of sort-merge files. The output procedure repeats the following processes:

1. Executes the RETURN statement. Records are read from the sort-merge file upon execution of this statement.
2. Performs special processing on the records read from the sort-merge file. For example, it edits and selects the records.

Sort-merge File

A sort-merge file is created and allocated internally for sorting and merging operations.

Unlike files used with the file module, the sort-merge file does not use the label procedure, blocking, buffer area, and reel concepts.

A sort-merge file can be referenced only in a MERGE, RELEASE, RETURN, or SORT statement.

Special Register

The special register SORT-STATUS is automatically created for the sort-merge module. SORT-STATUS is treated as a numeric data item defined as “PICTURE S9(4) COMPUTATIONAL-5.” It can be used to reference the return code for a sort operation or merge operation or to suspend a sort operation or merge operation.

The SORT-STATUS initial value is zero. The value is automatically initialized at the start execution of the SORT statement or MERGE statement.

Value of Return Code

At the end of execution of the SORT statement or MERGE statement, a return code is set in SORT-STATUS. The value of the return code is one of the following values:

0: Indicates the sort operation or merge operation ended normally.

16: Indicates the sort operation or merge operation did not end normally.

When the program does not contain a SORT-STATUS and the return code is a value other than zero, an error message is output to the system logic console and the program ends abnormally.

Suspending a Sort Operation or Merge Operation

When a program contains a SORT statement containing the INPUT PROCEDURE phrase, a SORT statement containing the OUTPUT PROCEDURE phrase, or a MERGE statement containing the OUTPUT PROCEDURE phrase, a sort operation or merge operation can be suspended by setting a value in SORT-STATUS.

Setting 16 in SORT-STATUS and executing a RELEASE statement at the input procedure shifts control to the end of the SORT statement.

Setting 16 in SORT-STATUS and executing a RETURN statement at the output procedure shifts control to the end of the SORT statement or MERGE statement.

Source Text Manipulation Module

The source text manipulation module fetches and replaces portions of a program to complete the program at compilation.

It contains a COPY statement and REPLACE statement. These statements are processed before any other statements at compilation. They do not carry meaning at execution.

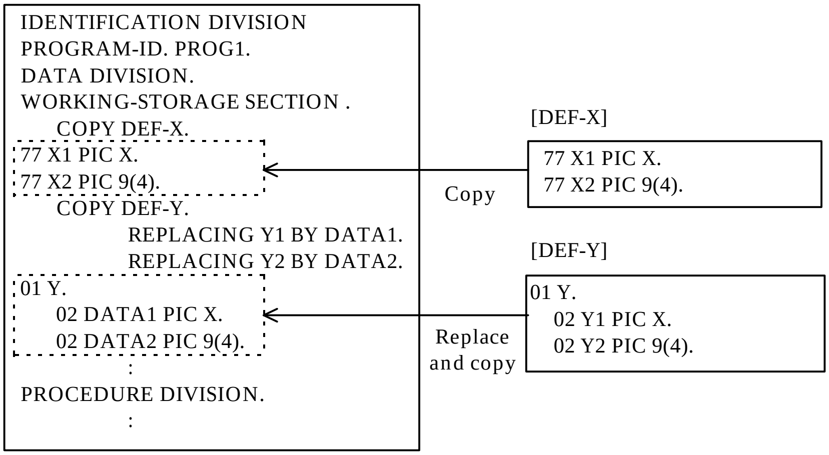
Use the COPY statement to copy a portion of a program from the COBOL library. The COBOL library is created separately from the program and is associated with the program at compilation.

Use the REPLACE statement to replace a portion of a program.

The COPY statement and REPLACE statement can be written in any position in a program or the COBOL library.

Example of COPY Statement

An example of the COPY statement is given below.



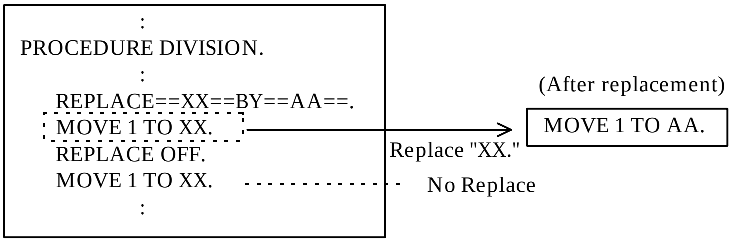
```
IDENTIFICATION DIVISION
PROGRAM-ID. PROG1.
DATA DIVISION.
WORKING-STORAGE SECTION .
    COPY DEF-X.
77 X1 PIC X.
77 X2 PIC 9(4).
    COPY DEF-Y.
           REPLACING Y1 BY DATA1.
           REPLACING Y2 BY DATA2.
01 Y.
    02 DATA1 PIC X.
    02 DATA2 PIC 9(4).
           :
PROCEDURE DIVISION.
           :
```

```
[DEF-X]
77 X1 PIC X.
77 X2 PIC 9(4).
```

```
[DEF-Y]
01 Y.
  02 Y1 PIC X.
  02 Y2 PIC 9(4).
```

Example of REPLACE Statement

Use the REPLACE statement in pairs: write the REPLACE statement at the start of the area to be replaced and write the REPLACE OFF statement at the end of the area. An example of the REPLACE statement follows.



```
              :
PROCEDURE DIVISION.
              :
    REPLACE==XX==BY==AA==.
    MOVE 1 TO XX.
    REPLACE OFF.
    MOVE 1 TO XX.
              :
```

```
(After replacement)
MOVE 1 TO AA.
```

Presentation File Module

A presentation file module inputs and outputs data and sends and receives messages via the presentation service control system or the message control system. It has the following functions:

- Handles data on a display using the screen descriptor.
- Outputs documents to a printer device using the form descriptor.
- Receives and passes messages between a unit and a program.

A screen form descriptor is data layout information used by a display or printer. It is created separately from the program.

The presentation file module uses a presentation file to input and output data and to send and receive messages. It forms the interface between a unit and a program. Executing an input-output statement for the presentation file causes the file to input or output data or to send or receive a message.

Use the following descriptions as required in a program which uses the presentation file module.

Program Description	**Main Facility**
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates the file-name of the presentation file with an external medium. Specifies the physical attribute for each destination type.
I-O CONTROL paragraph	Specifies processing methods peculiar to a presentation file, such as handling data items which extend over several conversations.
DATA DIVISION	
FILE SECTION	
File description entry	Defines the physical structure of the presentation file.
Record description entry	Defines the structure of a record in the presentation file.
PROCEDURE DIVISION	
Declarative	
USE statement and USE AFTER STANDARD EXCEPTION	Defines the input-output error procedure.
Procedure portion	
OPEN	Opens a file.
READ	Reads a record from a file.
WRITE	Writes a record to a file.
CLOSE	Closes a file.

Destination Type

Executing an input-output statement for the presentation file causes the file to input and output data and to send and receive messages. The destination type specifies which unit the input-output statement is to be executed for.

There are the following destination types:

- Display
- Printer
- Other logical units

Specify the destination type in the file control entry SYMBOLIC DESTINATION clause . The destination type must be set before a CLOSE, OPEN, READ, or WRITE statement can be executed.

Screen Form Descriptor

The screen layout on the display and the document layout printed at the printer can be defined separately from the program as a screen form descriptor. To use a screen form descriptor, specify the data item which sets the screen form descriptor name in the FORMAT clause of the file control entry.

The screen form descriptor edits records when the program is run. Executing a WRITE statement causes the records specified in the WRITE statement to be edited in accordance with the screen form descriptor and to be output to a unit. Executing a READ statement causes data input from a unit to be edited in accordance with the screen form descriptor and to be reported to the program.

The data entry for data passed between a unit and the program can be copied from the COBOL library created from the screen form descriptor.

File Organization and Access Mode

This section explains the file organization and access mode for a presentation file.

A presentation file has sequential organization. Write a SEQUENTIAL phrase in the ORGANIZATION clause of the file control entry or omit the ORGANIZATION clause.

A presentation file can be accessed only in the sequential access mode. Write a SEQUENTIAL phrase in the file control entry ACCESS MODE clause or omit the ACCESS MODE clause.

Operating Input-Output Statements

Write presentation file processing by combining input-output statements (OPEN, READ, WRITE, and CLOSE statements).

An OPEN statement must be executed first to open the presentation file before any processing can be performed. After processing, execute a CLOSE statement to close the file. The executable input-output statements between the OPEN and CLOSE statements depend on the open mode.

The following table shows the relationship between the open mode and the input-output statements for the presentation file.

Input-Output	**Open Mode**		
Statement	**Input Mode**	**Output Mode**	**I-O Mode**
READ	Possible		Possible
WRITE		Possible	Possible

I-O Status

The I-O status is a double character area provided in the concept of COBOL for indicating the result of an input-output statement. It is set during execution of a CLOSE, OPEN, READ, or WRITE statement.

More detailed information is also given in addition to the I-O status.

The I-O status and more detailed information are set each time input or output is executed before execution of an unconditional statement written in the input-output statement or before execution of a USE AFTER STANDARD EXCEPTION procedure related with the file.

The I-O status and more detailed information can be referenced by writing a FILE STATUS clause in the file control entry.

The first digit position of the I-O status indicates the classification of the I-O status.

The following table shows the classifications.

I-O status	Classification	Meaning
0x	**Succeed**	**The input-output statement has been executed successfully.**
1x	**At end condition**	**Execution of the sequential access READ statement failed due to the at end condition.**
3x	**Permanent error condition**	**Execution of the input-output statement failed due to an error which prevented continuation of file processing.**
4x	**Logical error condition**	**Execution of the input-output statement failed due to a sequence error in execution.**
9x	**Other error**	**Execution of the input-output statement failed due to an error not mentioned above.**

At End Condition

The “at end” condition may occur at execution of a READ statement. When this condition occurs, execution of the READ statement fails. The file is not affected.

Write the AT END phrase in the READ statement to enable the program to detect whether an “at end” condition has occurred.

Special Register

When creating a screen form descriptor, define not only the screen layout or document but also the data item to be used by both the program and the unit. If required, also specify the addition of an item controller in the data item at this time. An item controller is an area for passing the detailed information relating to data item input or output between a unit and the program. The program can reference and set the contents of the item controller using the special register.

When a data item or data items having an item controller have been written in a record description entry associated with the presentation file, the following five special registers are created for each data item:

- EDIT-MODE
- EDIT-OPTION
- EDIT-COLOR
- EDIT-STATUS
- EDIT-CURSOR

These five special registers are treated as single character alphanumeric character items.

The special registers can be referenced only by a statement in the PROCEDURE DIVISION. They are associated with a data item having an item controller, so they must be qualified with a data item having an item controller before they can be referenced. See the section titled “Qualification,” for information on qualifying a special register.

The facilities of the special registers are explained below. Refer to “COBOL85 User's Guide” for information on the item controller and the value in each special register.

1. The EDIT-MODE special register reports the processing mode of the data item to the system at execution of a WRITE statement. It can specify whether a data item in a record specified in the WRITE statement is to be processed and whether a National data item is to be displayed in the National or as an alphanumeric character.

2. The EDIT-OPTION special register reports the processing options of the data item to the system at execution of a WRITE statement. It can specify blinking, highlighting, reverse video, and underlining for a data item in a record specified in the WRITE statement.

3. The EDIT-COLOR special register reports the display color and brightness of the data item to the system at execution of a WRITE statement.

4. The EDIT-STATUS special register reports the input mode of the data item to the system at execution of a READ statement. It can specify whether a data item in a record specified in the READ statement is to be processed. If the READ statement is executed normally, the I-O status of the data item (indicating normal input or abnormal input) is set to EDIT-STATUS.

5. At execution of a READ statement, the EDIT-CURSOR special register reports positioning of the cursor for the data item to the system.

Built-in Function Module

The built-in function module performs specific processing based on the values of several arguments and returns the result to the program. The value of the function is called the “function value.”

The built-in function module can be used in a statement in the PROCEDURE DIVISION. To use the built-in function module, write a function-identifier in the statement.

The following table lists the built-in functions.

Function name	**Function value**
ACOS	Reverse cosine of argument
ADDR	Leading address of argument
ANNUITY	Equal payment amount for each quarter
ASIN	Reverse sine of argument
ATAN	Reverse tangent of argument
CAST-ALPHANUMERIC	National data item or National edited data item converted to Alphanumeric data item
CHAR	A single character in the order of the argument according to the collating sequence
COS	Cosine of argument
CURRENT-DATE	Current date and time
DATE-OF-INTEGER	Integer format date converted to standard format date
DAY-OF-INTEGER	Integer format date converted to annual format date
FACTORIAL	Factorial of argument
INTEGER	Maximum integer not exceeding value of argument
INTEGER-OF-DATE	Standard format date converted to integer format date
INTEGER-OF-DAY	Annual format date converted to integer format date
INTEGER-PART	Integer part of argument
LENG	Size of argument (number of bytes)
LENGTH	Length of argument (number of character positions or national character positions)
LOG	Natural logarithm of argument
LOG10	Common logarithm of argument

Function name	**Function value**
LOWER-CASE	Uppercase alphabetic characters in argument replaced by lowercase alphabetic characters
MAX	Maximum value in list of argument
MEAN	Arithmetic mean of list of argument
MEDIAN	Central value of list of argument
MIDRANGE	Arithmetic mean of minimum value and maximum value in list of argument
MIN	Minimum value in list of argument
MOD	Integer value of argument 1 modeled on argument 2
NATIONAL	COBOL character set converted to National character set.
NUMVAL	Character-string in numeric literal format converted to a numeral
NUMVAL-C	Character-string in numeric literal format (including a currency sign or comma) converted to a numeral
ORD	The order of an argument according to the collating sequence
ORD-MAX	Position of argument having the maximum value
ORD-MIN	Position of argument having the minimum value
PRESENT-VALUE	Current value of each end of quarter
RANDOM	Pseudo-random number
RANGE	Difference between maximum value and minimum value in list of argument
REM	Remainder after dividing argument 1 by argument 2
REVERSE	Character-string of argument in reverse order
SIN	Sine of argument
SQRT	Square root of argument
STANDARD-DEVIATION	Standard deviation of list of argument
SUM	Sum of list of argument
TAN	Tangent of argument
UPPER-CASE	Lowercase alphabetic characters in argument replaced by uppercase alphabetic characters
VARIANCE	Variance of list of argument
WHEN-COMPILED	Program compile date and time

Screen Handling Module

The screen handling module displays data on the display and accepts input data from the display via the screen control system. The SCREEN SECTION in the DATA DIVISION defines the screen position where the data is to be displayed and the screen position where the data is to be input. The SCREEN SECTION not only defines the arrangement of screen areas but also defines the display color and input-output attribute.

Use the following descriptions as required in a program using the screen handling module.

Program Description	**Main Facility**
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SPECIAL-NAMES paragraph	Specifies the data-name for accepting the cursor position and screen input status.
DATA DIVISION	
SCREEN SECTION	
Screen description entry	Defines the attribute of areas on the screen.
PROCEDURE DIVISION	
Procedure portion	
DISPLAY	Displays the screen.
ACCEPT	Inputs data from the screen.

Screen and Screen Item

The screen can be regarded as an area of a fixed size for displaying combinations of lines and columns. If the screen displayed at any one time is called a single screen, then several areas can be arranged in a single screen.

Screen Data Description Entry

A screen area can be defined as a screen item in the SCREEN SECTION screen data description entry . Define the arrangement of areas on the screen by specifying line numbers and column numbers in the screen item. Count the line numbers from the top of the physical screen starting from 1. Count the column numbers from the single alphanumeric character at the left of the physical screen starting from 1. Define the line number in the LINE NUMBER clause and the column number in the COLUMN NUMBER clause.

The screen data description entry can also define the following items for areas on the screen: input-output attribute, display color, brightness, cursor positioning method, and the attribute which specifies whether the screen is to be erased before display.

In addition to the screen item, data items for operation must also be defined before values set by the program can be displayed on the screen or data input from the screen can be accepted by the program. Specify the correspondence between the screen item and the data items in the screen data description entry.

Screen Item

There are two types of screen items: elementary screen items and group screen items. An area from one column to another column in a line is an elementary screen item. An area containing several elementary screen items is a group screen item. Several group screen items can also be grouped together and defined as a group screen item.

An elementary screen item is the smallest unit of screen item processed in one ACCEPT statement or DISPLAY statement. When a group screen item has been specified in the ACCEPT statement or DISPLAY statement, the elementary screen items subordinate to the group screen item can be processed all at one time.

There are five types of elementary screen items:

- Literal item
- Input item
- Output item
- Update item
- Input-output item

Literal Item

A literal item is a screen area for displaying a fixed value on the screen. Specify the value of the literal item in the VALUE clause of the screen data description entry. Executing the DISPLAY statement displays the value specified in the VALUE clause.

Input Item

An input item is a screen area for accepting data from the screen. Associate one data item with one input item in the screen data description entry. Executing an ACCEPT statement sets the data input from the screen in the data item associated with the input item. The input item cannot display a value set in the program.

Output Item

An output item is a screen area for displaying data on the screen. Associate one data item with one output item in the screen data description entry. Setting a value in the data item associated with the output item and executing a DISPLAY statement displays the set value. Data cannot be input from the screen to the output item.

Update Item

An update item is a screen area for accepting data from the screen and displaying it on the screen. Associate one data item with one update item in screen data description entry. The value of the data before update is not maintained. Setting a value in the data item associated with the update item and executing a DISPLAY statement displays the set value. Executing an ACCEPT statement sets the data input from the screen in the data item associated with the update item.

Input-Output Item

Input-output item is a screen area for accepting data from the screen and displaying data on the screen. Associate two data items for input and for output with one update item in the screen data description entry. The value of the data item before update is maintained. Setting a value in the data item for output and executing a DISPLAY statement displays the set value. Executing an ACCEPT statement sets the data input from the screen in the data item for input.

Input-Output Handling of the Screen

Execute a DISPLAY statement to display the screen. Execute an ACCEPT statement to input data from the screen.

Either “elementary screen item” or “group screen item” can be specified in a DISPLAY statement or ACCEPT statement. Specifying “elementary screen item “ causes input-output handling to be performed for the elementary screen item only. Specifying “group screen item” causes input-output handling to be performed for elementary group items subordinate to a group screen item.

Executing an ACCEPT statement causes data input from the screen to be stored in a data item associated with the input item or update item.

Executing a DISPLAY statement causes the value set in the data item associated with the output item or update item or the value of the literal item to be displayed on the screen. The screen item is displayed either after the current screen is erased or over the top of the current screen, depending on whether the BLANK clause or ERASE clause has been specified in the screen item.

Screen Input Status

A screen input status is a three character area for indicating the result of execution of an ACCEPT statement. The value of the screen input status is set before the unconditional statement written in the ACCEPT statement is executed and then each time the ACCEPT statement is executed.

The screen input status can be referenced by writing a CRT STATUS clause in the special names paragraph of the ENVIRONMENT DIVISION.

See the section titled “CRT STATUS clause,” for a list of the screen input status.

Operation Module for Command Line Argument and Environment Variable

The operation module for the command line argument and environment variable consists of the following two modules:

- A module for referencing the number of arguments specified in the command line and the values of the arguments

  A command line is a command which accesses the COBOL object program.

- A module for referencing and updating the value of the environment variable

  To operate the command line argument and environment variable, use a combination of ACCEPT statements and DISPLAY statements.

In an ACCEPT statement and DISPLAY statement, specify the mnemonic-name associated with the function-name to specify the processing classification.

Use the following descriptions as required in a program which uses an operation module for the command line argument or environment variable.

Program Description	Main Facility
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SPECIAL-NAMES paragraph	Associates a function-name with a mnemonic name.
PROCEDURE DIVISION	
Procedure portion	
ACCEPT	Obtains the number of arguments in the command line and their values. Obtains the value of the environment variable.
DISPLAY	Positions arguments in the command line. Positions the environment variable.

Operating a Command Line Argument

A parameter to be passed to the COBOL program, an external switch, or an execution time option can be specified as an argument in the command line. The number of arguments and their values can be referenced using the operation module for the command line argument.

Method for Finding the Number of Arguments

Execute an ACCEPT statement to find the number of arguments. Write the mnemonic-name associated with the function-name ARGUMENT-NUMBER in the FROM phrase of the ACCEPT statement. Executing the ACCEPT statement sets the number of arguments in the identifier of the ACCEPT statement.

Method for Referencing the Value of Arguments

There are two methods for referencing the value of arguments:

- Referencing the values in the sequence specifying the arguments
- Specifying the position of the arguments and then referencing the values

Determine the argument positions to be referenced using the argument position indicator.

Argument Position Indicator

An argument position indicator is an indicator provided in the concept of COBOL for specifying the argument to be referenced next.

Each argument specified in the command line is given a number in the sequence in which the arguments were specified (that is, moving toward the right). The initial value of the number is 1 and it is incremented by 1. The value of the argument position indicator represents the number given to the argument.

The value of the argument position indicator is an integer between 0 and 99. The value 0 corresponds to the command.

The value of the argument position indicator is affected only by a DISPLAY statement which specifies a mnemonic-name associated with the function-name ARGUMENT-NUMBER or by an ACCEPT statement which specifies a mnemonic-name associated with the function-name ARGUMENT-VALUE. The initial value of the argument position indicator before the DISPLAY and ACCEPT statements have been executed is 1.

Referencing the Values in the Sequence the Arguments Were Specified

Execute the ACCEPT statement repeatedly to reference the values in the sequence specifying the arguments. Write the mnemonic-name associated with the function-name ARGUMENT-VALUE in the FROM phrase of the ACCEPT statement.

Each time the ACCEPT statement is executed, the value of the argument in the position indicated by the current argument position indicator is set in the identifier of the ACCEPT statement. The value of the argument position indicator is incremented by one each time the ACCEPT statement is executed.

Specifying the Position of the Arguments and Then Referencing the Values

To specify the position of an argument and then reference the value, execute a DISPLAY statement and ACCEPT statement in that order.

Specify the position of the argument to be referenced in the identifier or literal of the DISPLAY statement. Then, write the mnemonic-name associated with the function-name ARGUMENT-NUMBER in the UPON phrase. Executing the DISPLAY statement sets the value specified in the identifier or literal of the DISPLAY statement in the argument position indicator.

Write the mnemonic-name associated with the function-name ARGUMENT-VALUE in the FROM phrase of the ACCEPT statement. Executing the ACCEPT statement sets the value of the argument in the position indicated by the current argument position indicator in the identifier of the ACCEPT statement. The current argument position indicator is then incremented by 1.

Operating an Environment Variable

The value of an environment variable can be referenced and updated using the environment variable operation module .

Referencing the Value of an Environment Variable

To reference the value of an environment variable, execute a DISPLAY statement and ACCEPT statement in that order.

Specify the environment variable name to be referenced in the identifier or literal of the DISPLAY statement. Then, write the mnemonic-name associated with the function-name ENVIRONMENT-NAME in the UPON phrase. Executing the DISPLAY statement enables input-output operation of the environment variable whose name has been specified in the identifier or literal of the DISPLAY statement.

Write the mnemonic-name associated with the function-name ENVIRONMENT-VALUE in the FROM phrase of the ACCEPT statement. Executing the ACCEPT statement sets the value of the environment variable enabling input-output operation by execution of the DISPLAY statement in the identifier of the ACCEPT statement.

Updating the Value of an Environment Variable

To update the value of an environment variable, execute two DISPLAY statements.

In the first DISPLAY statement, specify the name of the environment variable to be updated in the identifier or literal and write the mnemonic-name associated with the function-name ENVIRONMENT-NAME in the UPON phrase. Executing the DISPLAY statement enables input-output operation of the environment variable specified in the identifier or literal of the DISPLAY statement.

In the second DISPLAY statement, set the updated value in the identifier or literal and write the mnemonic-name associated with the function-name ENVIRONMENT-VALUE in the UPON phrase. Executing the DISPLAY statement causes the environment variable for which input-output operation has been enabled by execution of the first DISPLAY statement to be replaced by the value set in the identifier or literal of the DISPLAY statement.

Report Writer Module

A report writer module simplifies the procedure for writing a report by defining the report format in the DATA DIVISION. The report writer control system automatically performs processing necessary for determining the columns of the report. This eliminates the need to write processing such as the page number count, number of lines in a page, composition of a print line, and line feed in the PROCEDURE DIVISION.

The file outputting the report is called the “report file.” The report file is treated in the same way as a print file in the sequential I-O module. Define the report file in the DATA DIVISION file description entry .

Several report styles can be output to a single report file. Associate one page style with one report style, and define the report style in the report description entry of the DATA DIVISION. Define the relationship between the report file and the report description entry in the file control entry REPORT clause.

Several data items can be output to a report. Define the data to be output to the report in the report group description entry. Define the report group description entry in the hierarchical structure.

Use the following descriptions as required in a program using a report writer module.

Program Description	Main Facility
ENVIRONMENT DIVISION	
INPUT-OUTPUT SECTION	
FILE-CONTROL paragraph	
File control entry	Associates the file-name of a report file with an external medium. Defines the file organization and access mode.
I-O CONTROL paragraph	Specifies the storage area to be shared by several files containing report files.
DATA DIVISION	
FILE SECTION	
File description entry	Defines the physical structure of a report file. Associates the report file with a report-name.
REPORT SECTION	Defines the physical structure of a report.
Report description entry	Defines the items in a report.
Report group description entry	
PROCEDURE DIVISION	
Declarative	
USE statement and USE AFTER STANDARD EXCEPTION	Define the input-output error procedure.
USE BEFORE REPORTING statement and USE BEFORE REPORTING procedure	Define the procedure to be executed before the report group is displayed.
Procedure portion	
OPEN	Opens a file.
INITIALIZE	Starts report processing.
GENERATE	Writes a report.
SUPPRESS	Suppresses display of a report group.
TERMINATE	Ends report processing.
CLOSE	Closes a file.

Report File

A report file is a sequential organization output file. Define the report file in the file description entry of the DATA DIVISION. Write a REPORT clause in the file description entry. Associate the report file and report description entry by specifying the report-name of the report description entry in the REPORT clause.

A record description entry cannot be written in the FILE SECTION when a report file is defined, though it can be written when defining a sequential file. Define the record structure in the report file in the REPORT SECTION report group description entry . The report writer control system controls writing of records from the report file.

A report file can be referenced using the OPEN, GENERATE, INITIATE, SUPPRESS, TERMINATE, USE AFTER STANDARD EXCEPTION, USE BEFORE REPORTING, or CLOSE statement.

Special Register

The following two special registers are created for the report file module:

- PAGE-COUNTER
- LINE-COUNTER

One of each of these special registers is created for each report defined in the REPORT SECTION. The special register PAGE-COUNTER is called a “page counter,” and the special register LINE-COUNTER is called a “line-counter.”

Page Counter

A page counter is treated as a numeric data item defined as “PIC 9(6) USAGE IS PACKED-DECIMAL.” The range of values of the page counter is 1 to 999999. The page counter value is controlled by the report writer control system and can be used in a program to add report page numbers. Only the SOURCE clause of the REPORT SECTION or a statement in the PROCEDURE DIVISION can reference the page counter. The page counter value can be changed by the program.

Line-Counter

A line-counter is treated as a numeric data item defined as “PIC 9(6) USAGE IS PACKED-DECIMAL”. The range of values of the line-counter is 1 to 999999. The line-counter value is controlled by the report writer control system and can be used to determine a position within the length of a report. Only the SOURCE clause of the REPORT SECTION or a statement in the PROCEDURE DIVISION can reference the line-counter. The line-counter value can be changed by the program.

Chapter 3. Identification Division and End Program Header

The identification division specifies the name identifying the program and also the attribute of the program. The identification division must be placed at the start of the program.

The end program header specifies the end of the program.

Identification Division

The format of the identification division is shown below.

[Format]

IDENTIFICATION DIVISION.		IDENTIFICATION DIVISION
PROGRAM-ID. program identification-entry	] Program name paragraph	
[AUTHOR. [comment entry]...]	]AUTHOR paragraph	
[INSTALLATION. [comment entry]...]	] INSTALLATION paragraph	
[DATE-WRITTEN. [comment entry]...]	] DATE-WRITTEN paragraph	
[DATE-COMPILED. [comment entry]...]	] Compile date paragraph	
[SECURITY. [comment entry]...]	] SECURITY paragraph	

Syntax Rules

1. The program name paragraph must be the first paragraph in the identification division. The program name must be written in the program name paragraph.

2. The program name paragraph can be followed by the AUTHOR paragraph, INSTALLATION paragraph, DATE-WRITTEN paragraph, compile date paragraph, or SECURITY paragraph.

   The COBOL85 compiler allows the paragraphs to be written in any sequence. These paragraphs are obsolete language elements used for writing comment lines.

Program Name Paragraph

This paragraph specifies the name of the program.

[Format]

```
              ┌ program-name         ┐
PROGRAM-ID.   {                      }
              └ program-name-literal ┘

          ┌    ┌┃ COMMON  ┃┐          ┐
          │ IS {┃         ┃} PROGRAM  │ .
          └    └┃ INITIAL ┃┘          ┘
```

Syntax Rules

1. Specify the name of the program in program-name or program-name-literal. The name of an internal program must be the same as the name of the program which directly or indirectly contains the internal program.

2. The COMMON clause can be written in an internal program only.

3. Program-name must conform to the rules for user-defined words.

4. See the section titled “Literal for special applications,” for the rules for describing program-name-literal.

General Rules

1. The character-string specified in program-name or program-name-literal identifies the source program, object program, and all printed outputs.
2. Write a COMMON clause to give the common attribute to an internal program. A program having the common attribute is called a “common program.” A common program can be called from programs other than the program containing the common program.
3. Write the INITIAL clause to give the initial attribute to a program. A program having the initial attribute is called an “initial program.” Accessing an initial program starts the initial program itself and all other programs contained in the initial program.
4. When an ENTRY statement has been written in an initial program and a secondary entry name has been specified, accessing the initial program by referencing the secondary entry name also initializes the initial program and all other programs contained in the initial program.

Compile Date Paragraph

Insert the compile date paragraph in the source program list identification division. The compile date paragraph is an obsolete language element.

[Format]

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

General Rules

Writing the header (DATE-COMPILED) of this paragraph inserts the compile date in the source program list output at compilation. The compile date paragraph is replaced by the following format in the source program list.

$$\underline{\text{DATE-COMPILED}}\text{. compile-date.}$$

End Program Header

The end program header specifies the end of a program.

[Format]

$$\underline{\text{END}}\ \ \underline{\text{PROGRAM}}\begin{Bmatrix}\text{program-name}\\ \text{program-name-literal}\end{Bmatrix}\ .$$

Syntax Rules

1. Specify the name of the program in program-name or program-name-literal. The name of the program must be the same as the name of the program specified in the first program name paragraph.
2. The program name paragraph and end program header in the identification division must be written in pairs. For example, if program A contains program B, the end program header for program B must be written before the end program header for program A.
3. Program-name must conform to the rules for user-defined words.
4. See the section titled “Literal for special applications,” for the rules for describing program-name-literal.

General Rules

1. Specify the end program header at the end of the program.
2. When writing internal programs in a separately compiled program, an end program header must be written in each program in the separately compiled program.
3. One of the following must be written immediately after the end program header in an internal program:
   a. A header for the identification division of another internal program
   b. An end program header for the program including the internal program
4. When writing two or more separately compiled programs in succession, the end program headers must be written according to the following rules:
   a. An end program header indicating the end of the separately compiled program must be written in all separately compiled programs except the last one.
   b. The end program header indicating the end of the last separately compiled program can be omitted.

Chapter 4. Environment Division

Write an environment suited to the characteristics of the computer to be used in the environment division. Write the following information in the environment division:

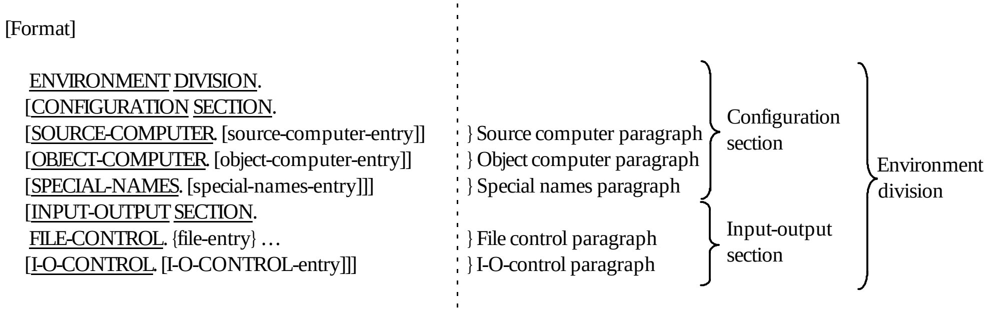
- Definition of character set
- Definition of collating sequence
- Association of function-name and mnemonic-name
- Definition of symbolic-constant, symbolic-character, and alphabet-name
- Association of file and external medium

Write the environment division after the identification division. The environment division may be omitted.

Configuration of Environment Division

The environment division consists of the configuration section and input-output section.

The configuration of the environment division is shown below. The sections and paragraphs making up the environment division must be written in the sequence shown.



[Format]

```
 ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER. [source-computer-entry]]    } Source computer paragraph
[OBJECT-COMPUTER. [object-computer-entry]]    } Object computer paragraph
[SPECIAL-NAMES. [special-names-entry]]]       } Special names paragraph
[INPUT-OUTPUT SECTION.
 FILE-CONTROL. {file-entry} …                 } File control paragraph
[I-O-CONTROL. [I-O-CONTROL-entry]]]           } I-O-control paragraph
```

Configuration Section

Write the characteristics of the source computer and object computer in the configuration section.

The configuration section can be written only in the outermost program. It cannot be written in an internal program.

Source Computer Paragraph

Specify the computer to be used to compile the program in the source computer paragraph.

[Format]

```
SOURCE-COMPUTER.
   [computer-name [WITH DEBUGGING MODE clause].]
```

Syntax Rules

See the section titled “System-name,” for the rules for describing computer-name.

General Rules

1. All clauses in the source computer paragraph apply both to the program which explicitly or implicitly specified the clause and to the internal programs it contains.
2. When a source computer paragraph has not been written in the outermost program or any of the internal programs, the system assumes that the source computer will compile the source program.
3. When the paragraph name of the source computer paragraph is written but source-computer-entry is omitted, the system assumes that the source computer will compile the source program.

WITH DEBUGGING MODE Clause

This clause specifies treatment of the debugging line.

[Format]

```
WITH DEBUGGING MODE
```

General Rules

1. When a WITH DEBUGGING MODE clause has been written, all debugging lines written in the program and internal programs in which the clause has been written are compiled exactly as they are written.
2. When a WITH DEBUGGING MODE clause has not been written in the outermost program or any of the internal programs, all debugging lines in the source program are regarded as comment lines.
3. When a COPY statement or REPLACE statement has been written in the source program, the program waits for processing of these statements to end and then checks whether a WITH DEBUGGING MODE clause has been written.

Object Computer Paragraph

Specify the computer to be used to run the program in the object computer paragraph.

[Format]

```
OBJECT-COMPUTER.
        [computer-name [MEMORY SIZE clause]
              [PROGRAM COLLATING SEQUENCE clause].]
```

Syntax Rules

See the section titled “System-name,” for the rules for describing computer-name.

General Rules

All clauses in the object computer paragraph apply both to the program which explicitly or implicitly specified the clause and to the internal programs it contains.

MEMORY SIZE Clause

This clause specifies information relating to storage capacity. It is an obsolete language element.

[Format]

$$\underline{\text{MEMORY}}\ \ \text{SIZE}\ \ \text{interger-1}\ \left\{\begin{array}{l}\underline{\text{WORDS}}\\ \underline{\text{CHARACTER}}\text{S}\\ \underline{\text{MODULES}}\end{array}\right\}$$

The COBOL85 compiler regards the MEMORY SIZE clause as a comment.

PROGRAM COLLATING SEQUENCE Clause

This clause specifies the collating sequence.

[Format]

$$\text{PROGRAM COLLATING }\underline{\text{SEQUENCE}}\text{ IS alphabet-name-1}$$

Syntax Rules

Alphabet-name-1 must be associated with the collating sequence in the ALPHABET clause of the special names paragraph.

Alphabet-name-1 must not be the mnemonic-name associated with the function-name in the ALPHABET clause.

General Rules

1. When a PROGRAM COLLATING SEQUENCE clause has been written, the collating sequence associated with alphabet-name-1 is adopted as the collating sequence for the program.
2. When the PROGRAM COLLATING SEQUENCE clause has been omitted, the native collating sequence is adopted as the collating sequence for the program.
3. If a character relation (excluding a national character relation) is performed in any of the following places, the collating sequence specified in the PROGRAM COLLATING SEQUENCE clause is used to determine the truth value of the condition:

   a. Relation condition specified explicitly

   b. condition-name condition specified explicitly

   c. CONTROL clause in a report description entry specified implicitly
4. When the COLLATING SEQUENCE specification has been omitted from a SORT statement or MERGE statement, the collating sequence for the program specified in the PROGRAM COLLATING SEQUENCE clause also applies to the SORT statement and MERGE statement.
5. The collating sequence of characters specified in the PROGRAM COLLATING SEQUENCE clause does not affect the collating sequence of national characters.

Special Names Paragraph

Define the symbolic-constant, symbolic-character, alphabet-name, character set, and collating sequence in the special names paragraph. Also, associate the mnemonic-name with the function-name.

[Format]

```
SPECIAL-NAMES.
 [[{function-name-1-clause} ...]
  [{function-name-2-clause} ...]
  [{function-name-3-clause} ...]
  [{ALPHABET clause} ...]
  [{CLASS clause} ...]
  [CRT STATUS clause]
  [CURRENCY SIGN clause]
  [CURSOR clause]
  [DECIMAL-POINT IS COMMA clause]
  [{POSITIONING UNIT clause} ...]
  [{PRINTING MODE clause} ...]
  [{SYMBOLIC CHARACTERS clause} ...]
  [{SYMBOLIC CONSTANT clause} ...].]
```

General Rules

All clauses in the special names paragraph apply both to the program which explicitly or implicitly specified the clause and to the internal programs it contains.

Function-name-1-clause

This clause associates the function-name relating to the input-output operation on data to the mnemonic-name.

[Format]

```
function-name-1 IS mnemonic-name-1
```

Syntax Rules

1. When writing mnemonic-name-1 associated with function-name-1 in the FROM phrase of the ACCEPT statement or the UPON phrase of the DISPLAY statement, function-name-1 must be one of the following:

    a. SYSOUT

    b. CONSOLE

    c. SYSIN

    d. SYSERR

    e. SYSPUNCH

    SYSOUT and SYSPUNCH are regarded as being synonymous.

2. When writing mnemonic-name-1 associated with function-name-1 in the BEFORE ADVANCING phrase or the AFTER ADVANCING phrase of the WRITE statement, function-name-1 must be one of the following:

    a. CHANNEL-01

    b. CHANNEL-02, ... , CHANNEL-12

    c. SLC

    d. CTL

    e. STACKER-01, STACKER-02

Each of CHANNEL-02 to CHANNEL-12, STACKER-01, and STACKER-02 is regarded as being synonymous with CHANNEL-01.

3. When writing mnemonic-name-1 associated with function-name-1 in the CHARACTER TYPE clause, function-name-1 must be one of the following:

   HSC

   F0202

   H0202

   F0102

   F0201

   YX-7P,

   YX-7P-12,YX-7P-21,YX-7P-22,

   YX-7P-H,

   YX-7P-12-H,YX-7P-21-H,YX-7P-22-H,

   YX-9P,

   YX-9P-12,YX-9P-21,YX-9P-22,

   YX-9P-H,

   YX-9P-12-H,YX-9P-21-H,YX-9P-22-H,

   YX-12P,

   YX-12P-12,YX-12P-21,YX-12P-22,

   YX-12P-H,

   YX-12P-12-H,YX-12P-21-H,YX-12P-22-H,

   YA,YA-12,YA-21,YA-22,

   YA-H,YA-12-H,YA-21-H,

   YA-22-H,

   YB,YB-12,YB-21,YB-22,

   YB-H,YB-12-H,YB-21-H,

   YB-22-H,

YC,YC-12,YC-21,YC-22,
YC-H,YC-12-H,YC-21-H,
YC-22-H,
YD-9P,YD-9P-12,
YD-9P-21,YD-9P-22,
YD-9P-H,YD-9P-12-H,
YD-9P-21-H,YD-9P-22-H,
YD-12P,YD-12P-12,
YD-12P-21,YD-12P-22,
YD-12P-H,YD-12P-12-H,
YD-12P-21-H,YD-12P-22-H,
TX-7P,TX-7P-12,
TX-7P-21,TX-7P-22,
TX-7P-H,TX-7P-12-H,
TX-7P-21-H,TX-7P-22-H,
TX-9P,TX-9P-12,
TX-9P-21,TX-9P-22,
TX-9P-H,TX-9P-12-H,
TX-9P-21-H,TX-9P-22-H,
TX-12P,TX-12P-12,
TX-12P-21,TX-12P-22,
TX-12P-H,TX-12P-12-H,
TX-12P-21-H,TX-12P-22-H,
TA,TA-12,TA-21,TA-22,
TA-H,TA-12-H,TA-21-H,
TA-22-H,
TB,TB-12,TB-21,TB-22,
TB-H,TB-12-H,TB-21-H,
TB-22-H,

TC,TC-12,TC-21,TC-22,
TC-H,TC-12-H,TC-21-H,
TC-22-H,
TD-9P,TD-9P-12,
TD-9P-21,TD-9P-22,
TD-9P-H,TD-9P-12-H,
TD-9P-21-H,TD-9P-22-H,
TD-12P,TD-12P-12,
TD-12P-21,TD-12P-22,
TD-12P-H,TD-12P-12-H,
TD-12P-21-H,TD-12P-22-H,
GYX-7P,GYX-7P-12,
GYX-7P-21,GYX-7P-22,
GYX-7P-H,GYX-7P-12-H,
GYX-7P-21-H,GYX-7P-22-H,
GYX-9P,GYX-9P-12,
GYX-9P-21,GYX-9P-12-22,
GYX-9P-H,GYX-9P-12-H,
GYX-9P-21-H,GYX-9P-22-H,
GYX-12P,GYX-12P-12,
GYX-12P-21,GYX-12P-12-22,
GYX-12P-H,GYX-12P-12-H,
GYX-12P-21-H,GYX-12P-22-H,
GYA,GYA-12,GYA-21,GYA-22,
GYA-H,GYA-12-H,GYA-21-H,
GYA-22-H,
GYB,GYB-12,GYB-21,GYB-22,
GYB-H,GYB-12-H,GYB-21-H,
GYB-22-H,

GYC,GYC-12,GYC-21,GYC-22,
GYC-H,GYC-12-H,GYC-21-H,
GYC-22-H,
GYD-9P,GYD-9P-12,
GYD-9P-21,GYD-9P-22,
GYD-9P-H,GYD-9P-12-H,
GYD-9P-21-H,GYD-9P-22-H,
GYD-12P,GYD-12P-12,
GYD-12P-21,GYD-12P-22,
GYD-12P-H,GYD-12P-12-H,
GYD-12P-21-H,GYD-12P-22-H,
GTX-7P,GTX-7P-12,
GTX-7P-21,GTX-7P-22,
GTX-7P-H,GTX-7P-12-H,
GTX-7P-21-H,GTX-7P-22-H,
GTX-9P,GTX-9P-12,
GTX-9P-21,GTX-9P-12-22,
GTX-9P-H,GTX-9P-12-H,
GTX-9P-21-H,GTX-9P-22-H,
GTX-12P,GTX-12P-12,
GTX-12P-21,GTX-12P-12-22,
GTX-12P-H,GTX-12P-12-H,
GTX-12P-21-H,GTX-12P-22-H,
GTA-H,GTA-12-H,GTA-21-H,
GTA-22-H,
GTB,GTB-12,GTB-21,GTB-22,
GTB-H,GTB-12-H,GTB-21-H,
GTB-22-H,
GTC,GTC-12,GTC-21,GTC-22,

GTC-H,GTC-12-H,GTC-21-H,
GTC-22-H,
GTD-9P,GTD-9P-12,
GTD-9P-21,GTD-9P-22,
GTD-9P-H,GTD-9P-12-H,
GTD-9P-21-H,GTD-9P-22-H,
GTD-12P,GTD-12P-12,
GTD-12P-21,GTD-12P-22,
GTD-12P-H,GTD-12P-12-H,
GTD-12P-21-H,GTD-12P-22-H

Function-name-2-clause

This clause associates the function-name for using an external switch with the mnemonic-name.

[Format]

```
function-name-2
  { IS mnemonic-name-1 [ON  STATUS IS condition-name-1      }
  {                    [OFF STATUS IS condition-name-2]]    }
  { IS mnemonic-name-2 [OFF STATUS IS condition-name-2      }
  {                    [ON  STATUS IS condition-name-1]]    }
  { ON  STATUS IS condition-name-1                          }
  {             [OFF STATUS IS condition-name-2]            }
  { OFF STATUS IS condition-name-2                          }
  {             [ON  STATUS IS condition-name-1]            }
```

Syntax Rules

1. Function-name-2 must be SWITCH-n (where n is an integer between 0 and 8).
2. The mnemonic-name associated with function-name-2 can be specified in the operand of the SET statement.

General Rules

1. SWITCH-n represents the external switch. SWITCH-0 and SWITCH-8 are synonymous. They represent the same external switch.
2. To check the ON status and OFF status of the external switch, associate the two statuses with condition-name. See the section titled “Switch-status condition,” for details on the method for checking the external switch.
3. The status of the external switch can be changed by executing a SET statement (Format 3 SET statement) which contains the mnemonic-name associated with the external switch.
4. A condition-name specified in the special names paragraph of the program which contains the condition-name can also be referenced from programs contained within the program.

Function-name-3-clause

This clause associates the function-name used for operating a command line argument and environment variable with the mnemonic-name.

[Format]

```
function-name-3 IS mnemonic-name-1
```

Syntax Rules

The table below shows the locations in which a name to be specified in function-name-3-clause or the mnemonic-name associated with function-name-3 can be written.

Name Specified in function-name-3-clause	**Location Where a Mnemonic-name Associated with function-name-3 Can Be Written**
ARGUMENT-NUMBER	FROM phrase in ACCEPT statement and UPON phrase of DISPLAY statement
ARGUMENT-VALUE	FROM phrase in ACCEPT statement
ENVIRONMENT-NAME	UPON phrase in DISPLAY statement
ENVIRONMENT-VALUE	FROM phrase in ACCEPT statement and UPON phrase in DISPLAY statement

General Rules

1. Use ARGUMENT-NUMBER to input the number of arguments in a command line or to position the arguments in a command line.
2. Use ARGUMENT-VALUE to input the value of arguments in a command line.
3. Use ENVIRONMENT-NAME to position the environment variable.
4. Use ENVIRONMENT-VALUE to input the value of the environment variable or to set a value in the environment variable.

ALPHABET Clause

This clause associates the alphabet-name with a character set or the collating sequence.

[Format]

```
ALPHABET alphabet-name-1 IS
  ⎧ STANDARD-1                                                  ⎫
  ⎪ STANDARD-2                                                  ⎪
  ⎪ NATIVE                                                      ⎪
  ⎨ EBCDIC                                                      ⎬
  ⎪ function-name-1                                             ⎪
  ⎪ ⎧            ⎡⎧THROUGH⎫  literal-2  ⎤⎫                     ⎪
  ⎪ ⎨literal-1   ⎢⎩THRU   ⎭             ⎥⎬ ...                 ⎪
  ⎩ ⎩            ⎣{ ALSO literal-3} ...  ⎦⎭                     ⎭
```

Syntax Rules

1. Function-name-1 must be CODE-n (where n is an integer between 0 and 9).

2. literal-1 to literal-3 must conform to the following rules:

   a. Any numeric literal specified in literal-1, literal-2, or literal-3 must be an integer between 1 and 256 and must not contain a sign.

   b. When writing a THROUGH phrase and specifying a nonnumeric literal in literal-1 and literal-2, the length of literal-1 and literal-2 must be a single character. Similarly, when writing an ALSO phrase and specifying a nonnumeric literal in literal-1 and literal-3, the length of literal-1 and literal-2 must be a single character.

   c. literal-1 to literal-3 must not be a national nonnumeric literal, Boolean literal, symbolic-character, or symbolic-constant.

3. The same character must not be specified more than once in a literal of a literal phrase.
4. THROUGH and THRU are synonymous.

General Rules

1. In the ALPHABET clause, add the name to the character alphabet and collating sequence. alphabet-name-1 can be written in the following locations:
   a. In the PROGRAM COLLATING SEQUENCE clause of the object computer paragraph
   b. In the COLLATING SEQUENCE of the SORT statement
   c. In the COLLATING SEQUENCE of the MERGE statement
   d. In the CODE-SET clause of the file description entry
   e. In the IN phrase of the SYMBOLIC CHARACTERS clause of the special names paragraph
2. When STANDARD-1 has been written, the character alphabet and collating sequence conform to the rules for the standard character set. Each character in the standard character set is associated with a corresponding character in the native character set. The standard character set in the COBOL85 compiler conforms to ASCII.
3. When STANDARD-2 has been written, the character alphabet and collating sequence conform to the rules of the ISO646 international standard. Each character in the standard character set is associated with a corresponding character in the native character set.
4. When EBCDIC has been written, the character alphabet and collating sequence conform to the rules for the EBCDIC code.

Each character in the EBCDIC code is associated with a corresponding character in the native character set.

5. When NATIVE has been written or when the ALPHABET clause has been omitted, the character alphabet and collating sequence conform to the native character set and native collating sequence.

6. CODE-n specified in function-name-1 represents the conversion table. When function-name-1 has been written, the character alphabet and collating sequence conform to the definition in the conversion table.

7. When a literal phrase has been written, the native character set is assumed as the character alphabet. Define the collating sequence in the literal phrase as follows:

   a. When writing a numeric literal in the literal phrase, specify the sequence numbers of the characters in the native character set in each literal. When writing a nonnumeric literal in the literal phrase, specify the actual characters of the native character set in each literal. When the value of the nonnumeric literal contains several characters, the characters in the literal are regarded as being collated in ascending order starting from the left.

   b. When two or more literal phrases have been written, the collating sequence is ascending in the sequence in which the literal phrases have been written.

   c. Characters belonging to the native character set which have not been written in the literal phrase are placed in a higher sequence position than characters which have been written in the literal phrase. The collating sequence of characters which have not been written in the literal phrase conforms to the native collating sequence.

d. When the THROUGH phrase has been written, the collating sequence of all characters which are in literal-1 to literal-2 and which belong to the native character set is ascending in the order literal-1 to literal-2. Characters in a higher sequence position than literal-2 can be specified in literal-1 in the collating sequence for the native character set.

e. When the ALSO phrase has been written, characters which are indicated by the value of literal-1 and literal-3 are regarded as being in the same sequence position as in the collating sequence. When alphabet-name-1 has been specified in the IN phrase of the SYMBOLIC CHARACTERS clause, only literal-1 is used to indicate the characters in the native character set.

8. The character having the highest sequence position in the collating sequence is assumed as the value of the figurative constant HIGH-VALUE, except when the figurative constant has been specified in literal-1 to literal-3. When there are two or more characters in the highest sequence position, the character which was most recently specified is assumed as the value of the figurative constant HIGH-VALUE.

9. The character having the lowest sequence position in the collating sequence is assumed as the value of the figurative constant LOW-VALUE, except when the figurative constant has been specified in literal-1 to literal-3. When there are two or more characters in the lowest sequence position, the character which was specified first is assumed as the value of the figurative constant LOW-VALUE.

10. When the figurative constants HIGH-VALUE and LOW-VALUE have been specified as the constants in the special names paragraph, HIGH-VALUE and LOW-VALUE represent the highest and lowest characters in the native character set when these constants are referenced.

CLASS Clause

This clause defines the class-name.

[Format]

$$\underline{\text{CLASS}}\ \text{class-name-1}\ \ \text{IS}$$

$$\left\{\text{literal-1}\left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\}\ \text{literal-2}\right]\right\}\ \ldots$$

Syntax Rules

1. literal-1 and literal-2 must conform to the following rules:

   a. Any numeric literal specified in literal-1 or literal-2 must be an unsigned integer between 1 and 256.

   b. When writing a THROUGH phrase and specifying a nonnumeric literal in literal-1 or literal-2, the length of each literal must be a single character.

   c. literal-1 and literal-2 must not be a national nonnumeric literal, Boolean literal, symbolic-character, or symbolic-constant.

2. THROUGH and THRU are synonymous.

General Rules

1. In the CLASS clause, add the name to a combination of characters. Class-name-1 can be written only in the class condition.

2. Specify the following literal in literal-1 and literal-2:

   a. When specifying a numeric literal in literal-1 or literal-2, specify the sequence number of characters in the native character set in each literal.

   b. When specifying a nonnumeric literal in literal-1 or literal-2, specify the actual characters of the native character set in each literal. When the value of the nonnumeric literal contains several characters, the characters in the literal are included in the combination of characters which are identified according to the class-name.

   c. When the THROUGH phrase has been written, all characters which are in literal-1 to literal-2 and which belong to the native character set are associated with the combination of characters having the name in class-name-1. The characters in the native character set can also be specified in ascending or descending order depending on the relationship of high and low values in literal-1 and literal-2.

CRT STATUS Clause

This clause specifies a data item for accepting the screen input status.

[Format]

<u>CRT</u> <u>STATUS</u> IS data-name-1

Syntax Rules

1. data-name-1 must be a group item specified in the working-storage section or an alphanumeric data item. There must be three character positions in the data item of data-name-1.

2. When specifying a group item in data-name-1, define each character position in the group item as follows:

    a. Specify a single character alphanumeric data item or an unsigned single column zoned decimal item in the first character position.

    b. Specify a single character alphanumeric data item or an unsigned single column zoned decimal item in the second character position.

    c. Specify a single character alphanumeric data item in the third character position.

General Rules

1. In the CRT STATUS clause, specify a data item for accepting the screen input status in the ACCEPT statement of the screen handling module.

2. The contents of data-name-1 are set and updated each time an ACCEPT statement is executed.

3. The first character position in data-name-1 is an area for accepting screen input status key-1. The status which ended the ACCEPT statement is set in screen input status key-1. The table below lists the meanings of screen input status key-1.

Screen Input Status Key-1	**Meaning of Screen Input Status Key-1**
0	Return from end key or last item
1	Return by user-defined function key
2	Return by default function key
9	Error

4. The value “0” in screen input status key-1 indicates that the ACCEPT statement ended normally.

5. The second character position in data-name-1 is an area for accepting screen input status key-2. The detailed information code of the status which ended the ACCEPT statement is set in screen input status key-2.

Screen Input Status Key-1	**Screen Input Status Key-2**	**Meaning of Screen Input Status Key-2**
0	0	Operator depressed the End key
	1	Return from last item
1	X”00” to X”FF”	Function key number
2	X”00” to X”FF”	Function key number
9	X”00”	No relevant item on screen

6. The system uses the area of the third character position in data-name-1. This area should not be referenced.

CURRENCY SIGN Clause

This clause specifies the currency symbol.

[Format]

CURRENCY SIGN IS literal-1

Syntax Rules

Literal-1 must not be a symbolic-character or symbolic-constant.

General Rules

1. Specify the currency symbol to be used in the PICTURE clause in literal-1. literal-1 must be a single character nonnumeric literal. The following characters must not be specified in literal-1:

a. Numeric characters: 0 to 9

b. Uppercase alphabetic characters: A B C D E G N P R S V X Z Lowercase alphabetic characters: a to z Space character

c. Special characters: * + - , . ; ( ) / = “

2. When the CURRENCY SIGN clause has been omitted, only the currency symbol in the COBOL character set is regarded as the currency symbol to be used in the PICTURE clause.

CURSOR Clause

This clause specifies a data item for passing the cursor position.

[Format]

CURSOR IS data-name-1

Syntax Rules

1. Data-name-1 must be a data item defined in the working-storage section. There must be four or six character positions in the data item of data-name-1.

2. Data-name-1 must be a group item to which only an unsigned zoned decimal item or a similar elementary item is subordinate.

General Rules

1. In the CURSOR clause, specify a data item for passing the cursor position in the ACCEPT statement of the screen handling module.

2. Set the desired cursor position in data-name-1 before executing the ACCEPT statement.

3. After the ACCEPT statement is executed, the last cursor position set by the user is set in data-name-1.

4. When the CURSOR clause has been omitted, the cursor is positioned at the start of the first input item or update item on the screen before the ACCEPT statement is executed.

5. The value set in data-name-1 and the returned value have the following meanings:

   a. When there are four character positions in the data item in data-name-1, the two higher characters represent the line number and the two lower characters represent the column.

   b. When there are six character positions in the data item in data-name-1, the three higher characters represent the line number and the three lower characters represent the column.

6. If the value set in data-name-1 is not found in the character positions for either the input item or update item on the screen when the ACCEPT statement is executed, the cursor is positioned at the start of the first input item or update item on the screen.

DECIMAL-POINT IS COMMA Clause

This clause replaces the comma and decimal point modules with each other.

[Format]

<u>DECIMAL-POINT</u> IS <u>COMMA</u>

General Rules

Writing DECIMAL-POINT IS COMMA clause causes the comma module and decimal point module in the character-string and numeric literal of the PICTURE clause to be replaced with each other.

POSITIONING UNIT Clause

This clause defines the positioning unit-name.

[Format]

<u>POSITIONING</u> <u>UNIT</u>
positioning-unit-name-1 IS literal-1 CPI

Syntax Rules

Literal-1 must be an unsigned numeric literal from 0.01 to 24.00. The number of decimal positions in literal-1 must not exceed 2.

General Rules

1. In the POSITIONING UNIT clause, affix a name to the positioning unit used when printing at the printing device. That is, affix a name to the unit which is used when specifying a column in the print line. Positioning-unit-name-1 can be specified in the PRINTING POSITION clause of the data description entry. See the section titled “PRINTING POSITION clause,” for more information.

2. Specify the elementary unit of the positioning unit in literal-1. The unit for literal-1 is CPI (characters per inch).

PRINTING MODE Clause

This clause defines the printing mode-name.

[Format]

```
        PRINTING MODE printing-mode-name-1
                    { MOCS }
           IS [FOR  { SOCS }   ]
                    { ALL  }
             { | IN SIZE literal-1 POINT           | }
             { | AT PITCH literal-2  CPI           | }
             { | WITH FONT function-name-1         | }
             { | AT ANGLE integer-1 DEGREES        | }
             { | BY FORM function-name-2           | }
```

Syntax Rules

1. Literal-1 must be an unsigned numeric literal from 3.0 to 300.0 which does not have a sign. The number of decimal positions in literal-1 must not exceed 1.

2. Literal-2 must be an unsigned numeric literal from 0.01 to 24.00. The number of decimal positions in literal-2 must not exceed 2.

3. Function-name-1 must be one of the following:

   MINCHOU: Mincho typeface

   MINCHOU-HANKAKU: Half-size Mincho typeface

   GOTHIC: Gothic typeface

   GOTHIC-HANKAKU: Half-size Gothic typeface

   GOTHIC-DP: Gothic DP

   FONT-nnn: character typeface number established by the system (“nnn” is a typeface number from 001 to 999 established by the system.)

4. Integer-1 must be 0 or 90.

5. Function-name-2 must be one of the following:

   F0102: Full-size long typeface

   F0201: Full-size flat typeface

   F0202: Full-size double-width typeface

   H0102: Half-size long typeface

   H0201: Half-size flat typeface

   H0202: Half-size double-width typeface

   H: Half size(En-size)

   F: Full size

6. When the FOR phrase has been omitted, FOR MOCS is assumed as the default.

7. When the FONT phrase has been omitted, the following are assumed as the default:

a. When printing a National data item or National edited data item, WITH FONT MINCHOU is assumed as the default.

b. When printing an alphabetic item, alphanumeric data item, alphanumeric edited data item, zoned decimal item, external Boolean item, external floating-point data item, or numeric edited data item, WITH FONT GOTHIC-10-LPW is assumed as the default.

8. When the ANGLE phrase has been omitted, AT ANGLE 0 DEGREES is assumed as the default.

9. When the FORM phrase has been omitted, BY FORM F is assumed as the default.

General Rules

1. In the PRINTING MODE clause, affix the name printing mode-name-1 to set the printing mode when printing at the printing device. Printing mode refers to the character size, character pitch, character typeface, character rotation, and character configuration. Printing mode-name-1 can be specified in the CHARACTER TYPE clause.

2. The FOR phrase specifies the item for which printing mode-name-1 is to be set as the printing mode.

   a. When specifying the printing mode for a national data item or a national edited data item, write the FOR MOCS phrase.

   b. When specifying the printing mode for an alphabetic data item, alphanumeric data item, alphanumeric edited data item, zoned decimal item, external Boolean item, external floating-point data item, or numeric edited data item, write the FOR SOCS phrase.

   c. When specifying the printing mode for a national data item, national edited data item, alphabetic data item, alphanumeric data item, alphanumeric edited data item, zoned decimal item, external Boolean item, external floating-point data item, or numeric edited data item, write the FOR ALL phrase.

3. Specify the character size in literal-1 in the SIZE phrase. The unit for the character size is a point.

4. Specify the character pitch in literal-2 in the PITCH phrase. The unit for the character pitch is CPI (characters per inch).

5. When either the SIZE phrase or the PITCH phrase has been omitted, the character size and character pitch are set as follows:

   a. When the PITCH phrase has been omitted, a character pitch appropriate for the value in the SIZE phrase is assumed as the value in the PITCH phrase.

   b. When the SIZE phrase has been omitted, a character size appropriate for the value in the PITCH phrase is assumed as the value in the SIZE phrase.

6. When both the SIZE phrase and PITCH phrase have been omitted, the character size and character pitch are assumed as follows:

   a. When printing an alphabetic item, alphanumeric data item, alphanumeric edited data item, zoned decimal item, external Boolean item, external floating-point data item, or numeric edited data item, the character size is set to 7 points and the character pitch is set to 10 CPI.

   b. When printing a national data item or national edited data item, the character size is set to 12 points and the character pitch is set to a value appropriate for the character size.

7. Specify the character typeface in function-name-1 of the FONT phrase.

8. Specify character rotation by specifying the angle for anticlockwise rotation in integer-1 in the ANGLE phrase. Specifying 0 in integer-1 selects horizontal printing writing. Specifying 90 in integer-1 selects vertical writing.

9. Specify the character configuration in function-name-1 of the FORM phrase.

SYMBOLIC CHARACTERS Clause

This clause defines symbolic-characters.

[Format]

$$\underline{\text{SYMBOLIC}}\ \ \text{CHARACTERS}$$

$$\left\{ \{\text{symbolic-character-1}\}\ \ldots \left\{\begin{matrix}\text{IS}\\ \text{ARE}\end{matrix}\right\}\ \{\text{literal-1}\}\ \ldots \right\}\ldots\ [\underline{\text{IN}}\ \ \text{alphabet-name-1}]$$

Syntax Rules

1. The same symbolic-character cannot be written more than once in symbolic-character-1.

2. When specifying the lists in symbolic-character-1 and integer-1, specify symbolic-character-1 and integer-1 as a pair. That is, the first symbolic-character-1 and the first integer-1 should make a pair and the second symbolic-character-1 and second integer-1 should also make a pair.

3. The number of lists for symbolic-character-1 and the number of lists for integer-1 must be the same.

4. When omitting the IN phrase, integer-1 must be the sequence position of characters in the native character set. When writing the IN phrase, integer-1 must be the sequence position of characters in the character set specified in alphabet-name-1.
5. alphabet-name-1 must not be the alphabet-name associated with the function-name in the ALPHABET clause.

General Rules

1. In a SYMBOLIC CHARACTERS clause which does not contain an IN phrase, affix the name symbolic-character-1 using characters from the native character set which have the sequence position specified in integer-1.
2. In a SYMBOLIC CHARACTERS clause which contains an IN phrase, affix the name symbolic-character-1 using characters from the character set specified in alphabet-name-1 which have the sequence position specified in integer-1.

SYMBOLIC CONSTANT Clause

This clause defines the symbolic-constant.

[Format]

```
SYMBOLIC CONSTANT
 {symbolic-constant-1 IS literal-1 }...
```

Syntax Rules

Literal-1 must not be a symbolic-character or a symbolic-constant.

General Rules

1. In the SYMBOLIC CONSTANT clause, affix the name symbolic-constant-1 in integer-1.

2. symbolic-constant-1 can be written only in these locations:
   a. In the literal of the POSITIONING UNIT clause and the literal and integer of the PRINTING MODE clause in the environment division.
   b. In the integer of the OCCURS clause and the literal of the VALUE clause in the data division.
   c. All locations in which a literal can be written in the procedure division.

Input-Output Section

In the input-output section, write information required for transferring data between an external medium and the object program and processing it.

File Control Paragraph

In the FILE-CONTROL paragraph, associate a file or files with an external medium.

[Format]

```
FILE-CONTROL.
  {file-control-entry}...
```

Several file control entries can be written in a single FILE-CONTROL paragraph. The clauses to be specified in the file control entry and the rules for describing the clauses differ depending on the type of file.

File Control Entries for a Sequential File, Relative File, and Indexed I-O Module

In the file control entries for a sequential file, relative file, and indexed I-O module, define the physical attribute pertaining to each file.

[Format 1] Sequential file

```
 SELECT clause
[ACCESS MODE clause]
 ASSIGN clause
[FILE STATUS clause]
[FORMAT clause]
[GROUP clause]
[LOCK MODE clause]
[ORGANIZATION clause]
[PADDING CHARACTER clause]
[RECORD DELIMITER clause]
[RESERVE clause].
```

[Format 2] Relative file

```
 SELECT clause
[ACCESS MODE clause]
 ASSIGN clause
[FILE STATUS clause]
[LOCK MODE clause]
 ORGANIZATION clause
[RESERVE clause].
```

[Format 3] Indexed file

```
 SELECT clause
[ACCESS MODE clause]
[{ALTERNATE RECORD KEY clause}...]
 ASSIGN clause
[FILE STATUS clause]
[LOCK MODE clause]
 ORGANIZATION clause
 RECORD KEY clause]
[RESERVE clause].
```

Syntax Rules

1. The SELECT clause must be written first in each file control entry. The clauses after the SELECT clause can be written in any sequence.

2. The file-name to be specified in the file description entry of the data division must be defined once in the FILE-CONTROL paragraph of the same program.

General Rules

1. When the file connector of the file to be specified in the SELECT clause is an external file connector, all file control entries in the run unit which reference the file connector must conform to the following rules:

   a. The OPTIONAL phrase in the SELECT clause must either be specified for all file control entries or omitted for all file control entries.

   b. When a value other than data-name is to be specified in the ASSIGN clause, the value must be the same for all file control entries.

   c. The value of the integer to be specified in the RESERVE clause must be the same for all file control entries.

d. The same organization must be specified in the ORGANIZATION clause for all file control entries.

e. The same access mode must be specified in the ACCESS MODE clause for all file control entries.

f. For relative files, the same external data item must be specified as the data-name in the RELATIVE KEY phrase of the ACCESS MODE clause.

g. File control entries for indexed files must also conform to the following rules:

   - The data-names to be specified in the RECORD KEY clause must have the same relative position in records relating to the file-name in the SELECT clause. The same value must be defined in all data description entries for the data-names.

   - The same number of data-names must be specified in the RECORD KEY clause.

   - The DUPLICATES phrase in the RECORD KEY clause must be specified for all indexed files or omitted for all indexed files.

   - The data-names to be specified in the ALTERNATE RECORD KEY clause must have the same relative position in records relating to the file-name in the SELECT clause. The same value must be defined in all data description entries for the data-names.

   - The same number of data-names must be specified in the ALTERNATE RECORD KEY clause.

   - The DUPLICATES phrase in the ALTERNATE RECORD KEY clause must be specified for all indexed files or omitted for all indexed files.

2. The value of the reference key for an indexed file is related according to the collating sequence of the native character set.

File Control Entry for a Sort-merge Module

In the file control entry for a sort-merge module, define the physical attribute pertaining to the sort-merge file.

[Format]

SELECT clause
ASSIGN clause.

Syntax Rules

1. The SELECT clause must be written first in the file control entry, followed by the ASSIGN clause.
2. The file-name to be specified in the file description entry of the data division must be defined once in the FILE-CONTROL paragraph of the same program.
3. The COBOL85 compiler regards an ASSIGN clause for the sort-merge file as a comment.

File Control Entry for a Presentation File Module

In the file control entry for a presentation file module, define the physical attribute pertaining to the presentation file.

[Format]

SELECT clause
[ACCESS MODE clause]
ASSIGN clause
[DESTINATION clause]
[END KEY clause]
[FILE STATUS clause]
[FORMAT clause]

[GROUP clause]
[MESSAGE CLASS clause]
[MESSAGE CODE clause]
[MESSAGE MODE clause]
[MESSAGE OWNER clause]
[MESSAGE SEQUENCE clause]
[ORGANIZATION clause]
[PROCESSING CONTROL clause]
[PROCESSING MODE clause]
[PROCESSING TIME clause]
[SELECTED FUNCTION clause]
[SESSION CONTROL clause]
[SYMBOLIC DESTINATION clause]
[UNIT CONTROL clause].

Syntax Rules

1. The SELECT clause must be written first in the file control entry. The clauses after the SELECT clause can be written in any sequence.

2. The file-name to be specified in the file description entry of the data division must be defined once in the FILE-CONTROL paragraph of the same program.

3. The following table lists destination types and indicates, by system, whether clauses can be specified for the destination types. Meaning: DSP—Display Unit, PRT—Printing Device, ACM—Asynchronous message communication, APL—Inter-program communication, and TRM—Terminal other than display unit and printing device.

Clause	**System**	**Destination Type**				
		DSP	**PRT**	**ACM**	**APL**	**TRM**

Clause	System	Destination Type				
		DSP	PRT	ACM	APL	TRM
DESTINATION-1 clause	DS	○	○	○	○	○
	HP	□	□	□	□	□
	Sun	□	□	○	○	□
	Win32	□	□	○	○	—
	Win16	□	□	○	—	—
DESTINATION-2 clause	DS	○	○	○	○	○
	HP	□	□	□	□	□
	Sun	□	□	○	□	□
	Win32	□	□	■	○	—
	Win16	□	□	■	—	—
DESTINATION-3 clause	DS	○	○	■	○	○
	HP	□	□	■	□	□
	Sun	□	□	■	□	□
	Win32	□	□	■	○	—
	Win16	□	□	■	—	—
END KEY clause	DS	■	■	■	○	○
	HP	■	■	■	□	□
	Sun	■	■	■	□	□
	Win32	■	■	■	○	—
	Win16	■	■	■	—	—
FILE STATUS clause	DS	○	○	○	○	○
	HP	○	○	□	□	□
	Sun	○	○	○	○	○
	Win32	○	○	○	○	—
	Win16	○	○	○	—	—
FORMAT clause	DS	○	○	■	■	○
	HP	○	○	■	■	□
	Sun	○	○	■	■	□
	Win32	○	○	■	■	—
	Win16	○	○	■	—	—
GROUP clause	DS	○	○	■	■	○
	HP	○	○	■	■	□
	Sun	○	○	■	■	□
	Win32	○	○	■	■	—
	Win16	○	○	■	—	—
MESSAGE CLASS clause	DS	■	■	○	■	■
	HP	■	■	□	■	■
	Sun	■	■	□	■	■
	Win32	■	■	○	■	—
	Win16	■	■	○	—	—
MESSAGE CODE clause	DS	■	■	■	○	■
	HP	■	■	■	□	■
	Sun	■	■	■	□	■
	Win32	■	■	■	○	—
	Win16	■	■	■	—	—

Clause	System	Destination Type				
		DSP	PRT	ACM	APL	TRM
MESSAGE MODE clause	DS	○	○	■	○	○
	HP	□	□	■	□	□
	Sun	□	□	■	□	□
	Win32	■	■	■	○	—
	Win16	■	■	■	—	—
MESSAGE OWNER clause	DS	○	■	■	○	○
	HP	□	■	■	□	□
	Sun	□	■	■	□	□
	Win32	■	■	■	○	—
	Win16	■	■	■	—	—
MESSAGE SEQUENCE clause	DS	×	○	×	×	×
	HP	×	□	×	×	×
	Sun	×	□	×	×	×
	Win32	×	□	×	×	—
	Win16	×	□	×	—	—
PROCESSING CONTROL clause	DS	○	○	■	○	○
	HP	□	□	■	□	□
	Sun	□	□	■	□	□
	Win32	□	□	■	○	—
	Win16	□	□	■	—	—
PROCESSING MODE clause	DS	○	○	○	■	○
	HP	○	○	□	■	○
	Sun	○	○	○	■	○
	Win32	○	○	○	■	—
	Win16	○	○	○	—	—
PROCESSING TIME clause	DS	■	■	○	■	■
	HP	■	■	□	■	■
	Sun	■	■	□	■	■
	Win32	■	■	○	■	—
	Win16	■	■	○	—	—
SELECTD FUNCTION clause	DS	○	○	■	■	○
	HP	○	□	■	■	□
	Sun	○	□	■	■	□
	Win32	○	□	■	■	—
	Win16	○	□	■	—	—
SESSION CONTROL clause	DS	○	○	■	○	○
	HP	□	□	■	□	□
	Sun	□	□	■	□	□
	Win32	□	□	■	○	—
	Win16	□	□	■	—	—
UNIT CONTROL clause	DS	○	○	■	■	■
	HP	○	○	■	■	■
	Sun	○	○	■	■	■
	Win32	○	○	■	■	—
	Win16	○	○	■	—	—

○ : The clause can (or must) be specified, and it is functionally significant.
□ : The clause can be specified, but it is not functionally significant. If specified, the clause is treated as a comment.
■ : The clause cannot be specified. If specified, the clause is treated as a comment.
× : The clause cannot be specified.
— : Specification of the clause is significant.

File Control Entry for a Report Writer Module

In the file control entry for a report writer module, define the physical attribute pertaining to the report file.

[Format]

```
 SELECT clause
[ACCESS MODE clause]
 ASSIGN clause
[FILE STATUS clause]
[ORGANIZATION clause]
[RESERVE clause].
```

Syntax Rules

1. The SELECT clause must be written first in the file control entry. The clauses after the SELECT clause can be written in any sequence.
2. The file-name to be specified in the file description entry of the data division must be defined once in the FILE-CONTROL paragraph of the same program. The REPORT clause must be written in the file description entry of the report file.

General Rules

When the file connector of the file to be specified in the SELECT clause is an external file connector, all file control entries in the run unit which reference the file connector must conform to the following rules:

1. The OPTIONAL phrase in the SELECT clause must either be specified for all file control entries or omitted for all file control entries.
2. The file-identifier to be specified in the ASSIGN clause must be the same for all file control entries.
3. The value of the integer to be specified in the RESERVE clause must be the same for all file control entries.
4. The same organization must be specified in the ORGANIZATION clause for all file control entries.
5. The same access mode must be specified in the ACCESS MODE clause for all file control entries.

ACCESS MODE Clause (Sequential File, Relative File, Indexed File, Presentation File, Report Writer)

This clause specifies the access mode for a file.

[Format 1] Sequential file, presentation file, and report writer

ACCESS MODE IS SEQUENTIAL

[Format 2] Relative file

$$\begin{array}{l}\underline{\text{ACCESS}}\ \text{MODE IS}\\ \left\{\begin{array}{l}\underline{\text{SEQUENTIAL}}\ [\underline{\text{RELATIVE}}\ \text{KEY IS data-name-1}]\\ \left\{\begin{array}{l}\underline{\text{RANDOM}}\\ \underline{\text{DYNAMIC}}\end{array}\right\}\ [\underline{\text{RELATIVE}}\ \text{KEY IS data-name-1}]\end{array}\right\}\end{array}$$

[Format 3] Indexed file

$$\underline{\text{ACCESS}}\ \text{MODE IS}\left\{\begin{array}{l}\underline{\text{SEQUENTIAL}}\\ \underline{\text{RANDOM}}\\ \underline{\text{DYNAMIC}}\end{array}\right\}$$

Syntax Rules

RULES FOR FORMAT 2

1. Data-name-1 can be qualified.
2. Data-name-1 must be an unsigned integer item. Do not specify “P” in the PICTURE character-string of the data description entry for data-name-1.
3. Data-name-1 must be defined in the working-storage section or linkage section.
4. When writing the file in the USING phrase or GIVING phrase of the SORT statement or MERGE statement, RANDOM or DYNAMIC must not be written in the ACCESS MODE clause for the file.
5. When writing the file in the START statement, a RELATIVE KEY phrase must be written in the ACCESS MODE clause for the file.

RULES FOR FORMAT 3

When writing the file in the USING phrase or GIVING phrase of the SORT statement or MERGE statement, RANDOM must not be written in the ACCESS MODE clause for the file.

General Rules

RULES COMMON TO FORMAT 1 TO FORMAT 3

1. ACCESS MODE IS SEQUENTIAL specifies that records are accessed in sequential access mode. A file in which ACCESS MODE IS SEQUENTIAL has been specified or a file in which the ACCESS MODE clause has been omitted is called a "sequential access mode file."

   ACCESS MODE IS RANDOM specifies that records are accessed in random access mode. A file in which ACCESS MODE IS RANDOM has been specified is called a "random access mode file."

   ACCESS MODE IS DYNAMIC specifies that records are accessed in dynamic access mode. A file in which ACCESS MODE IS DYNAMIC has been specified is called a "dynamic access mode file."

2. When the ACCESS MODE clause has been omitted, ACCESS MODE IS SEQUENTIAL is assumed as the default.

3. When the relevant file connector is an external file connector, the same access mode must be specified for all file control entry in the run unit which relate to the file connector. Additionally, in Format 2 the same external data item must be specified as data-name-1.

RULES FOR FORMAT 1

Records in the file are accessed in the sequence according to the file organization. The sequence according to the file organization refers to the sequence in which the records were written when the file was created or extended.

RULES FOR FORMAT 2

1. The records in a sequential access mode file are accessed in the sequence according to the file organization. The sequence according to the file organization refers to the ascending order of the relative record numbers of records in the file.
2. In a random access mode file, set the relative record numbers of the records to be accessed in data-name-1.
3. The records in a dynamic access mode file can be accessed in either sequential access mode or random access mode.
4. Records stored in a relative file are each identified by the relative record number. The relative record number indicates logical position of the record in the file. The relative record number of the first record is 1 and the relative record numbers of the following records are in ascending order (2, 3, 4, ...).
5. Use the data item of data-name-1 to link the relative record numbers between the program and the I-O-control system.

RULES FOR FORMAT 3

1. The records in a sequential access mode file are accessed in the sequence according to the file organization. The sequence according to the file organization refers to the ascending order of reference key values according to the collating sequence of the file.
2. In a random access mode file, set the records to be accessed in the prime record key item or the alternate record key item.

3. The records in a dynamic access mode file can be accessed in either sequential access mode or random access mode.

ALTERNATE RECORD KEY Clause (Indexed File)

This clause specifies the alternate record key.

[Format]

```
{ALTERNATE RECORD KEY IS
  {data-name-1}...[WITH DUPLICATES]}...
```

Syntax Rules

1. Data-name-1 can be qualified.
2. Data-name-1 must be defined in the record description entry relating to the file-name specified in the ALTERNATE RECORD KEY clause.
3. Data-name-1 must be one of the following:
   a. Alphanumeric data item
   b. National data item
   c. Unsigned zoned decimal item
   d. Group item which does not contain a variable occurrence data item
4. The leftmost character position in the data item of data-name-1 must not be the same as either of the following character positions:
   a. The leftmost character position in the data item of the prime record key.
   b. The leftmost character position in the data item of any alternate record keys in the same file.

5. When the indexed file contains a variable-length record, the first n characters of the variable-length record must contain the alternate record key. Here, n represents the size of the smallest record in the file. See the section titled “RECORD clause (sequential file, relative file and indexed file),” for more information.

6. Data-name-1 must not be in a variable position in the record.

7. See Appendix B, “System Quantitative Restrictions,” for information on the length of a data item in data-name-1 and the maximum number of data-name-1 descriptions allowed.

General Rules

1. In the ALTERNATE RECORD KEY clause, specify the alternate record key of the file relating to the clause.

2. The following items must be the same as at creation of the file:

   a. Data description entry of data-name-1

   b. Relative position in the record of data-name-1

   c. Number of data-name-1 descriptions

   d. Number of ALTERNATE RECORD KEY clauses

   e. Presence or absence of a DUPLICATES phrase

3. When the DUPLICATES phrase has been written, the file may contain records having equal alternate record key values. When the DUPLICATES phrase has been omitted, the file cannot contain records having equal alternate record key values.

4. When associating two or more record description entries with one file, data-name-1 must be written in only one of the record description entries. The same character position as data-name-1 can be referenced implicitly as a key in any of the other record description entries in which data-name-1 has not been written.

5. When the relevant file connector is an external file connector, all file description entries relating to the file connector in the run unit must conform to the following rules:

    a. Data-name-1 must be in the same relative position in the record for all data description entries.

    b. The contents of the data description entries of data-name-1 must be the same.

    c. The number of data-name-1 descriptions must be the same.

    d. The DUPLICATES phrase must be specified for all data description entries or omitted for all data description entries.

    e. The number of ALTERNATE RECORD KEY clauses must be the same.

6. When two or more data-name-1 descriptions have been written, they are linked in the sequence in which they were written and treated as a single alternate record key.

ASSIGN Clause (Sequential File, Relative File, and Indexed File)

This clause associates a file with an external medium.

[Format 1] Sequential file

$$\underline{\text{ASSIGN}}\ \text{TO}\ \left\{\begin{array}{l}\left\{\begin{array}{l}\text{file-identifier-1}\\ \text{file-identifier-literal-1}\\ \text{data-name-1}\end{array}\right\}\ \ldots\\ \underline{\text{DISK}}\\ \underline{\text{PRINTER}}\end{array}\right\}$$

[Format 2] Relative file and indexed file

$$\underline{\text{ASSIGN}}\ \text{TO}\ \left\{\begin{array}{l}\left\{\begin{array}{l}\text{file-identifier-1}\\ \text{file-identifier-literal-1}\\ \text{data-name-1}\end{array}\right\}\ \ldots\\ \underline{\text{DISK}}\end{array}\right\}$$

Syntax Rules

1. DISK or PRINTER must not be specified as data-name-1.
2. Data-name-1 must be an alphanumeric data item not exceeding 256 characters or a group item. It must be defined in the working-storage section or linkage section.
3. The format of file-identifier-1 is shown below.
   a. Format 1: [comment-]name
   b. Format 2: name

   Specify an external name for reporting files to the system in “name.” “name” must be a character-string not exceeding eight characters and consisting of only alphabetic characters and numeric characters. The first character in “name” must be an alphabetic character.

4. See the section titled “Literal for special applications,” for rules for describing file-identifier-literal-1.
5. DISK and PRINTER cannot be specified in a print file with FORMAT clause.

General Rules

1. In the ASSIGN clause, associate the file-name written in the SELECT clause with a storage medium.
2. When more than one file-identifier-1, file-identifier-literal-1, or data-name-1 descriptions has been written, the second and subsequent descriptions are regarded as comments.
3. In the ASSIGN clause in which file-identifier-1 has been written, associate the file-name written in the SELECT clause with the physical file corresponding to file-identifier-1. Specify the indirect name associated with the physical file on the storage medium in file-identifier-1.
4. In the ASSIGN clause in which file-identifier-literal-1 has been written, associate the file-name written in the SELECT clause with the physical file indicated by the value of file-identifier-literal-1. Directly specify the name of the physical file on the storage medium in file-identifier-literal-1.
5. In the ASSIGN clause in which data-name-1 has been written, associate the file-name written in the SELECT clause with the physical file indicated by the value of data-name-1. When data-name-1 has been specified, the value directly indicating the physical file name on the storage medium must be set in data-name-1 before the OPEN statement can be executed. Then, executing the OPEN statement causes the file-name to be associated with the physical file on the storage medium indicated by the value of data-name-1.

6. An ASSIGN statement in which the DISK phrase has been written indicates that the file-name written in the SELECT clause is a physical file-name on the mass storage unit for the current environment.
7. In the ASSIGN clause in which the PRINTER phrase has been written, associate the file-name written in the SELECT clause with a printing device.
8. Refer to “COBOL85 User's Guide” for information on the relationship between the contents of the ASSIGN clause and the physical file.

ASSIGN Clause (Sort-merge and Report Writer)

This clause associates a file with an external medium.

[Format]

ASSIGN TO {file-identifier-1}...

Syntax Rules

The format of file-identifier-1 is shown below.

[comment-]name

Specify an external name for reporting files to the system in “name.” “name” must be a character-string not exceeding eight characters and consisting of only alphabetic characters and numeric characters. The first character in “name” must be an alphabetic character.

General Rules

1. In the ASSIGN clause, associate the file-name written in the SELECT clause with the storage medium to be referenced according to file-identifier-1.
2. When more than one file-identifier-1 descriptions has been written, the second and subsequent descriptions are regarded as comments.
3. An ASSIGN clause written for a sort-merge file is regarded as a comment.

ASSIGN Clause (Presentation File)

This clause associates a file with an external medium.

[Format]

ASSIGN TO {file-identifier-1} ...

Syntax Rules

The format of file-identifier-1 is shown below.

GS-name

Specify an external name for reporting files to the system in "name." "name" must be a character-string not exceeding eight characters and consisting of only alphabetic characters and numeric characters. The first character in "name" must be an alphabetic character.

General Rules

1. In the ASSIGN clause, associate the file-name written in the SELECT clause with the storage medium to be referenced according to file-identifier-1.
2. When more than one file-identifier-1 description has been written, the second and subsequent descriptions are regarded as comments.

DESTINATION Clause (Presentation File)

The DESTINATION clause specifies a data item to specify or reference a destination-name.

[Format]

```
[DESTINATION-1 IS data-name-1]
[DESTINATION-2 IS data-name-2]
[DESTINATION-3 IS data-name-3]
```

Syntax Rules

1. Data-name-1, data-name-2, and data-name-3 must be eight-character alphanumeric data items. These data items must be defined in the Working-Storage or Linkage Section.
2. Data-name-1, data-name-2, and data-name-3 can be qualified.
3. Data-name-1, data-name-2, and data-name-3 must not be placed in variable locations in a record.

General Rules

1. Data-name-1, data-name-2, and data-name-3 specify message destination names. These destination names are posted to the system when an input-output statement is executed for a presentation file. When a READ statement has been executed successfully, the system sets a destination name.

   The meaning of the DESTINATION clause depends on the system, the input-output statement, and the message type. For details, refer to “COBOL85 User's Guide”.

2. If the DESTINATION clause is omitted, the system posts a space as the destination name.

END KEY Clause (Presentation File)

The END KEY clause specifies a data item to specify or reference segment information.

[Format]

END KEY IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item. This data item must be defined in the Working-Storage or Linkage Section.
2. Data-name-1 can be qualified.
3. Data-name-1 must not be placed in a variable location in a record.

General Rules

1. In the data-name-1 data item, specify segment information (i.e., whether a message is segmented and if so, which segment it is, first, intermediate, or last). When a READ statement has been executed successfully, the system determines the segment information. The segment information is also determined when a WRITE statement is executed.
2. If the END KEY clause is omitted, the system assumes that the message is not segmented when a WRITE statement is executed.

FILE STATUS Clause (Sequential File, Relative File, Indexed File, Presentation File, and Report Writer)

This clause specifies a data item for referencing the I-O status.

[Format]

FILE STATUS IS data-name-1 [data-name-2]

Syntax Rules

1. Data-name-1 and data-name-2 can be qualified.
2. Data-name-1 must be a two-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
3. Data-name-2 must be a four-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
4. Data-name-1 and data-name-2 must not be located in a variable position in a record.

5. Data-name-2 may be written only in the following locations:
   a. The FILE STATUS clause of a print file with FORMAT clause.
   b. The FILE STATUS clause of a presentation file.

General Rules

1. Specify the data-item for referencing the I-O status in data-name-1. Specify the data item for referencing detailed information on the I-O status in data-name-2.
2. Executing an input-output statement for a file specified in the FILE STATUS clause sets the I-O status into data-name-1 and detailed information into data-name-2. Refer to “COBOL85 User's Guide” for information on the values given as the I-O status and detailed information.

FORMAT Clause (Sequential File, Presentation File)

This clause specifies a data item for setting a screen form descriptor.

[Format]

```
FORMAT IS data-name-1
```

Syntax Rules

1. Data-name-1 can be qualified.
2. Data-name-1 must be an eight-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
3. Data-name-1 must not be located in a variable position in a record.

4. When writing a description for a sequential file, the FORMAT clause can be specified only for a print file. A print file for which the FORMAT clause has been specified is called a "print file with FORMAT clause."

General Rules

1. Specify a data item for setting a screen form descriptor in data-name-1. Executing a WRITE statement for a file in which the FORMAT clause has been specified causes the screen form descriptor set in data-name-1 to be reported to the system.
2. When a value other than a space has been set in data-name-1, records are edited in accordance with the screen form descriptor specified by the value.
3. Records are not edited when a space has been set in data-name-1.
4. When writing a description for a presentation file, omitting the FORMAT clause causes spaces to be reported to the system as the screen form descriptor.
5. Refer to "COBOL85 User's Guide" for information on the screen form descriptor.

GROUP Clause (Sequential File, Presentation File)

The GROUP clause specifies a data item as an item group name or an item name.

This clause specifies a data item for setting a partition name or item group name.

[Format]

```
GROUP IS data-name-1
```

Syntax Rules

1. Data-name-1 can be qualified.
2. Data-name-1 must be an eight-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
3. Data-name-1 must not be located in a variable position in a record.
4. A GROUP clause can be specified only for a file in which the FORMAT clause has been specified.

General Rules

1. Set a data item for setting an item name or item group name in data-name-1. Executing an input-output statement causes the item name or item group name specified in data-name-1 to be reported to the system.
2. When a value other than spaces has been set in data-name-1, input or output is performed for the item group specified by the value.
3. When the GROUP clause has been omitted, spaces are reported to the system as the item name or item group name.
4. Refer to “COBOL85 User's Guide” for information on the item name or item group name.

LOCK MODE Clause (Sequential File, Relative File, and Indexed File)

This clause sets the method for locking a file.

[Format 1] Sequential file

```
LOCK  MODE  IS
 ⎧ AUTOMATIC  [WITH  LOCK  ON  RECORD] ⎫
 ⎩ EXCLUSIVE                           ⎭
```

[Format 2] Relative file or indexed file

```
LOCK  MODE  IS
 ⎧ MANUAL  WITH  LOCK  ON  MULTIPLE  RECORDS ⎫
 ⎨ AUTOMATIC  [WITH  LOCK  ON  RECORD]       ⎬
 ⎩ EXCLUSIVE                                 ⎭
```

General Rules

RULES COMMON TO FORMAT 1 AND FORMAT 2

1. When the LOCK MODE clause has been omitted, files which are opened are set in exclusive mode unless the file has been opened in input mode. A file which has been opened in input mode is set in shared mode upon normal execution of an OPEN statement in which WITH LOCK has not been specified.
2. When LOCK MODE IS EXCLUSIVE has been specified, normal execution of an OPEN statement causes the status of the file to indicate that the file was opened in exclusive mode.
3. When LOCK MODE IS AUTOMATIC has been specified, normal execution of an OPEN statement in which WITH LOCK has not been specified causes the status of the file to indicate that the file was opened in shared mode.

4. The WITH LOCK ON RECORD phrase is regarded as a comment.
5. A file opened in exclusive mode cannot be opened by another file connector.
6. A file opened in shared mode can be opened by any other file connector which does not stipulate exclusive mode.
7. A file which does not exist on a mass storage unit cannot be opened in shared mode.
8. Files which are opened in output mode are set in exclusive mode regardless of whether the LOCK MODE clause has been specified.

RULES FOR FORMAT 1

When LOCK MODE IS AUTOMATIC has been specified, normal execution of a READ statement locks the record. The lock on the record is released upon normal execution of an input-output statement for the file connector which locked the record.

RULES FOR FORMAT 2

1. When LOCK MODE IS MANUAL has been specified, normal execution of an OPEN statement in which WITH LOCK has not been specified causes the status of the file to indicate that the file was opened in shared mode.
2. When LOCK MODE IS MANUAL has been specified, the READ WITH LOCK statement locks the record.
3. When LOCK MODE IS AUTOMATIC has been specified, normal execution of a READ statement locks the record. The lock on the record is released upon normal execution of an input-output statement other than a START statement for the file connector which locked the record.

MESSAGE CLASS Clause (Presentation File)

The MESSAGE CLASS clause specifies a data item to indicate the priority of a message in asynchronous message communication.

[Format]

<u>MESSAGE</u> <u>CLASS</u> data-name-1

Syntax Rules

1. Data-name-1 must be a one-digit unsigned external decimal integer item, the description of which is not “P”.
2. Data-name-1 must be defined in the Working-Storage or Linkage Section.
3. Data-name-1 can be qualified.
4. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. In the data-name-1 data item, specify the priority of a message in asynchronous message communication.
2. The value specified for data-name-1 must be in the range of 1 to 9, with 1 as the highest priority.
3. The value of data-name-1 applies when a message is sent using the WRITE statement. The message is given the processing priority specified in this data item.
4. If a message is given a priority when it is sent, the same priority applies when that message is received using the READ statement.

5. Messages with the same priority are received in the order they were sent.

6. If the MESSAGE CLASS clause is omitted, the message is given the lowest priority.

MESSAGE CODE Clause (Presentation File)

The MESSAGE CODE clause specifies a data item to specify or reference a reason code.

[Format]

MESSAGE CODE IS data-name-1

Syntax Rules

1. Data-name-1 must be a data item of 1 to 32,767 characters the, category of which is alphanumeric. This data item must be defined in the Working-Storage or Linkage Section.

2. Data-name-1 can be qualified.

3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. Data-name-1 specifies a reason code to forcibly terminate conversational processing. When a READ statement has been executed successfully, the system determines a reason code. The reason code is posted to the system when a WRITE statement is executed.

2. If the MESSAGE CODE clause is omitted, a space is posted as the reason code.

MESSAGE MODE Clause (Presentation File)

The MESSAGE MODE clause specifies a data item to specify or reference a message type.

[Format]

<u>MESSAGE</u> <u>MODE</u> IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item. This data item must be defined in the Working-Storage or Linkage Section.
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. In data-name-1, specify a message type (e.g., input message, response message, forced message, transmission result notification request, or forced termination of conversational processing). When a READ statement has been executed successfully, the system determines a message type. The message type is posted when a WRITE statement is executed.
2. The characters that can be specified to indicate message types depend on the system.

MESSAGE OWNER Clause (Presentation File)

The MESSAGE OWNER clause specifies a data item to specify or reference send authority information.

[Format]

MESSAGE OWNER IS data-name-1

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item. The data item must be defined in the Working-Storage or Linkage Section.
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. In data-name-1, specify send authority information. When a READ statement has been executed successfully, the system indicates whether the send authority has been granted. When a WRITE statement is executed, the system is notified whether to grant send authority to the program with which conversational processing is performed.
2. If the MESSAGE OWNER clause is omitted, a space is posted to the system as send authority information.

MESSAGE SEQUENCE Clause (Presentation File)

The MESSAGE SEQUENCE clause specifies a data item that contains the generation serial number of forms.

[Format]

```
MESSAGE SEQUENCE IS data-name-1
```

Syntax Rules

1. Data-name-1 must be an eight-digit unsigned zoned decimal integer item, the description of which is not “P”.
2. Data-name-1 must be defined in the Working-Storage or Linkage Section.
3. Data-name-1 can be qualified.
4. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. When a WRITE statement has been executed successfully, the system writes a generation serial number into the data-name-1 data item.
2. If execution of a WRITE statement fails, the contents of data-name-1 are undetermined.

ORGANIZATION Clause (Sequential File)

This clause specifies whether the logical organization of a file is sequential or line sequential.

[Format]

```
[ORGANIZATION IS] [LINE] SEQUENTIAL
```

General Rules

1. The ORGANIZATION IS SEQUENTIAL clause specifies that the logical organization of the file is sequential. The file organization is determined at the creation of the file and cannot be changed later.
2. The ORGANIZATION IS LINE SEQUENTIAL clause specifies that the logical organization of the file is line sequential. The file organization is determined at the creation of the file and cannot be changed later. The records in a line sequential file are delimited by delimiters. Each record can contain only printable characters and a record delimiter. One record is counted as one line.
3. When the ORGANIZATION clause has been omitted, the ORGANIZATION IS SEQUENTIAL clause is assumed as the default.

ORGANIZATION Clause (Relative File)

This clause specifies relative organization as the logical organization of a file.

[Format]

```
[ORGANIZATION IS] RELATIVE
```

General Rules

The ORGANIZATION IS RELATIVE clause specifies that the logical organization of the file is relative. The file organization is determined at the creation of the file and cannot be changed later.

ORGANIZATION Clause (Indexed File)

This clause specifies indexed organization as the logical organization of a file.

[Format]

```
[ORGANIZATION IS] INDEXED
```

General Rules

The ORGANIZATION IS INDEXED clause specifies that the logical organization of the file is indexed. The file organization is determined at the creation of the file and cannot be changed later.

ORGANIZATION Clause (Presentation File and Report Writer)

This clause specifies sequential organization as the logical organization of a file.

[Format]

```
[ORGANIZATION IS] SEQUENTIAL
```

General Rules

1. The ORGANIZATION IS SEQUENTIAL clause specifies that the logical organization of the file is sequential. The file organization is determined at the creation of the file and cannot be changed later.
2. When the ORGANIZATION IS SEQUENTIAL clause has been omitted, this clause is assumed as the default.

PADDING CHARACTER Clause (Sequential File)

This clause specifies a padding character in a block.

[Format]

$$\underline{\text{PADDING}}\ \text{CHARACTER IS}\ \begin{Bmatrix}\text{data-name-1}\\ \text{literal-1}\end{Bmatrix}$$

The COBOL85 compiler regards the PADDING CHARACTER clause as a comment.

PROCESSING CONTROL Clause (Presentation File)

The PROCESSING CONTROL clause specifies a data item to specify or reference extended message control information.

[Format]

$$\underline{\text{PROCESSING}}\ \underline{\text{CONTROL}}\ \text{IS data-name-1}$$

Syntax Rules

1. Data-name-1 must be a data item of up to alphanumeric 128 characters. This data item must be defined in the Working-Storage or Linkage Section.
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. In the data-name-1 data item, specify system-specific message control information. Refer to “COBOL85 User’s Guide” for the format of data-name-1.
2. At the completion of a READ statement, the system writes control information in data-name-1. The value of data-name-1 is posted when a WRITE statement is executed.

PROCESSING MODE Clause (Presentation File)

This clause specifies a data item for setting the processing type.

[Format]

PROCESSING MODE IS data-name-1

Syntax Rules

1. Data-name-1 must be a two-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Specify a data item for setting the processing type in data-name-1. Processing type refers to the type of sound alarm input and erase all screens input-output. Executing an input-output statement for a presentation file causes the processing type set in data-name-1 to be reported to the system.

2. When the PROCESSING MODE clause has been omitted, spaces are reported to the system as the processing type.
3. Refer to “COBOL85 User's Guide” for information on the processing type.

PROCESSING TIME Clause (Presentation File)

The PROCESSING TIME clause specifies a data item to specify the wait time for message transmission and reception in asynchronous message communication.

[Format]

<u>PROCESSING</u> <u>TIME</u> data-name-1

Syntax Rules

1. Data-name-1 must be a four-digit unsigned zoned decimal integer item the description of which is not “P”.
2. Data-name-1 must be defined in the Working-Storage or Linkage Section.
3. Data-name-1 can be qualified.
4. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. In data-name-1, specify the wait time in seconds for message transmission or reception.
2. The value of data-name-1 must be in the range of 0 to 9999. The value 0 indicates an infinite wait time.

3. The value of data-name-1 is only effective in the following statement; message transmission or reception is awaited for the time specified:

   a. READ or WRITE statement with a wait request specified in the processing type

4. If the PROCESSING TIME clause is omitted, the message wait time is infinite.

RECORD DELIMITER Clause (Sequential File)

This clause specifies the method for determining the length of a variable-length record on an external storage medium.

[Format]

```
RECORD DELIMITER IS STANDARD-1
```

The COBOL85 compiler regards the RECORD DELIMITER clause as a comment.

RECORD KEY Clause (Indexed File)

This clause specifies the prime record key.

[Format]

```
RECORD KEY IS {data-name-1} ...
    [WITH DUPLICATES]
```

Syntax Rules

1. Data-name-1 can be qualified.

2. Data-name-1 must be defined in the record description entry relating to the file-name specified in the RECORD KEY clause.

3. Data-name-1 must be one of the following:

   a. Alphanumeric data item

   b. National data item

   c. Unsigned zoned decimal item

   d. Group item which does not contain a variable occurrence data item

4. When the indexed file contains a variable-length record, the first n characters of the variable-length record must contain the prime record key. Here, n represents the size of the smallest record in the file. See the section titled “RECORD clause (sequential file, relative file and indexed file),” for more information.

5. Data-name-1 must not be in a variable position in the record.

6. See Appendix B, “System Quantitative Restrictions,” for information on the length of a data item in data-name-1 and the maximum number of data-name-1 descriptions allowed.

General Rules

1. In the RECORD KEY clause, specify the prime record key of the file relating to the clause.

2. When the DUPLICATES phrase has been written, the value of the prime record key need not be unique for each record in the file. When the DUPLICATES phrase has been omitted, the value of the prime record key must be unique for each record in the file.

3. The following items must be the same as at creation of the file:

   a. Data description entry of data-name-1

   b. Relative position in the record of data-name-1

   c. Number of data-name-1 descriptions

   d. DUPLICATES phrase

4. When associating two or more record description entries with one file, data-name-1 must be written in only one of the record description entries. The same character position as data-name-1 can be referenced implicitly as a key in any of the other record description entries in which data-name-1 has not been written.

5. When the relevant file connector is an external file connector, all file description entries relating to the file connector in the run unit must conform to the following rules:

   a. Data-name-1 must be in the same relative position in the record for all data description entries.

   b. The contents of the data description entries of data-name-1 must be the same.

   c. The number of data-name-1 descriptions must be the same.

   d. The DUPLICATES phrase must be specified for all data description entries or omitted for all data description entries.

6. When multiple data-name-1 descriptions have been written, they are linked in the sequence in which they were written and treated as a single prime record key.

RESERVE Clause (Sequential File, Relative File, Indexed File, and Report Writer)

This clause specifies the number of input-output areas.

[Format]

$$\underline{\text{RESERVE}}\ \ \text{integer-1}\ \ \begin{bmatrix}\text{AREA}\\ \text{AREAS}\end{bmatrix}$$

The COBOL85 compiler regards the RESERVE clause as a comment.

SELECT Clause (Sequential File, Relative File, Indexed File, and Report Writer)

This clause declares a file-name.

[Format]

$$\underline{\text{SELECT}}\ [\underline{\text{OPTIONAL}}]\ \text{file-name-1}$$

Syntax Rules

1. The SELECT clause must be written first in the file description entry.
2. The file description entry of file-name-1 must be written in a program in which the SELECT clause has been written.

General Rules

1. The OPTIONAL phrase specifies that the file of file-name-1 does not necessarily exist. A file in which the OPTIONAL phrase has been written is called an “optional file.”

2. The OPTIONAL phrase can be specified for the following files only:
    a. A file to be opened in input mode, I-O mode or extend mode when writing a description for a sequential file, relative file, or indexed file.
    b. A file to be opened in extend mode when writing a description for a report file.

SELECT Clause (Sort-merge and Presentation File)

This clause declares a file-name.

[Format]

<u>SELECT</u> file-name-1

Syntax Rules

1. The SELECT clause must be written first in the file description entry.
2. The file description entry of file-name-1 must be written in a program in which the SELECT clause has been written.

SELECTED FUNCTION Clause (Presentation File)

This clause specifies a data item for referencing the attention type.

[Format]

SELECTED <u>FUNCTION</u> IS data-name-1

Syntax Rules

1. Data-name-1 must be a four-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Specify a data item for referencing the attention type in data-name-1. Normal execution of a READ statement causes the attention type to be reported from the system and set in data-name-1.
2. The following values can be set as the attention key in data-name-1.
   a. A key which can be altered when the screen is defined (such as a Function key)
   b. ENTER key
   c. Item select
3. Refer to “COBOL85 User’s Guide” for information on the attention type.

SESSION CONTROL Clause (Presentation File)

The SESSION CONTROL clause specifies a data item to specify or reference session information.

[Format]

```
SESSION CONTROL IS data-name-1
```

Syntax Rules

1. Data-name-1 must be a one-character alphanumeric data item. The data item must be defined in the Working-Storage or Linkage Section.
2. Data-name-1 can be qualified.
3. The data-name-1 data item must not be placed in a variable location in a record.

General Rules

1. In data-name-1, specify the session information. When a READ statement has been executed successfully, the system determines the session information. The session information is posted when a WRITE statement is executed.
2. Depending on the value of data-name-1, one of the following specifications is provided as session information:
   a. Start of conversational processing
   b. End of conversational processing
   c. Continuation of conversational processing
3. If the SESSION CONTROL clause is omitted, the continuation of conversational processing is posted when a WRITE statement is executed.

SYMBOLIC DESTINATION Clause (Presentation File)

This clause specifies the destination type.

[Format]

$$\text{SYMBOLIC } \underline{\text{DESTINATION}} \text{ IS } \left\{ \begin{array}{l} \text{data-name-1} \\ \\ \text{literal-1} \end{array} \right\}$$

Syntax Rules

1. Data-name-1 must be a three-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Specify a data item for setting and referencing the destination type in data-name-1. Set the destination type in literal-1. Executing an input-output statement causes the destination type set in data-name-1 or literal-1 to be reported to the system.
2. The following table lists the values which can be set in data-name-1 and literal-1.

Value in data-name-1 or literal-1	**Destination Type**
DSP	Display unit
PRT	Printing device
ACM **DS** **Sun** **Win**	Asynchronous message communication
APL **DS** **Sun**	Inter-program communication
TRM **DS** **Sun**	Terminal other than display unit and printing device

3. When the SYMBOLIC DESTINATION clause has been omitted, display is reported to the system as the default destination type.

4. When data-name-1 has been written, executing a READ statement causes the destination type to be reported from the system and set in data-name-1.

UNIT CONTROL Clause (Presentation File)

This clause specifies a data item for setting unit control information.

[Format]

```
UNIT CONTROL IS data-name-1
```

Syntax Rules

1. Data-name-1 must be a six-character alphanumeric data item or a group item. It must be defined in the working-storage section or the linkage section.
2. Data-name-1 can be qualified.
3. Data-name-1 must not be located in a variable position in a record.

General Rules

1. Specify a data item for setting unit control information which performs special control according to the unit in data-name-1. Executing a WRITE statement causes the unit control information set in data-name-1 to be reported to the system.
2. Refer to “COBOL85 User's Guide” for information on the unit control information.

I-O-Control Paragraph

This paragraph specifies the special control technique used by the object program.

[Format]

```
I-O-CONTROL.
          [ [APPLY { MULTICONVERSATION-MODE } clause]
                   { MULTICON               }
          [APPLY SAVED-AREA clause]
          [{MULTIPLE FILE TAPE clause} ...]
          [{RERUN clause } ...]
          [{SAME clause} ...].]
```

Syntax Rules

1. The clauses can be written in any order.
2. Files having different organizations can be written in one clause. However, each clause allows files of certain organization types only to be written in it. The table below shows the clauses that can be written in the I-O-control paragraph.

Clause in I-O-control paragraph	**Sequential file**	**Relative file**	**Indexed file**	**Sort-merge file**	**Presentation file**	**Report file**
APPLY MULTICON clause	×	×	×	×	○	×
APPLY SAVED-AREA clause	×	×	×	×	○	×
MULTIPLE FILE TAPE clause	□	×	×	×	×	□
RERUN clause	□	□	□	×	×	×
SAME clause	○	○	○	○	○	○

○ : A file of the organization type indicate can be written.
× : A file of the organization type indicate cannot be written.
□ : A file of the organization type indicate can be written but it is regarded as a comment.

General Rules

The COBOL85 compiler regards the MULTIPLE FILE TAPE clause and RERUN clause as comments.

APPLY MULTICONVERSATION-MODE Clause (Presentation File)

The APPLY MULTICONVERSATION-MODE clause indicates that multiple conversational processing is performed for two or more destinations.

[Format]

$$\underline{\text{APPLY}} \left\{ \begin{array}{l} \underline{\text{MULTICONVERSATION-MODE}} \\ \underline{\text{MULTICON}} \end{array} \right\}$$

Syntax Rules

1. MULTICONVERSATION-MODE and MULTICON are synonymous.
2. The APPLY MULTICONVERSATION-MODE clause cannot be written in an internal program.

General Rules

1. The APPLY MULTICONVERSATION-MODE clause specifies that multiple conversational processing is performed for two or more destinations. If the APPLY MULTICONVERSATION-MODE clause is specified, the system automatically performs the steps of the program execution flow in conversational processing with multiple destinations. Thus, input-output statements can be written in the same way that they are written for processing for a single destination.
2. If the APPLY MULTICONVERSATION-MODE clause is specified, the APPLY SAVED-AREA clause must also be specified.
3. The APPLY MULTICONVERSATION-MODE clause is valid for all presentation files defined in the program containing the clause and in its internal programs.

APPLY SAVED-AREA Clause (Presentation File)

This clause guarantees the value of a data item over several conversations.

[Format]

APPLY SAVED-AREA TO {data-name-1}...

Syntax Rules

1. Data-name-1 must be a fixed-length data item of level-number 01 or 77. It must be defined in the working-storage section.
2. A VALUE clause must not be specified in data-name-1 or a data item subordinate to data-name-1.
3. An APPLY SAVED-AREA clause cannot be written in an internal program.

General Rules

1. Write the APPLY SAVED-AREA clause to guarantee the value of data-name-1 over several conversations.
2. The APPLY SAVED-AREA clause is effective for all presentation files defined in the program in which the clause was written and in its internal programs.

MULTIPLE FILE TAPE Clause (Sequential File and Report Writer)

This clause specifies the position of a file on a multiple file tape. The MULTIPLE FILE TAPE clause is an obsolete language element.

[Format]

$$\begin{array}{l}\underline{\text{MULTIPLE}}\ \underline{\text{FILE}}\ \text{TAPE CONTAINS}\\ \quad\{\text{file-name-1}\ [\underline{\text{POSITION}}\ \text{integer-1}]\}\ldots\end{array}$$

The COBOL85 compiler regards the MULTIPLE FILE TAPE clause as a comment.

RERUN Clause (Sequential File, Relative File and Indexed File)

This clause specifies the time at which rerun starts. The RERUN clause is an obsolete language element.

[Format]

$$\begin{array}{l}\underline{\text{RERUN}}\ \ \underline{\text{ON}}\ \ \text{file-identifier-1 EVERY}\\ \left\{\begin{array}{l}[\underline{\text{END}}\ \ \text{OF}]\left\{\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right\}\\ \text{integer-1}\ \ \underline{\text{RECORDS}}\end{array}\right\}\ \ \text{OF file-name-1}\end{array}$$

The COBOL85 compiler regards the RERUN clause as a comment.

SAME Clause (Sequential File, Relative File, Indexed File, Sort-merge, Presentation File, and Report Writer)

This clause specifies sharing of files in the storage area.

[Format 1] Sharing files (SAME AREA clause)

SAME AREA FOR file-name-1 {file-name-2} ...

[Format 2] Sharing records (SAME RECORD AREA clause)

SAME RECORD AREA FOR file-name-1 {file-name-2} ...

[Format 3] Allocating suitable sort-merge file storage areas (SAME SORT / SORT-MERGE AREA clause)

$$\underline{\text{SAME}} \left\{ \begin{array}{l} \underline{\text{SORT}} \\ \underline{\text{SORT-MERGE}} \end{array} \right\} \text{AREA}$$

FOR file-name-1 { file-name-2} ...

Syntax Rules

1. File-name-1 and file-name-2 must be written in the file-control paragraph of the same program.
2. The file connectors of file-name-1 and file-name-2 must not be external file connectors.
3. File-name-1 and file-name-2 must be files having one of the following organizations:
   a. In Format 1, they must be a sequential file, relative file, indexed file, presentation file, or report file.
   b. In Format 2, they must be a sequential file, relative file, indexed file, presentation file, or sort-merge file.
   c. In Format 3, they must be a sort-merge file, sequential file, relative file, or indexed file.

4. File-name-1 and file-name-2 need not have the same organization or same access mode.

5. SORT and SORT-MERGE are synonymous.

6. Two or more SAME clauses can be written in one program as long as they conform to the following rules:

   a. A file-name in a SAME AREA clause must not be repeated in another SAME AREA clause.

   b. A file-name in a SAME RECORD AREA clause must not be repeated in another SAME RECORD AREA clause.

   c. When writing a file-name which has been written in the SAME AREA clause in the SAME RECORD AREA clause also, the other file-names which have been written in the SAME AREA clause must all be written in the same SAME RECORD AREA clause. However, a file-name which has not been written in the SAME AREA clause can be written in any SAME RECORD AREA clause.

   d. A sort-merge file-name in a SAME SORT/SORT-MERGE AREA clause must not be repeated in another SAME SORT/SORT-MERGE AREA clause.

   e. When a file-name for a file which is not a sort-merge file has been written in both the SAME AREA clause and the SAME SORT/SORT-MERGE AREA clause, the other file-names in the SAME AREA clause must all be written in the same SAME SORT/SORT-MERGE AREA clause.

7. A file for which RECORD CONTAINS 0 has been written in the file description entry must not be written in the SAME AREA clause or SAME RECORD AREA clause.

General Rules

RULES FOR FORMAT 1

The SAME AREA clause specifies that two or more files written in the clause share the same storage area during processing. The entire portion of the storage area allocated to the files written in the clause is shared. Only one of the files written in the clause can be open at a time.

RULES FOR FORMAT 2

The SAME RECORD AREA clause specifies that files written in the clause share the same storage area to process current records. An area which is shared by several records is called a “shared area.” The files written in this clause can be open at the same time. The records in the shared area are used as records for those files which have been opened in output mode and as recently read records for those files which have been opened in input mode. This allocation is the same as redefining the record area for a file which has been opened in output mode in the record area of a file which has been opened in input mode. Both types of records are aligned at the left character position.

RULES FOR FORMAT 3

The SAME SORT/SORT-MERGE AREA clause specifies that the storage area for the SORT statement and MERGE statement is to be allocated as considered appropriate. At least one of the files to be written in this clause must be a sort-merge file.

The COBOL85 compiler allows the system to allocate the storage area, so the SAME SORT/SORT-MERGE AREA clause is regarded as a comment.

Chapter 5. Data Division

The data division defines the data to be processed by an object program. Data processed by object programs includes the following:

- File data entered from external storage and output to external storage or printing device
- File data for sort and merge
- File data for preparing a report
- Data displayed on and entered from a display unit
- Data specified by users

Data Division Configuration

The data division follows the environment division. The data division can be omitted.

The data division has seven sections: based-storage section, file section, working-storage section, constant section, linkage section, report section, and screen section

The data division configuration is shown below. Clauses and entries of the data division must be written in the following order.

[Format]

```
     DATA DIVISION.
   ⎡ BASED-STRAGE SECTION.            ⎤                                   ⎫ Based-Storage  ⎫
   ⎢  ⎡77-level description entry⎤ ... ⎥                                   ⎬ Section        ⎪
   ⎣  ⎣record description entry  ⎦     ⎦                                   ⎭                ⎪
   ⎡ FILE SECTION.                                                      ⎤ ⎫                ⎪
   ⎢  ⎡file description entry [record description entry] ...          ⎤ ... ⎥ ⎬ File Section   ⎪
   ⎣  ⎣sort-merge file description entry {record description entry} ...⎦     ⎦ ⎭                ⎪
   ⎡ WORKING-STORAGE SECTION .        ⎤                                   ⎫ Working-Storage⎪
   ⎢  ⎡77-level description entry⎤ ... ⎥                                   ⎬ Section        ⎪
   ⎣  ⎣record description entry  ⎦     ⎦                                   ⎭                ⎪
   ⎡ CONSTANT SECTION .               ⎤                                   ⎫ Constant       ⎬ Data
   ⎢  ⎡77-level description entry⎤ ... ⎥                                   ⎬ Section        ⎪ Division
   ⎣  ⎣record description entry  ⎦     ⎦                                   ⎭                ⎪
   ⎡ LINKAGE SECTION .                ⎤                                   ⎫ Linkage        ⎪
   ⎢  ⎡77-level description entry⎤ ... ⎥                                   ⎬ Section        ⎪
   ⎣  ⎣record description entry  ⎦     ⎦                                   ⎭                ⎪
   ⎡ REPORT SECTION .                                                  ⎤ ⎫                ⎪
   ⎣  [ report description entry {report group description entry} ...] ...⎦ ⎭ Report Section ⎪
   ⎡ SCREEN SECTION .                     ⎤                               ⎫                ⎪
   ⎣  [ screen data description entry] ... ⎦                               ⎭ Screen Section ⎭
```

Based-storage Section

The based-storage section defines other languages, other programs, the data for manipulating data in areas secured in other sections of the same program, and also the data structure of records.

Data items defined in the based-storage section must be referenced by pointer with pointer data items.

The section header (BASED-STORAGE SECTION) of the based-storage section is followed by 77-level description entries or record description entries.

File Section

The file section defines the file structure of sequential, relative, indexed, sort-merge, and presentation files, and the data structure of records. The file section also defines the file structure of report files.

The section header (FILE SECTION) of the file section is followed by the following entries. Files defined in the file section must be associated with the external media by file control entry in the environment division.

- When sequential file module, relative file module, indexed file module, or presentation file module is used, one file description entry is written for each file, followed by one or more record description entries. The record description entries must directly follow the file description entry.

- When a sort-merge module is used, one sort-merge file description entry is written for each sort-merge file, followed by one or more record description entries. The record description entries must directly follow the sort-merge file description entry.
- When a report writer module is used, one file description entry is written for each report file. The file section of the report files does not define the data structure of the records. The data structure of the records of the report files is defined in the report section of the data division.

Working-storage Section

The working-storage section defines the data structure of the data and records used in the object program.

The section header (WORKING-STORAGE SECTION) of the working-storage section is followed by 77-level description entries or record description entries.

Constant Section

The constant section defines constants used in the object program. Values specified in the constant section do not change during program execution.

The section header (CONSTANT SECTION) of the constant section is followed by 77-level description entries or record description entries.

Linkage Section

The linkage section defines the data referenced by both calling programs and called programs. The linkage section is written in called programs. The data items written in the USING phrase in the procedure division header or the ENTRY statement must be defined in the linkage section.

The section header (LINKAGE SECTION) of the linkage section is followed by 77-level description entries or record description entries.

Report Section

The report section defines report output to a report file. When a report writer module is used, the report section must be written. The report defined in the report section must be associated with the report file in the file section of the data division.

The section header (REPORT SECTION) of the report section is followed by report description entries or report group description entries. One report description entry is written for each report, followed by one or more report group description entries. The report group description entries must directly follow the report description entry.

Screen Section

The screen section defines the screen structure, and the data displayed on the screen or the data entered from the screen. When a screen handling module is used, the screen section must be written.

The section header (SCREEN SECTION) of the screen section is followed by one or more screen data description entries.

77-level Description Entry

Items that are not interdependent and that cannot be classified further can be defined as 77-level description entries. A 77-level description entry is a data description entry of level-number 77. All of the following clauses must be written in the 77-level description entry:

- Level-number 77
- Data-name
- The PICTURE clause, and USAGE IS INDEX clause, USAGE IS COMP-1 clause, USAGE IS COMP-2 clause, or USAGE IS POINTER clause.

Record Description Entry

An entire set of data description entries related to a record is called “record description entry.” Data which is interdependent is defined in a record description entry together. One record description entry defines several group items and elementary items. When data which is not interdependent is defined, one record description entry can also define only one elementary item.

The record configuration is indicated by level-number. Level-numbers from 01 to 49 are assigned to all data items which depend on a record. A record is the most comprehensive data item, so level-numbers of records are assigned starting from 01. Level-numbers of data items lower than those of level-number 01 do not need to be consecutive.

Data description entries of data items which depend on a group item follow data description entries of the group item. A group item includes all data items until appearance of a data description entry having level-number equal to or less than the level-number of the group item. All level-numbers of data items which depend on a group item must be the same or larger than the level-number of the group item.

File Description Entry

A file description entry defines the physical file structure. Write a file description entry in the file section or the report section.

[Format 1] Sequential file

```
FD  file-name-1
  [BLOCK CONTAINS clause]
  [CODE-SET clause]
  [CONTROL RECORDS clause]
  [DATA RECORDS clause]
  [EXTERNAL clause]
  [GLOBAL clause]
  [LABEL RECORDS clause]
  [LINAGE clause]
  [RECORD clause]
  [VALUE OF clause].
```

[Format 2] Relative file / indexed file

```
FD file-name-1
  [BLOCK CONTAINS clause]
  [DATA RECORDS clause]
  [EXTERNAL clause]
  [GLOBAL clause]
  [LABEL RECORDS clause]
  [RECORD clause]
  [VALUE OF clause].
```

[Format 3] Presentation file

```
FD file-name-1
  [EXTERNAL clause]
  [GLOBAL clause]
  [RECORD clause].
```

[Format 4] Report file (file used in report writer module)

```
FD file-name-1
  [BLOCK CONTAINS clause]
  [CODE-SET clause]
  [EXTERNAL clause]
  [GLOBAL clause]
  [LABEL RECORDS clause]
  [RECORD clause]
  [REPORT clause]
  [VALUE OF clause].
```

Syntax Rules

1. Level indicator FD must be written at the beginning of a file description entry, followed by file-name-1. The order of clauses which follow file-name-1 is optional.
2. For a non-report file, the file description entry must be followed by at least one record description entry. For a report file, the file description entry must not be followed by a record description entry.
3. For a report file, file-name-1 must be a sequential file.

General Rules

1. A file description entry relates file-name-1 to a file connector.
2. The DATA RECORDS, LABEL RECORDS, and VALUE OF clauses are obsolete language elements.
3. This compiler regards the BLOCK CONTAINS, CODE SET, DATA RECORDS, LABEL RECORDS, and VALUE OF clauses as comments.

BLOCK CONTAINS Clause (Sequential File, Relative File, Indexed File, and Record Writer Module)

The BLOCK CONTAINS clause specifies the block size.

[Format]

$$\underline{\text{BLOCK}}\ \text{CONTAINS}$$

$$[\text{interger-1}\ \ \underline{\text{TO}}]\ \ \text{interger-2} \left\{\begin{array}{l}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{array}\right\}$$

General Rule

This compiler regards the BLOCK CONTAINS clause as a comment.

CODE-SET Clause (Sequential File and Report Writer Module)

The CODE-SET clause specifies the character code convention used for data expression on the external storage data medium.

[Format]

```
CODE-SET IS code-name-1
```

Syntax Rules

1. The CODE-SET clause can be written only when both of the following conditions are satisfied:
   a. All data items included in a record description entry related to a file are used for display.
   b. The SIGN IS SEPARATE clause is specified in all signed numeric data items.
2. code-name-1 must not be one associated with a literal by the ALPHABET clause in the special-names paragraph.
3. The CODE-SET clause cannot be specified for a print file.

General Rule

This compiler regards the CODE-SET clause as a comment.

CONTROL RECORDS Clause (Sequential File)

The CONTROL RECORDS clause specifies a name of control record in a file.

[Format]

$$\underline{\text{CONTROL}} \quad \left\{ \begin{array}{l} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{array} \right\} \{\text{data-name-1}\}\ldots$$

Syntax Rules

1. data-name-1 must be a data-name of level-number 01 in a record description entry associated with a file.
2. data-name-1 must be unique in a record description entry associated with a file.
3. data-name-1 can be qualified.
4. The CONTROL RECORDS clause can be specified only for a print file with the FORMAT clause.

General Rules

1. When the record specified by data-name-1 is written with the WRITE statement, it is handled as a control record. A control record posts the record editing method to the system.
2. A control record associated with a file shares the record area with other records associated with the file.

DATA RECORDS Clause (Sequential File, Relative File, and Indexed File)

The DATA RECORDS clause lists names of data records.

The DATA RECORDS clause is an obsolete language element.

[Format]

$$\underline{\text{DATA}} \quad \left\{ \begin{array}{l} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{array} \right\} \ \{\text{data-name-1}\}\ldots$$

Syntax Rule

data-name-1 must be a data-name of level-number 01 in a record description entry related to a file.

General Rule

This compiler regards the DATA RECORDS clause as a comment.

EXTERNAL Clause (Sequential File, Relative File, Indexed File, Presentation File, and Report Writer Module)

The EXTERNAL clause gives external attribute to a file connector.

[Format]

IS EXTERNAL

General Rule

The EXTERNAL clause specifies that the file connector of the associated file is the external file connector. The external file connector in the run unit is associated with a file-name. The external file connector of the same file-name references the same record area.

GLOBAL Clause (Sequential File, Relative File, Indexed File, Presentation File, and Report Writer Module)

The GLOBAL clause specifies that a file-name is a global name.

[Format]

IS GLOBAL

Syntax Rule

The GLOBAL statement must not be written in the file description entry associated with the file specified in the SAME RECORD AREA clause.

General Rule

The GLOBAL clause specifies that the related file-name is a global name.

LABEL RECORDS Clause (Sequential File, Relative File, Indexed File, and Report Writer File)

The LABEL RECORDS clause specifies existence / non-existence of a file label. The LABEL RECORDS clause is an obsolete language element.

[Format]

$$\underline{\text{LABEL}} \left\{ \begin{array}{l} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \end{array} \right\}$$

General Rule

This compiler regards the LABEL RECORDS clause as a comment.

LINAGE Clause (Sequential File)

The LINAGE clause defines the configuration of logical pages.

[Format]

```
LINAGE IS { data-name-1 } LINES
          { integer-1   }

   [ WTTH FOOTING AT { data-name-2 } ]
   [                 { integer-2   } ]

   [ LINES AT TOP     { data-name-3 } ]
   [                  { integer-3   } ]

   [ LINES AT BOTTOM  { data-name-4 } ]
   [                  { integer-4   } ]
```

Syntax Rules

1. data-name-1 to data-name-4 must be unsigned integer items.
2. data-name-1 to data-name-4 can be qualified.
3. integer-2 must be equal to or less than integer-1.
4. Zero can be specified in integer-3 and integer-4.
5. The LINAGE clause can be specified only in a print file without the FORMAT clause.
6. For the maximum number of lines to make up a logical page, see Appendix B, “System Quantitative Restrictions.”

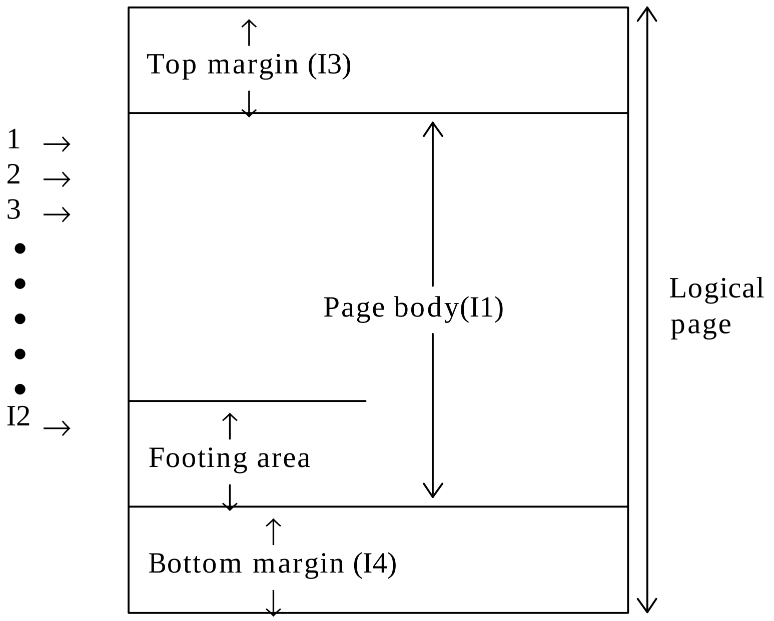
General Rules

1. Specification of the LINAGE clause and the corresponding logical page configuration are indicated below.

[Specification of the LINAGE clause]

```
LINAGE IS I1 LINES
    WITH FOOTING AT I2
    LINES AT TOP I3
    LINES AT BOTTOM I4
```

[Layout of a logical page]



The I1 line starting from the line below the top margin is called “page body.”

The I2 line from the first line of the page body to the end line of the page body is called “footing area.”

2. The size of a logical page is the total of the values written in each phrase other than the FOOTING phrase. If the LINES AT TOP phrase is omitted, “LINES AT TOP 0” is assumed. In the same way, if the LINES AT BOTTOM phrase is omitted, “LINES AT BOTTOM 0” is assumed.

3. If the FOOTING phrase is omitted, the end-of-page condition does not occur independently of the page overflow condition. There is no need to specify the size of a logical page according to the size of a physical page.

4. integer-1 or data-name-1 specifies the number of lines written or fed in a logical page. This value must be positive.

5. integer-2 or data-name-2 specifies the line position to start the footing area of the page body. The first line of the page body is 1. The value must be positive and equal to or less than integer-1 or data-name-1.

6. integer-3 or data-name-3 specifies the number of lines of the top margin of a logical page. Zero can be specified.

7. integer-4 or data-name-4 specifies the number of lines of the bottom margin of a logical page. Zero can be specified.

8. If integer-1, integer-3, and integer-4 are specified, the number of lines to make up each part of a logical page is determined according to the values when the OPEN OUTPUT statement is executed. If integer-2 is specified, the footing area is determined according to the value when the OPEN OUTPUT statement is executed. These integer values are applied to all logical pages output to the file which specified the LINAGE clause during the program execution.

9. If data-name-1, data-name-3, and data-name-4 are specified, the number of lines to make up each part of a logical page is determined as follows:

    a. When the OPEN OUTPUT statement is executed, the number of lines to make up each part of the first logical page is determined according to these data items.

    b. When the WRITE statement with the ADVANCING PAGE phrase is executed, or the page overflow condition occurs, the number of lines to make up each part of the next logical page is determined according to these data items.

10. If data-name-2 is specified, the number of lines to make up the footing area of the first logical page is determined according to the value of data-name-2 when the OPEN OUTPUT statement is executed. When the WRITE statement with the ADVANCING PAGE phrase is executed, or the page overflow condition occurs, the footing area of the next logical page is determined according to the value of this data item.

11. If the LINAGE clause is written, special register LINAGE-COUNTER is automatically generated. A value indicating the current line position of the page body of the logical page is always specified to the LINAGE-COUNTER. Rules for the LINAGE-COUNTER are listed below.

    a. One LINAGE-COUNTER is generated per file which specified the LINAGE clause.

    b. THE VALUE OF the LINAGE-COUNTER can be referenced by, but not specified by, a statement of the procedure division. The i-o-control system specifies a value in the LINAGE-COUNTER. If two or more files which specified the LINAGE clause are defined, the LINAGE-COUNTER must be qualified by a file-name.

c. THE VALUE OF the LINAGE-COUNTER automatically changes according to the following rules during the execution of the WRITE statement for the file which specified the LINAGE clause.

   If the WRITE statement with the ADVANCING PAGE phase is executed, the value of the LINAGE-COUNTER is set to 1.

   If the WRITE statement specified the ADVANCING phrase with an integer or identifier is executed, the value of the LINAGE-COUNTER is incremented by the value of the integer or identifier specified in the ADVANCING phrase.

   If the WRITE statement without the ADVANCING phrase is executed, the value of the LINAGE-COUNTER is incremented by one.

   If the value of the LINAGE-COUNTER comes to the first line position of the next logical page, it is set to 1.

d. If the OPEN OUTPUT statement is executed for the file which specified the LINAGE clause, the value of the LINAGE-COUNTER is set to 1.

12. If a file connector associated with the file which specified the LINAGE clause is an external file connector, all file description entries associated with the file connector in the run unit must follow the following rules:

    a. The LINAGE clause must be written in all file description entries associated with the file connector in the run unit.

    b. When integer-1 to integer-4 are written, the same value must be specified to all of the corresponding integers.

    c. When data-name-1 to data-name-4 are written, these data-name must be external data items. Also, all of the corresponding data-names must be the same.

Record Clause (Sequential File, Relative File, and Indexed File)

The RECORD clause specifies the size and format of records.

[Format 1] The size of a fixed length record is specified.

RECORD CONTAINS integer-1 CHARACTERS

[Format 2] The size of a variable-length record is specified.

```
RECORD IS VARYING IN SIZE
     [[FROM integer-2] [TO integer-3] CHARACTERS]
     [DEPENDING ON data-name-1]
```

[Format 3] The size and format of a record is specified in a record description entry.

```
RECORD CONTAINS
integer-4 TO integer-5 CHARACTERS
```

Syntax Rules

Rules for Format 1

A record having more than integer-1 character position must not be defined in the record description entry associated with a file which specified a RECORD clause of format 1.

Rules for Format 2

1. A record having fewer than integer-2 character positions must not be defined in the record description entry associated with a file which specified a RECORD clause of format 2. Also, a record having more than integer-3 character positions must not be defined.

2. integer-3 must be greater than integer-2.
3. data-name-1 must be an unsigned integer item defined in the working-storage section or the linkage section.
4. The DEPENDING ON phrase cannot be written in any of the following conditions:

   a. When the CHARACTER TYPE clause or the PRINTING POSITION clause is written in the record description entry associated with a file

   b. When the record associated with a file is written in the WRITE statement with the FROM phrase and one of the following data items is specified in the FROM phrase:

      - Data item which specified the CHARACTER TYPE clause or the PRINTING POSITION clause
      - Data item depending on the data item which specified the CHARACTER TYPE clause or the PRINTING POSITION clause
      - Data item possessing the data item which specified the CHARACTER TYPE clause or the PRINTING POSITION clause.

Rules for Format 3

integer-5 must be greater than integer-4.

General Rules

Rules common to format 3 and when the RECORD clause is omitted.

1. If a RECORD clause of format 3 is written or the RECORD clause is omitted, the size of a data record is determined according to the description of the record description entry.

2. If a file connector associated with a file is an external file connector, all record description entries associated with the file connector in the run unit must be the same length.

Rules for Format 1

1. The RECORD clause of format 1 is used to define the size of fixed length records. integer-1 specifies the number of character positions of records associated with a file.
2. integer-1 specifies a value to which the size of the control area (such as line control and specification of print character type) inserted to a record by compiler is not added.
3. If a file connector associated with a file is an external file connector, the same value must be specified to integer-1 in all file description entries associated with the file connector in the run unit.

Rules for Format 2

1. The RECORD clause of format 2 is used to define the size of variable-length records. integer-2 specifies the minimum number of character positions of records associated with a file. integer-3 specifies the maximum number of character positions of records associated with a file. However, if the number of character positions is specified by the system at execution time, the value is the maximum number of character positions.
2. integer-2 and integer-3 specify a value to which the size of the control area (such as line control and specification of print character type) inserted to a record by compiler is not added.
3. If a file connector associated with a file is an external file connector, the same value must be specified to integer-2 and integer-3 in all file description entries associated with the file connector in the run unit.

4. In the case when the record size specified by the RECORD clause is greater than the number of character positions of a record description entry, when the record is written, the following value is specified to the excess part of the record:

   a. If the CHARACTER TYPE clause or the PRINTING POSITION clause is written in the record description entry, the excess part of the record is padded with spaces.

   b. If neither the CHARACTER TYPE clause nor the PRINTING POSITION clause is written in the record description entry, contents of the excess part of the record are not regulated.

5. “Record description entry size” is the value to be achieved by adding enough character positions of filler to the total number of character positions of all elementary items (excluding data item which specified the REDEFINES or RENAMES clause). If the OCCURS clause with the DEPENDING ON phrase is written in a record description entry, the minimum and maximum sizes of the record description entry are determined according to the following rules:

   a. The minimum size of a record description entry is determined by using the minimum number of table elements.

   b. The maximum size of a record description entry is determined by using the maximum number of table elements.

6. If the FROM phrase is omitted, it is assumed that the minimum size of a record description entry is specified to integer-2.

7. If the TO phrase is omitted, it is assumed that the maximum size of a record description entry is specified to integer-3.

8. If the DEPENDING ON phrase is written, the value of data-name-1 is used as follows:

   a. The number of character positions of the record to be written out must be specified to data-name-1 before the RELEASE, REWRITE, or WRITE statement is executed. When these statements are executed, the number of character positions of the record to be written out is determined by the value of data-name-1.

   b. If the execution of the READ or RETURN statement is successful, the number of character positions of the record read is specified to data-name-1.

   c. If the INTO phrase is written in the READ statement, a value specified in data-name-1 is used as the number of character positions of the sending data item of the implicit MOVE statement.

   d. Even if the DELETE, RELEASE, REWRITE, START or WRITE statement is executed, the value of data-name-1 is not changed. data-name-1 is also not changed if execution of the READ statement is unsuccessful.

9. If the DEPENDING ON phrase is omitted, the number of character position of records written out is determined according to the following rules when the RELEASE, REWRITE, or WRITE statement is executed:

   a. If the record does not include variable occurrence data items, it is determined by the number of character positions of the record.

   b. If the record includes variable occurrence data items, it is determined by the total of the size of the table calculated by the number of occurrences of variable occurrence data items (the value of data-name of the DEPENDING ON phrase of the OCCURS clause) and the size of the data items other than variable occurrence data items.

10. If the DEPENDING ON phrase is omitted and the INTO phrase is written in the READ statement, a value to be specified in data-name-1 when the DEPENDING ON phase is written is used as the number of character positions of the sending data item of the implicit MOVE statement.

RECORD Clause (Presentation File)

The RECORD clause specifies the size and format of records.

[Format 1] The size of variable length records is specified.

```
RECORD IS VARYING IN SIZE
     [[FROM integer-2] [TO integer-3] CHARACTERS]
     [DEPENDING ON data-name-1]
```

[Format 2] The size and format of records is specified by a record description entry.

```
RECORD CONTAINS
       integer-4 TO integer-5 CHARACTERS
```

Format 1 and format 2 are the same as format 2 and format 3 of the RECORD clause of sequential files, respectively.

See the section titled “RECORD clause (sequential file, relative file, and indexed file).”

RECORD Clause (Report Writer Module)

The RECORD clause specifies the size and format of records.

[Format 1] The size of fixed length records is specified.

RECORD CONTAINS integer-1 CHARACTERS

[Format 2] The size of variable length records is specified.

RECORD CONTAINS
integer-4 TO integer-5 CHARACTERS

Format 1 and format 2 are the same as format 1 and format 3 of the RECORD clause of sequential files, respectively.

See the section titled “RECORD clause (sequential file, relative file, and indexed file).”

REPORT Clause (Report Writer Module)

The REPORT clause relates report-names to report files.

[Format]

$\left\{\begin{array}{l}\underline{\text{REPORT}}\ \text{IS}\\ \underline{\text{REPORTS}}\ \text{ARE}\end{array}\right\}$ {report-name-1}...

Syntax Rules

1. Report-name-1 must be a report-name written in the report description entry of the report section of the same program. The writing order of report-name-1s is optional.
2. One report-name written in a report description entry can be written only in one REPORT clause.
3. The file which specified the REPORT clause is called a report file. A report file can be written only in the USE statement, CLOSE statement, OPEN statement of the OUTPUT phrase, or the OPEN statement of the EXTEND phase.

General Rules

1. If two or more kinds of reports are associated with one report file, two or more report-name-1s are written.
2. During the time between execution of the INITIATE statement and the TERMINATE statement for one report file, an input-output statement which references the report file must not be executed.
3. If a file connector associated with a file is an external file connector, define the file as a report file in all file description entries associated with the file connector in the run unit.

VALUE OF Clause (Sequential File, Relative File, Indexed File, and Report Writer Module)

The VALUE OF clause defines the value of data items in label records. The VALUE OF clause is an obsolete language element.

[Format]

$$\underline{\text{VALUE}}\ \underline{\text{OF}}\ \left\{\text{data-name-1 IS}\ \left\{\begin{array}{l}\text{data-name-2}\\ \text{literal-1}\end{array}\right\}\right\}\ \ldots$$

Syntax Rules

1. Data-name-2 must be qualified if necessary. Data-name-2 must not be an index data item.
2. Data-name-2 must be a data item defined in the working-storage section.

General Rule

This compiler regards the VALUE OF clause as a comment.

Sort-Merge File Description Entry

The sort-merge file description entry defines the physical structure of sort-merge files. It can be written in the file section.

[Format]

```
SD file-name-1
  [DATA RECORDS clause]
  [RECORD clause].
```

Syntax Rules

1. Level indicator SD must be written at the beginning of a sort-merge file description entry, followed by file-name-1. The order of clauses which follow the file-name-1 is optional.
2. A sort-merge file description entry must be followed by at least one record description entry. However, an input-output statement must not be executed for a sort-merge file.

General Rules

1. Rules for the DATA RECORDS clause of sort-merge files are the same as those of sequential files. See the section titled "DATA RECORDS clause (sequential file, relative file, and indexed file)."
2. Rules for the RECORD clause of sort-merge files are the same as those of sequential files. See the section titled "RECORDS clause (sequential file, relative file, and indexed file)."

Data Description Entry

A data description entry defines data items. Data description entries can be written in the based-storage, file, working-storage, constant, and linkage sections. Data description entries are those to make up a record description entry and 77-level description entry. A data description entry has three formats: one for defining data-names, one for renaming data-names, and one for defining condition-names.

[Format 1] Defines data-names.

```
level-number ⎡data-name-1⎤
             ⎣FILLER     ⎦
        [REDEFINES clause]
        [BASED ON clause]
        [BLANK WHEN ZERO clause]
        [CHARACTER TYPE clause]
        [EXTERNAL clause]
        [GLOBAL clause]
        [JUSTIFIED clause]
        [OCCURS clause]
        [PICTURE clause]
        [PRINTING POSITION clause]
        [SIGN clause]
        [SYNCHRONIZED clause]
        [USAGE clause]
        [VALUE clause].
```

[Format 2] Renames data-names.

```
66 data-name-1
   RENAMES clause.
```

[Format 3] Defines condition-names.

```
88 condition-name-1
   VALUE clause.
```

Syntax Rules

Rules for Format 1

1. Level-number must be one of 01 to 49 or 77.
2. Level-number must be written at the beginning of a data description entry of format 1. If a data-name or a FILLER is written, it must be written right after the level-number. If the REDEFINES clause is written, it must be written right after the data-name or the FILLER if there is one. The REDEFINES clause must be written right after the level-number if a data-name or a FILLER is omitted. The order of other clauses is optional.
3. If an elementary item other than an index data item, internal floating-point data item, or pointer data item is defined, the PICTURE clause must be written. If an index data item, internal floating-point data item, or pointer data item is defined, the PICTURE clause must not be written.
4. If a group item is defined, the BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses must not be written.
5. The EXTERNAL clause can be written only in the data description entry of level-number 01 in the working-storage section.
6. The EXTERNAL and REDEFINES clauses must not be written in the same data description entry.

7. The GLOBAL clause can be written only in the data description entry of level-number 01.
8. If the GLOBAL or EXTERNAL clause is written, data-name-1 must be written.
9. If the record description entry associated with the file description entry which defined the EXTERNAL or GLOBAL clause is written, data-name-1 must be written.

Rules for Format 2

Level-number 66 must be written at the beginning of a data description entry of format 2. Level-number must be followed by data-name-1, then the RENAMES clause.

Rules for Format 3

Level-number 88 must be written at the beginning of a data description entry of format 3. Level-number must be followed by condition-name-1, then the VALUE clause.

General Rules

Rules for Format 1

1. data-name-1 specifies the name of a data item. FILLER is used for defining data items which are not referenced explicitly.
2. If data-name-1 and FILLER are omitted, FILLER is assumed. This item is called filler.
3. The item which was defined using FILLER in a data description entry is called “FILLER.” FILLER cannot be referenced directly.

Rules for Format 3

1. A data description entry of format 3 is called “condition-name data description entry.” condition-name-1 specifies the name of a condition, and the VALUE clause specifies the value of condition-name-1 or the range of values.
2. A condition-name data description entry must directly follow a data description entry of a conditional variable. Conditional variable must not be any of the following data items:
   a. Condition-name
   b. Data item of level-number 66
   c. Group items influencing the data item which specified the USAGE clause other than the JUSTIFIED, SYNCHRONIZED, or USAGE IS DISPLAY clause.
   d. INDEX data item
3. If two or more entries having level-number 01 depend on a level lower than FD or SD, these entries implicitly redefine the same storage.

Notes on Data Description Entries in the Based-storage Section

The VALUE clause must not be written in data description entries other than condition-name data description entry (level-number 88) in the based-storage section. Initial values of data items defined in the based-storage section are not specified.

The EXTERNAL, CHARACTER TYPE, and PRINTING POSITION clauses must not be written in data description entries in the based-storage section.

Data items defined in the based-storage section can be written in place of “identifier” in formats.

Data items defined in the based-storage section can be written in place of “data-name” in the following formats. However, these data-names must not be pointed explicitly.

- Data-name of the REDEFINES clause
- Data-name of the RENAMES clause
- Data-name of the THROUGH phrase of the RENAMES clause
- Data-name of the KEY phrase of the OCCURS clause
- Data-name of the WHEN phrase of the SEARCH ALL statement

If the data item defined in the based-storage section is specified in the data-name of the WHEN phrase of the SEARCH ALL statement, an identifier of the SEARCH ALL statement must be pointed implicitly.

Notes on Record Description Entries in the File Section

The value clause must not be written in data description entries other than condition-name data description entry (level-number 88) in the file section. Initial values of data items defined in the file section are not specified.

Notes on Data Description Entries in the Working-storage Section

Initial values can be specified in data items other than index data items defined in the working-storage section. The VALUE clause is written to specify initial values. If the VALUE clause is omitted, initial values of data items are not specified.

Notes on Data Description Entries in the Constant Section

Initial values must be specified in data items defined in the constant section by writing the VALUE clause.

Data items defined in the constant section can be written anywhere data items in the working-storage section can be written.

Notes on Data Description Entries in the Linkage Section

The VALUE clause must not be written in data items other than condition-name data description entry (level-number 88) in the linkage section. Initial values of data items defined in the linkage section are not specified.

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause replaces a value with spaces when the value of a data item is zero.

[Format]

$$\underline{\text{BLANK}}\ \text{WHEN} \left\{ \begin{array}{l} \underline{\text{ZERO}} \\ \underline{\text{ZEROS}} \\ \underline{\text{ZEROES}} \end{array} \right\}$$

Syntax Rules

1. The BLANK WHEN ZERO clause can be specified in elementary items only.
2. The BLANK WHEN ZERO clause can be specified in numeric data items or numeric edited data items only.
3. The only numeric data item in which the BLANK WHEN ZERO clause can be specified is an unsigned zoned decimal item.
4. ZERO, ZEROS, and ZEROES are synonyms.

General Rules

1. The BLANK WHEN ZERO clause specifies that the value of a data item is replaced with spaces when the value is zero.
2. If the BLANK WHEN ZERO clause is specified in a numeric data item, the category of the numeric data item is assumed to be numeric edited.

CHARACTER TYPE Clause

The CHARACTER TYPE clause specifies format of characters when printing.

[Format 1] Specifies the character size of National data items or national edited data items.

$$[\underline{\text{CHARACTER}}\ \underline{\text{TYPE}}\ \underline{\text{IS}}]\ \left\{\begin{array}{l}\underline{\text{MODE-1}}\\ \underline{\text{MODE-2}}\\ \underline{\text{MODE-3}}\end{array}\right\}\ [\text{BY mnemouic-name-1}]$$

[Format 2] Specifies the character size, pitch, font, rotations, and style of National data items or national edited data items.

$$\underline{\text{CHARACTER}}\ \underline{\text{TYPE}}\ \text{IS mnemonic-name-2}$$

[Format 3] Specifies the character size, pitch, font, rotations, and style of National data items or national edited data items or optional data item.

$$\begin{array}{l}\underline{\text{CHARACTER}}\ \underline{\text{TYPE}}\ \text{IS}\\ \quad\left\{\begin{array}{l}\text{printing-mode-name-1}\\ [\text{printing-mode-name-2}]\ \ldots\ \underline{\text{DEPENDING}}\ \text{ON data-name-1}\end{array}\right\}\\ \quad[\underline{\text{OR}}\\ \quad\left\{\begin{array}{l}\text{printing-mode-name-3}\\ [\text{printing-mode-name-4}]\ \ldots\ \underline{\text{DEPENDING}}\ \text{ON data-name-2}\end{array}\right\}\ ]\end{array}$$

Syntax Rules

Rules Common to Format 1 to Format 3

1. The CHARACTER TYPE clause can be written in data items of any level-number.
2. The CHARACTER TYPE clause cannot be specified in a data item which specified a REDEFINES clause other than level-number 01 or 77 or a data item which depends on such a data item.
3. The OCCURS clause of the DEPENDING ON phrase must not be written in the record description entry containing the data item which specified the CHARACTER TYPE clause.

Rules Common to Format 1 and Format 2

1. The CHARACTER TYPE clause can be specified only in a group item which influences National data items and national edited data items or a group item which influences national edited data items.
2. If the CHARACTER TYPE clause is specified in a group item, it is applied to all National data items and national edited data items which depend on the group item.
3. If the CHARACTER TYPE clause is specified both in a group item and a data item which depends on the group item, CHARACTER TYPE clauses having the same meaning must be specified.

Rules for Format 1

1. Mnemonic-name-1 must be associated with the following function-name in the special-names paragraph of the environment division.

   HSC, F0202, H0202, F0102, F0201

2. If mnemonic-name-1 is associated with the following function-name, only MODE-1 or MODE-2 can be specified.

   HSC, H0202

Rules for Format 2

Mnemonic-name-2 must be associated with function-name starting with the following character-string in the special-names paragraph of the environment division.

GTA, GTB, GTC, GTD, GTX, GYA, GYB, GYC, GYD, GYX, TA, TB, TC, TD, TX, YA, YB, YC, YD, YX

Rules for Format 3

1. The CHARACTER TYPE clause of format 1 or format 2 must not be specified in a data item that depends on the data item which specified the CHARACTER TYPE clause of format 3 or in a data item that influences the data item which specified the CHARACTER TYPE clause of format 3.

2. Printing-mode-name-1 to printing-mode-name-4 must be defined with the PRINTING MODE clause in the special-names paragraph.

3. The CHARACTER TYPE clause with the OR phrase can be specified only in group items.

4. If the CHARACTER TYPE clause is specified in an elementary item, the following rules must be observed:

   a. Usage of the elementary item must be presentation.

   b. If the elementary item is a National data item or a national edited data item, printing-mode-name-1 and printing-mode-name-2 must not be associated with the PRINTING MODE clause which specified FOR SOCS in the special-names paragraph.

   c. If the elementary item is an alphabetic item, alphanumeric data item, alphanumeric edited data item, external decimal item, external Boolean item, zoned floating-point data item, or numeric edited data item, printing-mode-name-1 and printing-mode-name-2 must not be associated with the PRINTING MODE clause which specified FOR MOCS in the special-names paragraph.

5. If the CHARACTER TYPE clause is specified in a group item, it is applied to all data items for presentation which depend on the group item. Among data items which depend on the group item, however, the CHARACTER TYPE clause is not applied to a data item which specified the CHARACTER TYPE clause of format 3 and a data item which depends on the data item.

6. If printing-mode-name-2 is specified twice or more in one CHARACTER TYPE clause, all FOR phrases of the PRINTING MODE clause in the special-names paragraph for these names must be the same.

7. If the OR phrase is written, the following rules must be observed:

   a. Printing-mode-name-1 to printing-mode-name-4 must not be associated with the PRINTING MODE which specified FOR ALL in the special-names paragraph.

   b. If printing-mode-name-4 is specified twice or more in one CHARACTER TYPE clause, all FOR phrases of the PRINTING MODE clause in the special-names paragraph for these names must be the same.

   c. The FOR phrase of the PRINTING MODE clause in the special-names paragraph for printing-mode-name-3 or printing-mode-name-4 must not be the same as the FOR phrase of the PRINTING MODE clause in the special-names paragraph for printing-mode-name-1 or printing-mode-name-2.

8. Data-name-5 and data-name-6 can be qualified.

9. Data-name-5 and data-name-6 must be integer items.

10. Data-name-5 and data-name-6 must be data items defined in the working-storage, file, constant or linkage section.

11. Data-name-5 and data-name-6 must not be in the variable position in a record.

12. If the CHARACTER TYPE clause is written in the data description entry contained in the record description entry which specified the EXTERNAL clause, data-name-5 and data-name-6 must be data items having external attribute. The data description entry and the data description entry of data-name-5 and data-name-6 must be written in the same data division.

13. If the CHARACTER TYPE clause is written in the data description entry contained in the record description entry which specified the GLOBAL clause, data-name-5 and data-name-6 must be global names. The data description entry and the data description entry of data-name-5 and data-name-6 must be written in the same data division.

General Rules

Rules for Format 1

1. The CHARACTER TYPE clause of format 1 specifies the character size when National data items or national edited data items are printed.
2. MODE-1 indicates 12 point characters, MODE-2 indicates 9 point characters, and MODE-3 indicates 7 point characters.

Function-name associated with mnemonic-name-1 has the following meanings.

- HSC: En-size character
- F0202: Double size character
- H0202: Double size character of en-size
- F0102: Tall character
- F0201: Expanded character

Rules for Format 2

1. A CHARACTER TYPE clause of format 2 specifies character size, pitch, font, printing direction and style when National data items or national edited data items are printed.

2. Function-name associated with mnemonic-name-2 has the following format and meanings.

[Format]

$$[\mathrm{G}]\quad \left\{\begin{matrix}\mathrm{T}\\ \mathrm{Y}\end{matrix}\right\} \left\{\begin{matrix}\mathrm{A}\\ \mathrm{B}\\ \mathrm{C}\\ \mathrm{D}\text{-}\left\{\begin{matrix}\mathrm{12P}\\ \mathrm{9P}\end{matrix}\right\}\\ \mathrm{X}\text{-}\left\{\begin{matrix}\mathrm{12P}\\ \mathrm{9P}\\ \mathrm{7P}\end{matrix}\right\}\end{matrix}\right\}\quad [\text{-}\left\{\begin{matrix}12\\ 21\\ 22\end{matrix}\right\}]\ [\text{-H}]$$

Meanings:

- Specification of san serif font

  G: San serif font

- Specification of character printing direction

  T: Vertical printing (Characters are rotated 90 degrees counterclockwise for printing.)

  Y: Horizontal printing (normal printing)

- Specification of the character size and pitch

  A: 9 point characters are printed by 2 pitch.

  B: 9 point characters are printed by 1.5 pitch.

  C: 9 point characters are printed by 7.5 CPI.

  D-12P: 12 point characters are printed by 6 CPI.

  D-9P: 9 point characters are printed by 6 CPI.

  X-12P: 12 point characters are printed by character pitch.

  X-9P: 9 point characters are printed by character pitch.

  X-7P: 7 point characters are printed by character pitch.

- Specification of the character style

  12: Tall character

  21: Expanded character

  22: Double size character

  H: En-size character

a. 12P, 9P, and 7P are synonyms of MODE-1, MODE-2, and MODE-3 of the CHARACTER TYPE clause of format 1, respectively.

b. 12, 21, and 22 are synonyms of function-name F0101, F0201, and F0202, respectively.

c. H is a synonym of function-name HSC. 22-H is a synonym of function-name H0202.

Rules for Format 3

1. The CHARACTER TYPE clause of format 3 specifies the character size, pitch, font, rotations, and style when optional data items are printed.

2. When the DEPENDING ON phrase is specified, if the value of data-name-5 is n, then the nth printing-mode-name-2 in the list of pointing-mode-names-2 takes effect. In the same way, if the value of data-name-6 is n, then, the nth printing-mode-name-2 in the list of pointing-mode-names-4 takes effect.

3. If the number in the list of printing-mode-names-2 is m, the value of data-name-5 must be 1 to m. In the same way, if the number in the list of printing-mode-names-4 is m, the value of data-name-6 must be 1 to m.

Rules Common to Format 1 to Format 3

1. The CHARACTER TYPE clause has the meaning when records are printed with the WRITE statement.
2. For details on the character size, pitch, font, rotations, and style, see the section titled “PRINTING MODE clause.”

EXTERNAL Clause

The EXTERNAL clause gives external attribute to records.

[Format]

$$\text{IS } \underline{\text{EXTERNAL}} \left[\begin{array}{l}\underline{\text{REFERENCE}}\\ \underline{\text{REF}}\\ \underline{\text{DEFINITION}}\\ \underline{\text{DEF}}\end{array}\right]$$

DS Sun The REFERENCE, REF, DEFINITION and DEF phrases can be specified.

Syntax Rules

1. The EXTERNAL clause can be written only in the data description entry of level-number 01 in the working-storage section.
2. The data-name of level-number 01 which specified the EXTERNAL clause must not be the same as the data-name of level-number 01 which specified the other EXTERNAL clause in the same program.

3. The VALUE clause must not be written in the data item which wrote the EXTERNAL clause or a data item which depends on the data item. The VALUE clause can be written in a data description entry which wrote the EXTERNAL clause or in a condition-name entry associated with a data description entry which depends on the data description entry.

General Rules

1. The EXTERNAL clause gives external attribute to a record. External attribute is given to a data item which specified the EXTERNAL clause and all data items which depend on the data item.
2. The record which specified the EXTERNAL clause in a record description entry is called "external data record." An external data record in the run unit is associated with record-name. The external data records of the same record-name reference the same storage. The number of character positions of the external data records of the same record-name must be the same.
3. An external data record can be written on the right side hand of the REDEFINES clause. An external data record can be optionally redefined by optional programs in the run unit.
4. A name of the data description entry which wrote both of the EXTERNAL clause and the GLOBAL clause must not be used as record-name having the external attribute in the program contained in the program which wrote the data description entry.

GLOBAL Clause

The GLOBAL clause specifies that data-name is global name.

[Format]

IS GLOBAL

Syntax Rules

1. The GLOBAL clause can be written only in the data description entry of level-number 01 in the file section, working-storage section, linkage section, or constant section.
2. The GLOBAL clause must not be written in two data description entries which specified the same data-name in the same data division.
3. The GLOBAL clause must not be written in the record description entry associated with the file specified in the SAME RECORD AREA clause.

General Rules

1. The GLOBAL clause specifies that data-name is a global name. The name of the data item which specified the GLOBAL clause and the names of data items which depend on this data item are global names.
2. A global name can be referenced without redefining it in the program contained directly or indirectly in the program which defined the global name.
3. If the GLOBAL clause is specified in the data item which specified the REDEFINED clause, only a data-name of subject of the REDEFINES clause becomes global name.

JUSTIFIED Clause

The JUSTIFIED clause specifies that data be transcribed, adjusted to the rightmost end of the receiving data item.

[Format]

$$\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \\ \underline{\text{JUST}}\end{array}\right\}\ \text{RIGHT}$$

Syntax Rules

1. The JUSTIFIED clause can be specified only in an elementary item.

2. The JUSTIFIED clause can be specified only in an alphabetic item, alphanumeric data item, or National data item.

3. JUSTIFIED is a synonym of JUST.

General Rules

1. The JUSTIFIED clause specifies that data be transcribed, adjusted to the rightmost end of the receiving data item.

2. In transcription, if the JUSTIFIED clause is specified in a receiving data item, a sending data item is adjusted to the rightmost end of the receiving data item and stored. If the sizes of the sending data item and receiving data item are different, transcription is made according to the following rules:

   a. If the sending data item is larger than the receiving data item, excess characters at the leftmost end of the sending data item are truncated.

   b. If the sending data item is shorter than the receiving data item, the leftmost end of the sending data item is padded with spaces.

3. If the JUSTIFIED clause is omitted, standard alignment rules are applied.

OCCURS Clause

The OCCURS clause specifies the number of occurrences of the same data structure.

[Format 1] Specifies the fixed number of occurrences.

$$\underline{\text{OCCURS}}\ \text{integer-2 TIMES}$$
$$\left[\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\ \text{KEY IS}\ \{\text{data-name-2}\}\ \ldots\right]\ldots$$
$$[\underline{\text{INDEXED}}\ \text{BY}\ \{\text{index-name-1}\}\ldots]$$

[Format 2] Specifies the variable number of occurrences.

$$\underline{\text{OCCURS}}\ \text{integer-1}\ \ \underline{\text{TO}}\ \text{integer-2 TIMES}$$
$$\underline{\text{DEPENDING}}\ \text{ON data-name-1}$$
$$\left[\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\ \text{KEY IS}\ \{\text{data-name-2}\}\ \ldots\right]\ldots$$
$$[\underline{\text{INDEXED}}\ \text{BY}\ \{\text{index-name-1}\}\ldots]$$

Syntax Rules

Rules Common to Format 1 and Format 2

1. The OCCURS clause cannot be specified in a data description entry of level-number 01, 66, 77, or 88.

2. The OCCURS DEPENDING ON clause must not be specified in a data item depending on a data item which specified the OCCURS clause. However, this compiler can specify the OCCURS DEPENDING ON clause in a data item depending on a data item which specified the OCCURS clause.

3. data-name-1 and data-name-2 can be qualified.

4. If the KEY IS phrase is written, data-name-2 must observe the following rules:

   a. data-name-2 must be a data-name of a data description entry itself which wrote the OCCURS clause or a data-name of a data description entry which depends on the data description entry.

   b. If a data-name of a data description entry which wrote the OCCURS clause is specified in data-name-2, only one data-name-2 must be written.

5. data-name-2 must not be subscripted.

6. If data-name-2 is not unique in the range of a data description entry which specified the OCCURS clause and a data description entry which depends on the data description entry.

7. For the maximum value of integer-2, see Appendix B, “System Quantitative Restriction.”

8. The OCCURS clause must not be written in a data description entry of data-name-2 except for the case when a data-name of the data description entry which wrote the OCCURS clause is specified in data-name-2.

9. In a data description entry which wrote the KEY IS phrase, if a data-name of a data description entry which depends on the data description entry is specified in data-name-2, a data description having the OCCURS clause with another KEY IS phrase must not be written between the data description entry which wrote the KEY IS phrase and the data description entry of data-name-2.

10. The data item length of data-name-2 must be 256 bytes or less.

11. data-name-2 must not be a Boolean item or index data item.
12. For the maximum number of a sequence of data-name-2, see Appendix B, “System Quantitative Restriction.”
13. A data item which specified index-name-1 and a data item depending on the data item which specified index-name-1 can be referenced with index-name-1 as a subscript.
14. Storage of index-name is automatically allocated by the compiler. data-name is not data and it does not depend on the hierarchy of data.
15. For the maximum number of sequence of index-name-1, see Appendix B, “System Quantitative Restrictions.”
16. The OCCURS clause with the INDEXED BY phrase must not be written in a data description entry with the EXTERNAL clause available.

Rules for Format 2

1. integer-1 must be 0 or more. integer-2 must be greater than integer-1.
2. data-name-1 must be an integer item.
3. data-name-1 must not be defined in the range from the first character position of a data description entry which wrote the OCCURS clause to the last character position of a record description entry which contains the data description entry.
4. In a record description entry, a data description entry which specified an OCCURS clause of format 2 can be followed by a data description entry which depends on the data description entry only. In this compiler, however, a data description entry which wrote an OCCURS clause of format 2 can be followed by a data description entry which does not depend on the data description entry.

5. An OCCURS clause of format 2 must not be specified in a data item in the constant section.

6. If the OCCURS clause with the KEY IS phrase is written in a data description entry included in a record description entry which specified the EXTERNAL clause, data-name-1 must be a data item having external attribute. Also, the data description entry and a data description entry of data-name-1 must be written in the same data division.

7. If the OCCURS clause with the KEY IS phrase is written in a data description entry contained in a record description entry which specified the GLOBAL clause, data-name-1 must be a global name. Also, the data description entry and a data description entry of data-name-1 must be written in the same data division.

General Rules

Rules Common to Format 1 and Format 2

1. Clauses written in a data description entry which specified the OCCURS clause are applied to each data description entry repeated. However, the OCCURS clause itself is not applied repeatedly.

2. If a group item which specified the OCCURS clause influences a binary item which specified the SYNCHRONIZED clause, the compiler adds necessary slack bytes to each occurrence of the group item. For slack bytes, see the section titled “Data boundary adjustment.”

3. The number of occurrences of a data item which specified the OCCURS clause is as follows:

   a. In format 1, the value of integer-2 indicates the fixed number of occurrences.

   b. In format 2, the current value of the data item of data-name-1 indicates the number of occurrences.

4. If the KEY IS phrase is written, the occurrence data must be arranged in ascending order (when ASCENDING is written) or descending order (when DESCENDING is written) according to the value of data-name-2. The order is determined by rules for comparison. These data-names are listed from left to right in the order of the key strength.

5. If the OCCURS clause with the INDEXED BY phrase is specified in an internal Boolean item, the SYNCHRONIZED clause must be specified in the internal Boolean item.

Rules for Format 2

1. Format 2 indicates that the number of occurrences of the data item which specified the OCCURS clause is variable. The maximum number of occurrences is specified in integer-2. The minimum number of occurrences is specified in integer-1. Format 2 indicates that the size of the data item which specified the OCCURS clause is not variable, but the number of occurrences is variable. The data item which specified the OCCURS clause of format 2 is called the variable occurrence data items.

2. The value of data-name-1 must be equal to or greater than integer-1 and also equal to or less than integer-2 when the data item which specified the OCCURS clause is referenced or the data item depending on or influencing the data item which specified the OCCURS clause is referenced. When the value of data-name-1 is changed, contents of the data item

having occurrence number larger than integer-2 are not specified.

3. If a group item influencing a data item which specified an OCCURS clause of format 2 is referenced, the following part is processed among the table area of the data item which specified the OCCURS clause:

    a. When the group item does not influence data-name-1, if the value of the data item of data-name-1 at the start of processing is n, the table area from the first to nth elements is processed.

    b. When the group item influences data-name-1 and it is used as a sending item, if the value of the data item of data-name-1 at the start of processing is n, the table area from the first to nth elements is processed. When the group item is used as a receiving item, the table area indicated by integer-2 is processed.

4. If an OCCURS clause of format 2 is specified in a record description entry and also the VARYING phrase is written in the RECORD clause of the related file description entry or sort-merge file description entry, the record is variable length.

5. If the record description entry containing the data item which specified an OCCURS clause of format 2 is associated with the file description entry or sort-merge file description entry which wrote the RECORD clause with the VARYING phrase and without the DEPENDING ON phrase, the number of occurrences must be specified in data-name-1 before the RELEASE, REWRITE, or WRITE statement is executed.

6. An OCCURS clause of format 2 must not be specified for an internal Boolean item which omitted the SYNCHRONIZED clause.

PICTURE Clause

The PICTURE clause specifies category, size, and editing format of elementary items.

[Format]

$$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \\ \underline{\text{PIC}}\end{array}\right\}\ \text{IS character-string}$$

Syntax Rules

1. The PICTURE clause can be specified only in an elementary item.
2. A PICTURE character-string is made up of a particular combination of characters included in COBOL character set. The category of elementary items is specified by this combination.
3. Lower case alphabetic characters corresponding to PICTURE symbols A, B, E, N, P, S, V, X, Z, CR, and DB are equivalent to their upper case alphabetic characters in a PICTURE character-string.
4. A PICTURE character-string may contain a maximum of 30 characters.

5. The PICTURE clause must be specified in an index data item, internal floating-point data item, pointer data item, or every elementary item except for the subject of the RENAMES clause. The PICTURE clause must not be written in an index data item or the subject of the RENAMES clause.

6. PICTURE and PIC are synonyms..

7. If asterisk (*), zero suppression symbol, is specified in a PICTURE character-string, the BLANK WHEN ZERO clause cannot be specified.

General Rules

Combination of PICTURE Symbols and Category of Elementary Items

The category of elementary items is specified by a combination of PICTURE symbols. There are nine categories of elementary items: alphabetic character, numeric character, alphanumeric character, alphanumeric edited, numeric edited, national language, national edited, Boolean, and external floating-point.

1. An elementary item whose category is alphabetic character is called “alphabetic item.” Rules on specifying an alphabetic item and its contents are shown below.

   a. A PICTURE character-string is made up of only symbol A.

   b. Usage of an alphabetic item can be DISPLAY only.

   c. An alphabetic item must consist of one or more alphabetic characters.

2. An elementary item whose category is numeric character is called “numeric data item.” Rules on defining a numeric data item and its contents are shown below.

   a. A PICTURE character-string is made up of a combination of symbols 9, P, S, and V. The number of digit positions allowed in a PICTURE character-string is 1 to 18. That is, the total number of symbols 9 and P must be 1 to 18.

   b. The usage of a numeric data item can be DISPLAY, COMPUTATIONAL, BINARY, PACKED-DECIMAL, or COMPUTATIONAL-5.

   c. If signs are not specified in a numeric data item, the contents of a numeric data item must contain one or more numeric characters. If specified, it may also contain operational signs.

   d. A numeric data item which does not have digits after the decimal point is called “integer item.”

3. An elementary item whose category is alphanumeric character is called “alphanumeric data item.” Rules on defining an alphanumeric data item and its contents are shown below.

   a. A PICTURE character-string is made up of a combination of symbols A, X, and 9. An alphanumeric data item having a PICTURE character-string which contains symbols other than X is handled as one having a PICTURE character-string which contains only X. However, a data item having a PICTURE character-string which contains all As or all 9s is not an alphanumeric data item.

   b. The usage of an alphanumeric data item can be DISPLAY only.

   c. An alphanumeric data item must consist of one or more characters of computer's character set.

4. An elementary item whose category is alphanumeric edited is called “alphanumeric edited data item.” Rules on defining an alphanumeric edited data item and its contents are shown below.

    a. A PICTURE character-string is made up of a combination of symbols A, X, 9, B, 0, and /. This character-string must contain at least one A or X, and at least one B, 0, or /.

    b. The usage of an alphanumeric edited data item can be DISPLAY only.

    c. An alphanumeric edited data item must consist of at least two characters of computer's character set.

5. An elementary item whose category is numeric edited is called “numeric edited data item.” Rules on defining a numeric edited data item and its contents are shown below.

    a. A PICTURE character-string is made up of a combination of symbols 9, P, V, B, /, Z, 0, comma (,), decimal point (.), *, +, -, CR, DB, and currency symbol. This character-string must contain at least one B, / , Z, 0, comma, decimal point, *, +, -, CR, DB, or currency symbol. Both symbol P and decimal point must not be specified in a PICTURE character-string. The number of digit positions represented in a PICTURE character-string ranges from 1 to 18. That is, the total number of symbols 9 and P must range from 1 to 18.

    b. The usage of a numeric edited data item can be DISPLAY only.

    c. Contents of each character position of a numeric edited data item must not conflict with a PICTURE character-string.

6. An elementary item whose category is National is called “National data item.” Rules on defining a National data item and its contents are shown below.

   a. A PICTURE character-string is made up of only symbol N.

   b. The usage of a National data item can be DISPLAY only.

   c. A National data item must consist of one or more national characters.

7. An elementary item whose category is National edited is called a “national edited data item.” Rules on defining a national edited data item and its contents are shown below.

   a. A PICTURE character-string is made up of a combination of N and B. This character-string must contain at least one N and one B.

   b. The usage of a national edited data item can be DISPLAY only.

   c. A national edited data item must consist of two or more national characters.

8. An elementary item whose category is Boolean is called “Boolean item.” Rules on defining a Boolean item and its contents are shown below.

   a. A PICTURE character-string is made up of only symbol 1.

   b. The usage of a Boolean item can be BIT or DISPLAY.

   c. A Boolean item must consist of a combination of Boolean symbols 0 and 1.

9. An elementary item whose category is external floating point is called “external floating-point data item.” The usage of an external floating-point data item can be DISPLAY only. Rules on defining an external floating-point data item and its contents are shown below.

   a. Format of a PICTURE character-string is shown below.

$$\left\{\begin{array}{c}+\\-\end{array}\right\} \text{mantissa E} \left\{\begin{array}{c}+\\-\end{array}\right\} \text{exponent}$$

   b. Signs of mantissa part and characteristic must be either one plus sign (+) or one minus sign (-). These signs have the following meanings:

      +: If the value of the data item is positive or zero, append +. If the value of the data item is negative, append “-”. A “+” is counted in the size of an elementary item.

      -: If the value of the data item is positive or zero, append a space. If the value of the data item is negative, append “-”. A “-” is counted in the size of an elementary item.

   c. Mantissa must be a character-string made up of “.”, “9”, and “V”. This character-string must contain either “V” or “.” which indicates a floating point. These characters have the following meanings:

      9: Indicates one numeric character. “9” is counted in the size of an elementary item. The number of 9s, that is digit positions of mantissa, must range from 1 to 16.

      V: Indicates the position of assumed decimal point. A “V” is not counted in the size of an elementary item.

      .: Indicates the position of actual decimal point. A “.” is counted in the size of an elementary item.

d. An "E" indicates that characteristic follows. An "E" is counted in the size of an elementary item.Exponent must be character-string "99". A "99" is counted in the size of an elementary item.

e. The value of an external floating-point data item is indicated in the following formula.

$$\left\{\begin{matrix}+\\-\end{matrix}\right\}\text{(value of mantissa)} * 10^{\left\{\begin{matrix}+\\-\end{matrix}\right\}\text{(value of exponent)}}$$

f. The VALUE and BLANK WHEN ZERO clauses must not be specified in an external floating-point data item.

g. The following table lists examples of a PICTURE character-string and the value of an external floating-point data item.

A PICTURE Character-string	**External Data Format**	**Value**
+9.9E+99	-5.4E-79	-5.4x(10 to the -79th power)
-99.9(5)E-99	12.34567E 00	12.34567x(10 to the 0th power)
+9(8)VE-99	+12345678E-09	+12345678x(10 to the -9th power)
-V9(5)E+99	-.72000E+76	-0.72x(10 to 76th power)

Rules for a PICTURE Character-string

1. The number of digits of an elementary item counted by standard data format is called "elementary item size." The elementary item size is determined by the number of characters indicating character positions.

2. Symbols A, comma (,), X, N, 9, 1, P, Z, *, B, / , 0, +, -, and currency symbol can be used repeatedly. If the same symbol is used repeatedly, the number of its occurrences enclosed in parentheses is specified next to the symbol. The number of occurrences must be an unsigned integer other than 0. A sequence of a same symbol is equal to the symbol with the number of its occurrences enclosed in parentheses. For an example, 999 is equal to 9(3).
3. A PICTURE character-string can contain only one of the following: E, S, V, decimal point (.), CR, and DB
4. For the maximum size of an elementary item, see Appendix B, “System Quantitative Restriction.”

Meaning of PICTURE Symbols

1. An “A” indicates one character position including only an alphabetic character or space. An “A” is counted in the size of an elementary item.
2. A “B” indicates one character position for inserting space or one national character position for inserting space of national language. A “B” is counted in the size of an elementary item.
3. A “P” indicates an assumed decimal scaling position. A “P” is counted in the size of an elementary item. A “P” may appear in the leftmost or rightmost digit positions in a PICTURE numeric character. If “P” is written at the left of a PICTURE numeric character, assumed decimal point is assumed at the left of “P” at the head. If “P” is written at the right of a PICTURE numeric character, assumed decimal point is assumed at the right of “P” at the end. In both cases, “V,” which indicates the position of assumed decimal point, can be omitted.

4. An “S” indicates that an operational sign exists. An “S” must be written at the head of the PICTURE character-string. If the SEPARATE CHARACTER phrase is written in the SIGN clause, “S” is counted in the size of an elementary item. If a phrase other than the SEPARATE CHARACTER phrase is written or the SIGN clause is omitted, “S” is not counted in the size of an elementary item. “S” does not indicate expression format or the position of operational sign.

5. A “V” indicates the position of assumed decimal point. One “V” can be written in a PICTURE character-string. A “V” is not counted in the size of an elementary item, because it does not indicate character position. If assumed decimal point is at the rightmost end of a PICTURE character-string (“P” or repetition of “P”) which indicates digit position or digit alignment position, “V” can be omitted.

6. An “X” indicates one character position. Character position of “X” can contain optional character allowed for computer's character set. An “X” is counted in the size of an elementary item.

7. An “N” indicates one national character position. National character position of “N” can contain optional national character. An “N” is counted in the size of an elementary item.

8. A “Z” indicates that a leading zero string (upper digits of effective numeric characters) is replaced with spaces. A “Z” is counted as a character position in the size of an elementary item.

9. A “9” indicates one digit position including numeric character. A “9” is counted in the size of an elementary item.

10. A “1” indicates one Boolean position including Boolean character. A “1” is counted in the size of an elementary item.

11. A “0” indicates one character position for inserting numeric character zero. A “0” is counted in the size of an elementary item.

12. A “/” indicates one character position for inserting character “/.” A “/” is counted in the size of an elementary item.

13. A comma (,) indicates one character position for inserting character comma. A comma is counted in the size of an elementary item.

14. A decimal point (.) indicates the one character position where the decimal point should be set. A decimal point also indicates the decimal point position to be the standard for aligning digits of a data item. A decimal point is counted in the size of an elementary item. If the DECIMAL-POINT IS COMMA clause is written in the special-names paragraph, functions of decimal point and comma in a PICTURE character-string written in a program and numeric literal are exchanged. In this case, rules related to a decimal point in the PICTURE clause are applied to a comma and rules related to a comma are applied to a decimal point.

15. Symbols “+”, “-”, CR, and DB indicate the character position where a symbol indicating positive or negative should be stored. These symbols are called editing sign control symbols. One of these symbols can be specified in a PICTURE character-string. “+” and “-” are each counted as a character position in the size of an elementary item. CR and DB are each counted as two character positions in the size of an elementary item.

16. An “*” indicates that a leading zero string (upper digits of effective numeric characters) is replaced with “*”. An “*” is counted as a character position in the size of an elementary item.

17. A currency symbol indicates the character position where a currency symbol should be stored. “Currency symbol” is a character specified by the CURRENCY SIGN clause in the special-names paragraph. If the CURRENCY SIGN clause is omitted, the currency sign becomes the currency symbol. A currency symbol is counted as one character position in the size of an elementary item.

18. An “E” indicates the start of the characteristic in an external floating-point data item. An “E” is counted as one character position in the size of an elementary item.

Rules for a Data Item Containing “P”

1. If a data item which contains “P” is written in the places explained below, the algebraic value rather than the actual character representation is used as the value of the data item. An algebraic value is a value that assumes the decimal point at a position determined by the position of “P” and assumes zero at the position of “P.” The size of an algebraic value is the number of digit positions represented by the PICTURE character-string, that is the total number of “P” symbols and “9” symbols.

   a. In a movement where the sending operand is a numeric data item, when a numeric data item which contains “P” is specified in the sending operand

   b. When a numeric data item which contains “P” is specified in the sending operand of a MOVE statement

   c. When a numeric edited data item which contains “P” is specified in the sending operand of the MOVE statement and a numeric data item or numeric edited data item is specified in the receiving operand

d. In a comparison operation where both operands are numeric data items, when a numeric data item which contains “P” is specified in at least one of the operands

If a data item which contains “P” is written at a place other than mentioned above, the position of “P” is ignored and not counted in the size of the operand.

Editing Rules for an Edited Item

Two editing methods, insertion and zero suppression, can be specified in a PICTURE character-string. One or more editing methods shown below are specified in a PICTURE character-string of a numeric edited data item.

- Editing method of a numeric edited data item
  - Insertion
    - Simple insertion
    - Special insertion
    - Fixed insertion
    - Floating insertion
  - Zero suppression
    - Zero suppression by space
    - Zero suppression by asterisk “* ”

Simple insertion of space, “0,” or “ / ” is specified in a PICTURE character-string of an alphanumeric edited data item.

Simple insertion of space of National is specified in a PICTURE character-string of a national edited data item.

SIMPLE INSERTION

1. In the "simple insertion" method, a comma, space, "0," or "/" is inserted in the position of the insertion character specified by a PICTURE character-string. When simple insertion is made, a comma, "B," "0," or "/" is written in a PICTURE character-string as an insertion character. These insertion characters are called "simple insertion character." A simple insertion character, "B," indicates that space is inserted. A simple insertion character shows the character position for insertion.

2. If a comma (,) is written at the end of a PICTURE character-string, the PICTURE clause must be the last clause in the data description entry. Also the PICTURE clause must be directly followed by a period. In the result, ",." appears in the data description entry. If the DECIMAL-POINT IS COMMA clause is written, two consecutive periods ".." appear.

SPECIAL INSERTION

1. In the "special insertion" method, a decimal point is inserted in the position of the insertion character specified by a PICTURE character-string. If special insertion is made, a decimal point (.) is written in a PICTURE character-string as an insertion character. This insertion character is handled as one of the simple insertion characters. A decimal point is not only an insertion character, but also indicates the actual position of the decimal point. Period (.) and "V" are mutually exclusive in one PICTURE clause.

2. If a period (.) is written at the end of a PICTURE character-string, the PICTURE clause must be the last clause in the data description entry. Also the PICTURE clause must be directly followed by a period. In the result, two consecutive periods ".." appear in the data description entry. If the DECIMAL-POINT IS COMMA clause is written, ",." appear.

FIXED INSERTION

1. In the "fixed insertion" method, a character corresponding to an insertion character is inserted in the position of the insertion character specified by a PICTURE character-string according to the value of a data item. If fixed insertion is made, a currency symbol and editing sign control symbol (+, -, CR, or DB) are written as insertion characters. These characters are called fixed insertion characters. Only one currency symbol and one editing sign control symbol can be written in a PICTURE character-string.

2. The position of CR or DB in a PICTURE character-string must be the right end of all character positions counted in the size of an elementary item.

3. The position of CR or DB in a PICTURE character-string must be the left end or right end of all character positions counted in the size of an elementary item.

4. The position of a currency symbol must be the left end of all character positions counted in the size of an elementary item except for cases when the currency symbol is preceded by "+" or "-".

5. Characters listed in the following table are inserted in the position of editing sign control symbols written in a PICTURE character-string according to the value of a data item.

Editing Sign Control Symbol Written in PICTURE Character-string	**Character Inserted as a Result of Editing**	
	When the Value of a Data Item is Positive or Zero	**When the Value of a Data Item is Negative**
+	+	-
-	One space	-
CR	Two spaces	CR
DB	Two spaces	DB

FLOATING INSERTION

1. In the “Floating insertion” method, a character corresponding to an insertion character is inserted in the position of consecutive insertion characters specified by a PICTURE character-string according to the value of a data item. If floating insertion is made, two or more consecutive currency symbols, two or more consecutive plus symbols, or two or more consecutive minus symbols are written as insertion characters. These characters are called “floating insertion characters.” A PICTURE character-string must not contain both a plus and a minus symbol.
2. Simple insertion characters can be written between floating insertion character strings or just on the right side of a string. These simple insertion characters are regarded as a part of the floating insertion character-string.
3. If a currency symbol is written as a floating insertion character, a fixed insertion character CR or DB can be written at the immediate right of the floating insertion character-string.

4. The leftmost character of a floating insertion character-string indicates the leftmost character position for inserting a floating insertion character. The rightmost character of a floating insertion character-string indicates the rightmost character position for inserting a floating insertion character. The second character from the left side of a floating insertion character-string indicates the rightmost character position for string numeric data in a data item.

5. If the head of digit positions corresponding to the integer part of a PICTURE character-string or all digit positions corresponding to the integer part are specified by insertion characters, the result of floating insertion is as follows:

   a. If the value of integer part of a data item is zero, one insertion character is inserted in the character position to the right of the decimal point. Spaces are inserted on the left side of the insertion character.

   b. If the value of the integer part of a data item is not zero, a insertion character is inserted in the character position to the left of the first non-zero numeric character in the integer part. Spaces are inserted on the left side of the insertion character.

6. If all digit positions of a PICTURE character-string are specified by insertion characters, the result of floating insertion is as follows:

   a. If the value of a data item is zero, the data item is filled with spaces.

   b. If the value of a data item is not zero, one insertion character is inserted in the character position to the left of the decimal point or at the first non-zero character in the integer part. Spaces are inserted on the left side of the insertion character.

7. Currency symbols are inserted in the position of currency symbols written in a PICTURE character-string. Characters listed in the following table are inserted in the position of “+” or “-” written in a PICTURE character-string:

Editing Sign Control Symbol Written in PICTURE Character-string	Character Inserted as a Result of Editing	
	When the Value of a Data Item is Positive or Zero	When the Value of a Data Item is Negative
+	+	-
-	One space	-

8. To avoid truncation in transfer in which a numeric edited data item is the receiving area, the number of characters of the PICTURE character-string must be more than the string to which the following three are added:

   a. The number of characters of the sending data item

   b. The number of fixed insertion characters specified in the PICTURE character-string in the receiving data item

   c. One character for a floating insertion character

   If truncation occurs, the value of data after truncation is used for editing. For details on the truncation method, see the section titled “Standard alignment rules.”

Zero Suppression

1. “Zero suppression” is replacement of digits of a leading zero string (digits above significant digits) with spaces or asterisks (*). If zero suppression is made, one “Z” or consecutive Z's or one asterisk or consecutive asterisks are written in the PICTURE character-string. These characters are called “zero suppression characters.” Zero suppression characters indicate digit positions for replacement of a leading zero string. “Z” written in a digit position of a PICTURE character-string is replaced with a space, and “*” in a digit position is replaced with “*”. “Z” and “*” are mutually exclusive in a PICTURE character-string.
2. Simple insertion characters can be written between zero suppression character-strings or just on the right side of a string. These simple insertion characters are regarded as a part of the zero suppression character-string.
3. If the head of digit positions corresponding to the integer part of a PICTURE character-string or all digit positions corresponding to the integer part are specified by zero suppression characters, the result of zero suppression is as follows:
    a. If the value of the integer part of a data item is zero, the leading zero string up to the left side of the decimal point is replaced with spaces or asterisks (*).
    b. If the value of the integer part of a data item is not zero, the leading zero string up to the left side of the first non-zero numeric character of the integer part is replaced with spaces or asterisks (*).

4. If all digit positions of a PICTURE character-string are specified by zero suppression characters, the result of zero suppression is as follows:

   a. If the value of a data item is zero, the result is as follows:

      If Z is written as the zero suppression character, the data item is entirely replaced with spaces.

      If * is written as the zero suppression character and simple insertion character decimal point is not written, the data item is entirely replaced with asterisks (*).

      If * is written as the zero suppression character and also simple insertion character decimal point (.) is written, digit positions other than the decimal point are replaced with asterisks (*) and the digit position of the decimal point is replaced with a decimal point.

   b. If the value of a data item is not zero, the decimal point or the leading zero string up to the left side of the first non-zero numeric character of the integer part is replaced with space(s) or asterisk(s) (*).

Combination of Floating Insertion and Zero Suppression

Floating insertion and zero suppression are mutually exclusive for one numeric edited data item. Also, both kinds of zero suppression cannot be specified. Therefore, when +, -, *, Z, and the currency symbol are used as floating insertion characters or zero suppression characters, only one of them can be written.

Rules for a PICTURE Character-string Sequence

Rules for a PICTURE character-string sequence are shown below. Characters which can be written in a PICTURE character-string depend on the category. Apply rules for the table based on the category.

The following explains how to read the table:

o: The characters shown at the top of the row can be written before (no need to be immediately before) the character shown at the left side of the line.

-: The characters shown at the top of the row must not be written before the character shown at the left side of the line.

Currency: Currency symbol

Left: A first symbol or second symbol is written to the left of the decimal point.

Right: A first symbol or second symbol is written to the right of the decimal point.

First symbol Second symbol			Simple insertion symbols					Fixed insertion symbols				Zero suppression symbols		Floating insertion symbols				Other symbols							
			B	0	/	,	.	+ -		CB DB	Currency	Z *		+ -		Currency		9	A X	S	V	P		N	I
								L	R			L	R	L	R	L	R					L	R		
Simple insertion symbols	B		○	○	○	○	○	○	-	-	○	○	○	○	○	○	○	○	○	-	○	-	○	○	-
	0		○	○	○	○	○	○	-	-	○	○	○	○	○	○	○	○	○	-	○	-	○	-	-
	/		○	○	○	○	○	○	-	-	○	○	○	○	○	○	○	○	○	-	○	-	○	-	-
	,		○	○	○	○	○	○	-	-	○	○	○	○	○	○	○	○	-	-	○	-	○	-	-
	.		○	○	○	○	-	○	-	-	○	○	-	○	-	○	-	○	-	-	-	-	-	-	-
Fixed insertion symbols	+ -	L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		R	○	○	○	○	○	-	-	-	○	○	○	-	-	○	○	○	-	-	○	○	○	-	-
	CB DB		○	○	○	○	○	-	-	-	○	○	○	-	-	○	○	○	-	-	○	○	○	-	-
	Currency		-	-	-	-	-	○	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Zero suppression symbols	Z *	L	○	○	○	○	-	○	-	-	○	○	-	-	-	-	-	-	-	-	-	-	-	-	-
		R	○	○	○	○	○	○	-	-	○	○	○	-	-	-	-	-	-	-	○	-	○	-	-
Floating insertion symbols	+ -	L	○	○	○	○	-	-	-	-	○	-	-	○	-	-	-	-	-	-	-	-	-	-	-
		R	○	○	○	○	○	-	-	-	○	-	-	○	○	-	-	-	-	-	○	-	○	-	-
	Currency	L	○	○	○	○	-	○	-	-	-	-	-	-	-	○	-	-	-	-	-	-	-	-	-
		R	○	○	○	○	○	○	-	-	-	-	-	-	-	○	○	-	-	-	○	-	○	-	-
Other symbols	9		○	○	○	○	○	○	-	-	○	○	-	○	-	○	-	○	○	○	○	-	○	-	-
	A X		○	○	○	-	-	-	-	-	-	-	-	-	-	-	-	○	○	-	-	-	-	-	-
	S		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	V		○	○	○	○	-	○	-	-	○	○	-	○	-	○	-	○	-	○	-	○	-	-	-
	P	L	○	○	○	○	-	○	-	-	○	○	-	○	-	○	-	○	-	○	-	○	-	-	-
		R	-	-	-	-	-	○	-	-	○	-	-	-	-	-	-	-	-	○	○	-	○	-	-
	N		○	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○	-
	I		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	○

PRINTING POSITION Clause

The PRINTING POSITION clause specifies a column for when printing.

[Format]

PRINTING POSITION IS integer-1
[BY positioning-unit-name-1]

Syntax Rules

1. If the PRINTING POSITION clause is specified in a National data item or national edited data item, the CHARACTER TYPE clause must be specified in the data item or in a group item which influences the data item.
2. The PRINTING POSITION clause must not be specified in a data item which specified the OCCURS clause or in a data item depending on the data item.
3. The OCCURS clause with the DEPENDING ON phrase must not be written in a record description entry containing a data item which specified the PRINTING POSITION clause.
4. The PRINTING POSITION clause must not be specified in a data item of level number other than 01 or 77 which specified the REDEFINED clause or a data item which depends on the data item.
5. Any data items in a record description entry containing a data item which specified the PRINTING POSITION clause must not be specified in the subject or object of the RENAMES clause.

6. positioning-unit-name-1 must be associated with a literal indicating positioning unit by the POSITIONING UNIT clause in the special-names paragraph.

General Rules

1. integer-1 specifies the absolute printing position in a line of printable data items.
2. positioning-unit-name-1 specifies column units. If positioning-unit-name-1 is omitted, the column units will be 10CPI.
3. integer-1 specifies the printing start position of a data item to be printed with the leftmost column of a line regarded as 1.
4. If the PRINTING POSITION clause is specified in a group item, integer-1 indicates the head printing position of the group item.
5. integer-1 must indicate a column to the right of the rightmost column of the data item defined immediately before the printable data items of the record description entry.
6. If the PRINTING POSITION clause is omitted, a data item is printed in the following column:
   a. The head data item of a record description entry is printed from the first digit of a print line.
   b. The second and subsequent data items in a record description entry are printed directly after the previous data item.
7. Some printing devices cannot use the PRINTING POSITION clause. For the relationship between printing devices and the PRINTING POSITION clause, see “COBOL85 User’s Guide.”

REDEFINES Clause

The REDEFINES clause specifies different data items in the same storage area.

[Format]

```
level-number ┌data-name-1┐  REDEFINES data-name-2
             └FILLER     ┘
```

Note: Level-number, data-name-1, and FILLER are not part of the REDEFINES clause. They are included in the format only for clarity.

Syntax Rules

1. The REDEFINES clause must directly follow level-number, data-name-1, or FILLER.
2. The level-numbers of data-name-2 and the subject of the REDEFINES clause must be the same. The level-numbers must not be 66 or 88.
3. If two or more record description entries of the file section are written, area of the record description entries are redefined implicitly. The REDEFINES clause must not be written in a data description entry of level-number 01 in the file section.
4. A data description entry of data-name-2 must not contain an OCCURS clause. It can depend on a data item which contains an OCCURS clause. A data description entry of data-name-2, a data description entry which contains a REDEFINES clause, and a data description entry which depends on either of these data items must not contain an OCCURS clause with a DEPENDING ON phrase.

5. If a data description entry of level-number other than 01 contains the REDEFINES clause or data-name-2 is a record-name of an external data record, the number of character positions of a data item of data-name-2 must be greater than or equal to the number of character positions of a data item referenced by the subject of the REDEFINES clause. If a data description entry of level-number 01 contains the REDEFINES clause and a data-name other than a record-name of an external data record is specified for data-name-2, there is no such a restriction.

6. A data description entry which specified a level-number lower than that of data-name-2 or the subject of the REDEFINES clause must not be written between a data description entry of data-name-2 and a data description entry which contains the REDEFINES clause.

7. The same character position can be redefined repeatedly. I n this case, data-name-2 must be a data-name of a data description entry which defined the character position first. In this compiler, however, data-name-2 does not need to be a data-name of an entry which defined the character position first.

8. A data description entry which contains the REDEFINES clause must not contain a VALUE clause. However, a condition-name data description entry associated with the data description entry can contain a VALUE clause.

9. A data item of data-name-2 can depend on a data item which specified the REDEFINES clause.

10. If an internal BOOLEAN item whose SYNCHRONIZED clause is omitted is specified in data-name-2, a data item to be redefined must be an internal BOOLEAN item.

11. A data description entry of data-name-2 cannot contain a CHARACTER TYPE clause. A data-name-2 cannot depend on a data item which is specified by CHARACTER TYPE clause.

General Rules

1. A storage area which starts at the head of a storage area of data-name-2 and has the size of the data item which specified the REDEFINES clause is allocated to the data item which specified the REDEFINES clause.
2. If the same character position is defined by two or more data description entries, data-names of any data description entries can be used to reference the character position.
3. If a SYNCHRONIZED clause is specified in a data item which specified the REDEFINES clause or in the first elementary item which depends on the data item, data-name-2 must be defined to match the natural boundary. For example, A must be start at the halfword boundary in the following case.

```
02 A PICTURE X(2).
02 B REDEFINES A PICTURE S9(4) BINARY SYNCHRONIZED.
```

RENAMES Clause

The RENAMES clause names a set of some elementary items.

[Format]

$$\text{66 data-name-1 }\underline{\text{RENAMES}}$$

$$\text{data-name-2}\left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\}\text{data-name-3}\right].$$

Note: Level-number and data-name-1 in the format are indicated only for clarifying the format. They are not part of the RENAMES clause.

Syntax Rules

1. A data description entry which contains the RENAMES clause is called “RENAMES entry.” Any number of record description entries can be written in one record description entry.
2. data-name-2 and data-name-3 must be names of elementary items or group items of the same record description entry. They must not be the same name. They must not be data-names defined by the data description entry of level-number 77, 88, 01, or 66.
3. RENAMES entries must be written together after the last data description entry of a record description entry.
4. data-name-1 must not be used as a qualifier.

5. data-name-1 is qualified only by a name of a data description entry of 01 level in a record description entry which contains the RENAMES clause, a file description entry, or a sort-merge file description entry.data-name-2 and data-name-3 must not be a data item which specified an OCCURS clause or a data item which depends on such a data item.

6. data-name-2 and data-name-3 can be qualified.

7. If data-name-2 or data-name-3 is not unique in a record description entry which contains the RENAMES description entry, data-name-2 or data-name-3 must be qualified.

8. A variable occurrence data item must not be specified in the range between data-name-2 and data-name-3.

9. THRU is a synonym of THROUGH.

10. A data item of data-name-3 must not depend on a data item of data-name-2.

11. The left end of the area of data-name-3 must not be to the left of the left end of the area of data-name-2. The right end of the area of data-name-3 must be to the right of the right end of the area of data-name-2.

12. If data-name-3 is written, data-name-2 and data-name-3 must not be internal BOOLEAN items.

General Rules

1. If data-name-3 is written, data-name-1 will be a group item. This group item influences data items in the following range:
   a. The first elementary item which depends on the group item is one which depends on data-name-2 if data-name-2 is an elementary item or a group item.
   b. The last elementary item which depends on the group item is one which depends on data-name-3 if data-name-3 is an elementary item or a group item.
2. If data-name-3 is omitted, all data attributes of data-name-2 become data attributes of data-name-1.
3. If data-name-3 is written, the CHARACTER TYPE and PRINTING POSITION clauses which are specified in data-name-2, data-name-3, and elementary items between them are ignored in execution of the WRITE statement specified for the FROM phrase.

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign.

[Format]

$$[\underline{\text{SIGN}}\ \text{IS}]\ \left\{\begin{array}{l}\underline{\text{LEADING}}\\ \underline{\text{TRAILING}}\end{array}\right\}\ [\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

Syntax Rules

1. The SIGN clause can be specified only in a numeric data item whose PICTURE character-string contains symbol “S” or in a group item that contains at least such a numeric data item.
2. The usage of a numeric data item to which the SIGN clause is applied must be display.

General Rules

1. The SIGN clause specifies the sign position and representation mode of a numeric data item. If the SIGN clause is specified in an elementary item, it is applied to the elementary item. If the SIGN clause is specified in a group item, it is applied to all signed numeric data items which depend on the group item. Existence of a sign is specified by S in a PICTURE character-string. The representation mode and sign position are specified by the SIGN clause.
2. If the SIGN clause is specified in a group item depending on a group item which specified the SIGN clause, the SIGN clause specified in the dependent group item is applied to the dependent group item.
3. If the SIGN clause is specified in a numeric data item depending on a group item which specified the SIGN clause, the SIGN clause specified in the dependent numeric data item is applied to the dependent numeric data item.
4. If the SIGN clause is not specified in a numeric character whose PICTURE character-string contains “S,” only TRAILING is assumed to be written in the SIGN clause.

5. Rules for signs when SEPARATE CHARACTER is omitted are shown below:

    a. An operational sign is appended to the leftmost digit position if LEADING is written or to the rightmost digit position if TRAILING is written.

    b. "S" in a PICTURE character-string is not counted in the size of a data item.

6. Rules for signs when SEPARATE CHARACTER is written are show below.

    a. An operational sign is appended to the leftmost digit position if LEADING is written or to the rightmost digit position if TRAILING is written. A character position of an operational sign occupies a character position other than digit position.

    b. The symbol S in a PICTURE character-string is counted in the size of a data item in standard data format.

    c. Positive and negative operational signs are "+" and "-" in standard data format.

    d. When "+" or "-" is not specified in a character position for an operational sign, the result of execution is undefined.

7. If a numeric data item which specified the SIGN clause is used for computation or comparison, conversion may take place. The conversion takes place automatically. For the mode of representation of a data item which specified the SIGN clause, see the section titled "USAGE clause."

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the mapping of an elementary item on a natural boundary in computer storage.

[Format]

$$\left\{\begin{array}{l}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{array}\right\}\left[\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right]$$

Syntax Rules

1. The SYNCHRONIZED clause can be specified only in an elementary item.
2. SYNC is a synonym of SYNCHRONIZED.

General Rules

1. The SYNCHRONIZED clause is effective only for an elementary item with BINARY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-5, or BIT specified in the USAGE clause. The SYNCHRONIZED clause is ignored if it is specified in other elementary items.
2. An elementary item which specified the SYNCHRONIZED clause is aligned on a natural boundary shown in the following table.

USAGE Clause Specification	Number of Digits	Natural Boundary
BINARY	1 to 4 digits	2-byte (en-size) boundary
COMPUTATIONAL, or	5 to 9 digits	4-byte (em-size) boundary
COMPUTATIONAL-5	10 to 18 digits	8-byte (double size) boundary
COMPUTATIONAL-1	-	4-byte (em-size) boundary
COMPUTATIONAL-2	-	8-byte (double size) boundary
BIT	-	1-byte boundary

3. A data item which specified the SYNCHRONIZED clause is aligned on a natural boundary by inserting necessary slack byte(s) or slack bit(s). For details on slack bytes and slack bits, see the section titled “Data boundary adjustment.”

4. Program execution efficiency is better with the SYNCHRONIZED clause specified than without.

5. The LEFT and RIGHT phrases are regarded as comments.

6. The SYNCHRONIZED clause may be specified in an elementary item which specified a REDEFINES clause or in the first elementary item that depends on a group item which specified a REDEFINES clause. In this case, the user is responsible for adjusting the data item specified in the object of the REDEFINES clause to the natural boundary specified by the SYNCHRONIZED clause.

7. The compiler regards that data items of level-number 01 in the file, working-storage, linkage, and constant sections and data items of level-number 77 in the linkage section start at 8-byte (double size) boundary. The user must secure the above mentioned by doing the following:

   a. For records in the file section, necessary slack byte between records must be defined to start data items of level-number 01 at 8 bytes boundary.

   b. The user does not need to consider records in the working-storage section and constant section, because this alignment takes place automatically.

c. For records in the linkage section and data items of level-number 77, data items corresponding to those in the calling program must be start at an 8-byte boundary.

USAGE Clause

The USAGE clause specifies the usage of a data item.

[Format]

$$[\underline{\text{USAGE}}\ \ \text{IS}] \left\{ \begin{array}{l} \underline{\text{BINARY}} \\ \underline{\text{BIT}} \\ \underline{\text{COMPUTATIONAL}} \\ \underline{\text{COMP}} \\ \underline{\text{COMPUTATIONAL-1}} \\ \underline{\text{COMP-1}} \\ \underline{\text{COMPUTATIONAL-2}} \\ \underline{\text{COMP-2}} \\ \underline{\text{COMPUTATIONAL-3}} \\ \underline{\text{COMP-3}} \\ \underline{\text{COMPUTATIONAL-5}} \\ \underline{\text{COMP-5}} \\ \underline{\text{DISPLAY}} \\ \underline{\text{INDEX}} \\ \underline{\text{PACKED-DECIMAL}} \\ \underline{\text{POINTER}} \end{array} \right\}$$

Syntax Rules

1. The USAGE clause can be written in data description entries of level-number other than 66 or 88.
2. If the USAGE clause is written in a data description entry of a group item, the USAGE clause can also be written in a data description entry of a dependent elementary item or a group item. In this case, the same USAGE clause must be written in both data description entries.
3. Elementary items with the following specified by the USAGE clause and elementary items which depend on a group item with the following specified by the USAGE clause must be numeric data items. That is, a PICTURE character-string must consist of only P, S, V, and 9.
    a. BINARY
    b. COMPUTATIONAL or COMP
    c. COMPUTATIONAL-5 or COMP-5
    d. PACKED-DECIMAL, COMPUTATIONAL-3, or COMP-3
4. The following paired words are synonyms. Either one can be written.
    a. COMPUTATIONAL and COMP
    b. COMPUTATIONAL-1 and COMP-1
    c. COMPUTATIONAL-2 and COMP-2
    d. COMPUTATIONAL-3 and COMP-3
    e. COMPUTATIONAL-5 and COMP-5

5. Elementary items which specified the USAGE IS INDEX clause and elementary items depending on a group item which specified the USAGE IS INDEX clause can be written in the following places:

   a. SEARCH statement

   b. SET statement

   c. DISPLAY statement

   d. Relation condition

   e. Header of the procedure division or USING phrase of the ENTRY statement

   f. USING phrase of the CALL statement

6. The following clauses must not be written in a data description entry which contains the USAGE IS INDEX clause or a data description entry which depends on the data description entry:

   a. BLANK WHEN ZERO clause

   b. CHARACTER TYPE clause

   c. JUSTIFIED clause

   d. PICTURE clause

   e. PRINTER POSITION clause

   f. SYNCHRONIZED clause

   g. VALUE clause

7. A condition-name data description entry must not associated with an elementary item which specified the USAGE IS INDEX clause.

8. The USAGE IS INDEX clause must not be specified in a data item in the constant section.

9. An elementary item whose USAGE clause is specified with BIT or an elementary item depending on a group item whose USAGE clause is specified with BIT must be a BOOLEAN item. That is, a PICTURE character-string must consist of only 1's.

10. The following clauses must not be written in a data description entry whose USAGE clause is specified with COMPUTATIONAL-1, COMP-1, COMPUTATIONAL-2, or COMP-2 or a data description entry which depends on the data description entry:

    a. BLANK WHEN ZERO clause

    b. CHARACTER TYPE clause

    c. JUSTIFIED clause

    d. PICTURE clause

    e. PRINTING POSITION clause

    f. VALUE clause

11. A USAGE IS POINTER clause can be written only in a data description entry of the based-storage, working-storage, constant, or linkage section.

12. The following clauses must not be written in a data description entry which contains a USAGE IS POINTER clause or a data description entry which depends on the data description entry:

    a. BLANK WHEN ZERO clause

    b. CHARACTER TYPE clause

    c. JUSTIFIED clause

    d. PICTURE clause

    e. PRINTING POSITION clause

    f. SIGN clause

13. A pointer data item can be written only in a data-name of a BASED ON clause in the data division.

14. A pointer data item can be written only in the following places in the procedure division:

    a. Header of the procedure division or a USING phrase of an ENTRY statement

    b. USING phrase of the CALL statement

    c. DISPLAY statement

    d. MOVE statement

    e. Relation condition of an IF statement and an EVALUATE statement

General Rules

1. If the USAGE clause is specified in a group item, it is applied to all elementary item which depend on the group item.

2. The USAGE clause specifies the usage of a data item. The representation mode of a data item in storage area is determined by the description of the USAGE clause.

3. Internal representation of a numeric data item is determined by the USAGE clause. The following table lists numeric data item contents.

USAGE Clause		SIGN Clause	PICTURE Clause	Value	Internal Representation	Remarks
DISPLAY (zoned decimal)		None	9(4)	1234	31323334	3: Zone bit 4: Positive operation sign 5: Negative operation sign
			S9(4)	+1234	31323344	
				-1234	31323354	
		LEADING	S9(4)	+1234	41323334	
				-1234	51323334	
		TRAILING	S9(4)	+1234	31323344	
				-1234	31323354	
		LEADING SEPARATE	S9(4)	+1234	2B31323334	2B: + of standard data format 2D: - of standard data format
				-1234	2D31323334	
		TRAILING SEPARATE	S9(4)	+1234	313233342B	
				-1234	313233342D	
BINARY, COMP (binary)		---------------	9(4)	1234	04D2	The leftmost bit indicates an operational sign: 0: positive 1: negative A negative value is represented by twos complement
			S9(4)	+1234	04D2	
				-1234	FB2E	
COMP-5 (binary)	DS HP Sun	---------------	9(4)	1234	04D2	The leftmost bit indicates an operational sign: 0: positive 1: negative A negative value is represented by twos complement
			S9(4)	+1234	04D2	
				-1234	FB2E	
	Win	---------------	9(4)	1234	D204	The leftmost bit in the rightmost byte indicates an operational sign: 0: positive 1: negative A negative value is represented by twos complement
			S9(4)	+1234	D204	
				-1234	2EFB	
PACKED-DECIMAL (packed decimal)		---------------	9(4)	1234	01234F	F: Operational sign indicating absolute value C: Positive operational sign D: Negative operational sign
			S9(4)	+1234	01234C	
				-1234	01234D	

-: Combination which cannot be specified

The internal representation of a Boolean item is determined by the description of the USAGE clause. The following table lists Boolean item contents.

USAGE Clause	**PICTURE Clause**	**Value**	**Internal Representation**	**Remarks**
DISPLAY (External BOOLEAN)	1(8)	11100101	3131313030313031	Indicated by one character of 0 or 1
BIT (Internal BOOLEAN)	1(8)	11100101	E5	Indicated by one bit of 0 or 1

5. If the USAGE clause is not specified in an elementary item or a group item which influences the elementary item, a USAGE IS DISPLAY clause is assumed. A data item to which a USAGE IS DISPLAY clause is effective is called "data item whose usage is display."

6. The USAGE IS DISPLAY clause indicates that a data item is represented in the standard data format in the storage device and a data item is adjusted to the character boundary. Contents of a data item with the USAGE IS DISPLAY clause specified explicitly or implicitly are handled in a unit of character position, or byte. The size of storage occupied by a data item in the standard data format is called "the number of character positions." The number of character positions is equal to the number of bytes.

7. The USAGE IS DISPLAY clause can be specified in alphabetic, alphanumeric, alphanumeric edited, numeric edited, numeric, national language, national edited, Boolean, and external floating-point data items. If the USAGE IS DISPLAY clause is specified in a group item, it is applied to these elementary items among those which depend on a group item.

8. A numeric data item which specified the USAGE IS DISPLAY clause explicitly or implicitly is called "zoned decimal item." Digit position of each numeric character in the zoned decimal item is represented in a unit of character position, or byte.

9. If symbol S is specified in a PICTURE character-string of the zoned decimal item, an operational sign is represented as follows. If a SIGN clause is omitted, a SIGN IS TRAILING clause is assumed.

   a. If a SIGN IS LEADING clause is specified, an operational sign is included in upper four bits of the leading digit position.

   b. If a SIGN IS TRAILING clause is specified, an operational sign is included in lower four bits of the last digit position.

   c. If a SIGN IS LEADING SEPARATE CHARACTER clause is specified, + or - in the standard data format is placed in the character position directly before the digit position.

   d. If a SIGN IS TRAILING SEPARATE CHARACTER clause is specified, + or - in the standard data format is placed in the character position directly after the digit position.

10. A Boolean item which specified the USAGE IS DISPLAY clause explicitly or implicitly is called “external Boolean item.” Each Boolean position of an external Boolean item contains 1 or 0 in the standard data format.

11. A numeric data item which specified any of the USAGE IS COMPUTATIONAL clause, the USAGE IS COMPUTATIONAL-5 and USAGE IS BINARY clauses explicitly or implicitly is called “binary item.” The USAGE IS COMPUTATIONAL clause is a synonym of the USAGE IS BINARY clauses.

DS Sun HP The USAGE IS COMPUTATIONAL-5 clause is also a synonym of the above two clauses.

Win The USAGE IS COMPUTATIONAL-5 clause is different in internal expression.

The size of the storage area allocated to a binary item is determined as follows by the number of digits specified by a PICTURE clause.

1 to 4 digits: 2 bytes

5 to 9 digits: 4 bytes

10 to 18 digits: 8 bytes

12. An elementary item which specified the USAGE IS INDEX clause explicitly or implicitly is called “index data item.” An index data item has the value corresponding to the occurrence number of table elements. The size of an index data item is 4 bytes. In a data item, the value corresponding to the index-name is stored as it is by a SET statement and other statements.

13. If the USAGE IS INDEX clause is specified in a group item, all elementary items which depend on the group item become index data items. However, the group item which specified the USAGE IS INDEX clause is not an index data item.

14. If a group item which influences an index data item is specified in a MOVE statement or input-output statement, the index data item is not converted.

15. A numeric data item which specified the USAGE IS PACKED-DECIMAL clause explicitly or implicitly is called an “packed decimal item.” In an packed decimal item, each byte other than the last byte contains a two-digit number and the lower 4 bits of the last byte contain an operational sign.

16. A Boolean item which specified the USAGE IS BIT clause explicitly or implicitly is called an “internal Boolean item.” One Boolean character of an internal Boolean item is represented by one bit of 1 or 0.

17. An elementary item which specified the USAGE IS COMPUTATIONAL-1 or COMP-1 explicitly or implicitly is called a “single-precision internal floating-point data item.” Its length is 4 bytes.

18. An elementary item which specified the USAGE IS COMPUTATIONAL-2 or COMP-2 explicitly or implicitly is called a “double-precision internal floating-point data item.” Its length is 8 bytes.

19. An elementary item which specified the USAGE IS COMPUTATIONAL-1, COMP-1, COMPUTATIONAL-2, or COMP-2 explicitly or implicitly is called an “internal floating-point data item.”

20. An elementary item which specified the USAGE IS POINTER clause explicitly or implicitly is called a “pointer-data data item.” Its length is 4 bytes. A pointer-data data item can hold the value which indicates the address in the storage area.

VALUE Clause

The VALUE clause gives the initial value to a data item or specifies the value of condition-name.

[Format 1] Gives the initial value to a data item.

$$\underline{\text{VALUE}}\ \text{IS literal-1}$$

[Format 2] Specifies the value of condition-name.

$$\left\{\begin{matrix}\underline{\text{VALUE}}\ \ \text{IS}\\ \underline{\text{VALUES}}\ \ \text{ARE}\end{matrix}\right\}\left\{\text{literal-2}\ \left[\left\{\begin{matrix}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{matrix}\right\}\text{literal-3}\right]\right\}\ldots$$

Syntax Rules

1. THRU is a synonym of THROUGH.
2. If the VALUE clause is specified in a numeric data item or in condition-name of conditional variable of a numeric data item, literal-1 to literal-3 must be numeric literal. In this case, the value of numeric literal must be in the range of the value indicated by a PICTURE clause of the numeric data item.
3. If the VALUE clause is specified in a signed numeric data item or in condition-name of conditional variable which is a signed numeric data item, literal-1 to literal-3 must be signed numeric literal.
4. If nonnumeric literal, national nonnumeric literal, or Boolean literal is specified in literal-1 to literal-3, the value must not exceed the one specified in the PICTURE clause.

5. The following table lists examples of values which can be specified in the VALUE clause of a data item whose PICTURE clause contains P:

PICTURE Character-string	Values Specified in the VALUE Clause
999PP	0, 100, 200, ..., 99900
PP999	0, .00001, .00002, ..., .00999

6. The VALUE clause must not be written in the following data description entries. However, the VALUE clause can be written in condition-name data description entry associated with the following data description entries.

   a. A data description entry which contains an EXTERNAL clause or a data description entry which depends on the data description entry

   b. A data description entry which contains a REDEFINES clause or a data description entry which depends on the data description entry.

7. A THROUGH phrase must not be specified in condition-name whose conditional variable is a BOOLEAN item.

General Rules

Rules Common to Format 1 and Format 2

1. The VALUE clause must not conflict with other clauses written in a data description entry which contains the VALUE clause or written in a data description entry which influences the data description entry.

2. If the VALUE clause is specified in a numeric data item, the literal of the VALUE clause must be numeric literal. For a numeric data item of the working-storage section or constant section, the literal of the VALUE clause is stored according to the standard alignment rules.

3. If the VALUE clause is specified in an alphabetic, alphanumeric, alphanumeric edited, numeric edited, or group item, the literal of the VALUE clause must be nonnumeric literal.

4. If the VALUE clause is specified in a National data item or national edited data item, the literal of the VALUE clause must be national nonnumeric literal.

5. If the VALUE clause is specified in a BOOLEAN item, a literal of the VALUE clause must be BOOLEAN literal.

6. If the VALUE clause is specified in an internal floating-point data item, the literal of the VALUE clause must be floating-point literal, numeric literal, or figurative constant zero.

7. If the VALUE clause is specified in a pointer data item, the literal of the VALUE clause must be figurative constant zero. The value of the figurative constant zero is numeric 0.

8. If the VALUE clause is specified in an alphanumeric edited data item or numeric edited data item, the literal of the VALUE clause must be specified in the edited form.

9. If a BLANK WHEN ZERO or JUSTIFIED clause is specified in a data item which specified the VALUE clause, the functions of these clauses are ignored.

Rules for Format 1

1. The data item in which the VALUE clause can be specified varies as follows depending on the section in which the data item is defined:

   a. For the file section, the VALUE clause can be written only in a condition-name data description entry. Initial values of the data item in the file section are undefined.

b. For the linkage section, the VALUE clause can be written only in a condition-name data description entry.

c. For the working-storage section, the VALUE clause can be written in a condition-name data description entry or in a data description entry. The VALUE clause in a data description entry of the working-storage section is effective only when the program is set in the initial state. A data item which specified the VALUE clause is initialized by the value of a literal specified in the VALUE clause. The initial value of a data item which omitted the VALUE clause is not defined.

d. For the constant section, the VALUE of a data item must be given by specifying the VALUE clause.

2. If the VALUE clause is specified in a group item, the literal of the VALUE clause must be figurative constant or nonnumeric literal. In the area of a group item, the initial value is stored regardless of each elementary item and group item which depend on the group item.

3. The VALUE clause must not be specified in a data item that depends on the group item which specified the clause.

4. The VALUE clause must not be specified in a group item which influences a data item which specified a JUSTIFIED clause, SYNCHRONIZED clause, or USAGE clause other than USAGE IS DISPLAY.

5. If the VALUE clause is specified in a variable occurrence data item, group item which influences the variable occurrence data item, or data item which depends on the variable occurrence data item, the size of the variable occurrence data item is regarded as the maximum size of the variable occurrence data item (the value of integer-2 of TO integer-2 TIMES in an OCCURS clause), and the initial value is stored.

At this time, even if the VALUE clause has been specified in a data item specified in the DEPENDING ON phrase of the variable occurrence data item, the value does not effect on initialization of the variable occurrence data item.

6. If the VALUE clause is specified in a data item which specified the OCCURS clause or a data item depending on the data item, the value specified in the VALUE clause is stored in each repetition of a related data item.

Rules for Format 2

1. A VALUE clause of format 2 can be written only in a condition-name data description entry.
2. In a condition-name data description entry, condition-name and a VALUE clause of format 2 must be written. Character of condition-name is determined by conditional variable.
3. If the THRU phrase is written, literal-2 must be less than literal-3.

BASED ON Clause

Implicit pointer qualifier is specified in a data item of the based-storage section.

[Format]

```
BASED ON data-name-1
```

Syntax Rules

1. The BASED ON clause can be written only in a data description entry of level-number 01 or 77 in the based-storage section.
2. A REDEFINES clause must not be written in a data description entry which contains the BASED ON clause.
3. data-name-1 must be a pointer data item defined in the working-storage or linkage section.
4. data-name-1 can be qualified.

General Rules

1. If a data item which specified the BASED ON clause or a data item depending on the data item is referenced without explicit pointer, the value indicating the area which can be referenced must be set in data-name-1.
2. If a data item which specified the BASED ON clause is written in the object of a REDEFINES clause, the same BASED ON clause is assumed in the subject of the REDEFINES clause. In the same way, if a data item which specified the BASED ON clause is written in the object of a REDEFINES clause, the same BASED ON clause is assumed in the subject of the REDEFINES clause.

Screen Data Description Entry

A screen data description entry defines a screen item. It defines mapping and display attributes when a screen item is displayed on screen, the size of a screen item, and input-output attributes. It can be written in the screen section.

There are two kinds of screen items: elementary screen items and group screen items. “elementary screen items” are the various independent areas on the screen. A “group screen item” is a set of several elementary items.

There are four kinds of elementary screen items depending on the input-output attribute: “literal item,” “input item,” “output item,” and “update item.”

[Format 1] Defines a group screen item.

```
level-number
    [ data-name-1 ]
    [ FILLER      ]
    [AUTO clause]
    [BACKGROUND-COLOR clause]
    [BLANK SCREEN clause]
    [FOREGROUND-COLOR clause]
    [FULL clause]
    [REQUIRED clause]
    [SECURE clause]
    [SIGN clause]
    [USAGE clause].
```

[Format 2] Defines a literal item.

```
level-number
    ⎡data-name-1 ⎤
    ⎣FILLER      ⎦
    [BACKGROUND-COLOR clause]
    [BELL clause]
    ⎡BLANK LINE clause   ⎤
    ⎣BLANK SCREEN clause ⎦
    [BLINK clause]
    [COLUMN NUMBER clause]
    [ERASE clause]
    [FOREGROUND-COLOR clause]
    ⎡HIGHLIGHT clause ⎤
    ⎣LOWLIGHT clause  ⎦
    [LINE NUMBER clause]
    [REVERSE-VIDEO clause]
    [UNDERLINE clause]
    VALUE clause.
```

[Format 3] Defines an input, output, or update item.

```
level-number
    ⎡data-name-1⎤
    ⎣FILLER     ⎦
    [AUTO clause]
    [BACKGROUND-COLOR clause]
    [BELL clause]
    ⎡BLANK LINE clause  ⎤
    ⎣BLANK SCREEN clause⎦
    [BLANK WHEN ZERO clause]
    [BLINK clause]
    [COLUMN NUMBER clause]
    [ERASE clause]
    [FOREGROUND-COLOR clause]
    [FULL clause]
    ⎡HIGHLIGHT clause⎤
    ⎣LOWLIGHT clause ⎦
    [JUSTIFIED clause]
    [LINE NUMBER clause]
    [PICTURE clause]
    [REQUIRED clause]
    [REVERSE-VIDEO clause]
    [SECURE clause]
    [SIGN clause]
    [UNDERLINE clause]
    [USAGE clause].
```

Syntax Rules

1. A screen data description entry can be written only in the screen section.
2. Level-number must be either 01 to 49
3. If data-name-1 or FILLER is written, it must directly follow level-number. Other clauses can be written in optional order.
4. If data-name-1 or FILLER is omitted, FILLER is assumed.
5. If level-number is 01, data-name-1 must be written.
6. A group screen item can influence group screen items or elementary screen items. A group screen item must be defined with format 1. An elementary screen item must be defined with format 2 or format 3.
7. In format 2 and format 3, at least one of the BELL, BLANK LINE, BLANK SCREEN, COLUMN NUMBER, and LINE number clauses must be written.
8. A clause specified in a group screen item can be specified in a screen item which depends on the group screen item. In this case, a clause specified in a screen item at the lowest level in the hierarchical structure is effective.
9. A screen item can be written only in an ACCEPT statement or DISPLAY statement of the screen handling module.

General Rules

1. Distinction of an input, output, or update item of format 3 is specified by a PICTURE clause.
2. If a SECURE clause is specified in a group screen item, the SECURE clause is applied to all input items which depend on the group screen item.
3. If one of the following clauses is specified in a group screen item, the clause is applied to all input items and update items which depend on the group screen item:
    a. AUTO clause
    b. BACKGROUND-COLOR clause
    c. BLANK SCREEN clause
    d. FOREGROUND-COLOR clause
    e. FULL clause
    f. REQUIRED clause
4. Do not make a specification in a LINE NUMBER clause or COLUMN NUMBER clause that may lead to overlapping of areas on the screen. Also, do not make a specification that exceeds the limits of the physical screen.

AUTO Clause

The AUTO clause automates movement of the cursor to input items and update items.

[Format]

AUTO

Syntax Rule

The AUTO clause can be specified in an elementary or group screen item other than a literal item.

General Rules

1. If the AUTO clause is specified in a group screen item, the AUTO clause is applied to all input and update items which depend on the group screen item.
2. During execution of an ACCEPT statement for a group screen item, as soon as the last character is input for an input or output item for which the AUTO clause is effective, the cursor automatically moves to the next input or update item.
3. The AUTO clause does not carry meaning when a DISPLAY statement is executed.

BACKGROUND-COLOR Clause

The BACKGROUND-COLOR clause specifies the background color of a screen item.

[Format]

BACKGROUND-COLOR IS integer-1

Syntax Rules

1. The BACKGROUND-COLOR clause can be specified in an optional screen item.
2. The value of integer-1 can be from 0 to 7.

General Rules

1. If the BACKGROUND-COLOR clause is specified in a group screen item, it is applied to all elementary screen items which depend on the group screen item.
2. The BACKGROUND-COLOR clause is effective only for a color screen.
3. Integer-1 specifies the value which indicates the background color of a screen item. The following table lists background colors corresponding to the value of integer-1 table.

Integer-1	**Background Color**
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	White

4. If the BACKGROUND-COLOR clause is omitted, the background color is black.
5. The BACKGROUND-COLOR clause is effective when DISPLAY and ACCEPT statements are executed. A screen item with an effective BACKGROUND-COLOR clause is displayed in the color specified in integer-1.
6. If a screen item with effective BLANK SCREEN and BACKGROUND-COLOR clauses is displayed with a DISPLAY statement, the default value of the background color changes to the color specified with the BACKGROUND-COLOR clause.

BELL Clause

The BELL clause specifies the sounding of the audiotone.

[Format]

```
    BELL
```

Syntax Rule

The BELL clause can be specified only in an elementary screen item.

General Rule

When a screen item which specified the BELL clause is displayed by execution of a DISPLAY statement, the audiotone sounds.

BLANK LINE Clause

The BLANK LINE clause specifies clearing of a display line before a screen item is displayed.

[Format]

<u>BLANK</u> <u>LINE</u>

Syntax Rule

The BLANK LINE clause can be specified only in an elementary screen item.

General Rules

1. When a screen item which specified the BLANK LINE clause is to be displayed by a DISPLAY statement, the display line of the screen item is cleared from the left end to the right end before the screen item is displayed.
2. The BLANK LINE clause does not carry meaning when an ACCEPT statement is executed.

BLANK SCREEN Clause

The BLANK SCREEN clause specifies clearing of the entire screen before a screen item is displayed.

[Format]

<u>BLANK</u> <u>SCREEN</u>

Syntax Rule

The BLANK SCREEN clause can be specified in an optional screen item.

General Rules

1. If the BLANK SCREEN clause is specified in a group screen item, it is applied to all elementary items which depend on the group screen item.
2. If a screen item with an effective BLANK SCREEN clause is to be displayed by a DISPLAY statement, the entire screen is cleared before the screen item is displayed. At this time, the cursor is set to the first column of the first line.
3. If the BLANK SCREEN and BACKGROUND-COLOR clauses are used together, the default value of the background color can be changed. For the default value of the background color, see the section titled “BACKGROUND-COLOR clause.”
4. If the BLANK SCREEN and FOREGROUND-COLOR clauses are used together, the default value of the foreground color can be changed. For the default value of the foreground color, see the section titled “FOREGROUND-COLOR clause.”

5. The BLANK SCREEN clause is ignored when an ACCEPT statement is executed.

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies displaying of spaces when the value of a screen item is zero.

[Format]

$$\underline{\text{BLANK}}\ \underline{\text{WHEN}}\ \left\{\begin{array}{l}\underline{\text{ZERO}}\\ \underline{\text{ZEROS}}\\ \underline{\text{ZEROES}}\end{array}\right\}$$

Syntax Rules

1. The BLANK WHEN ZERO clause can be specified only in a numeric or numeric edited elementary screen item.
2. If the BLANK WHEN ZERO clause is specified in a numeric screen item, the usage of the numeric screen item must be display.
3. ZERO, ZEROS, and ZEROES are synonyms.

General Rules

1. When the value of a screen item is zero, the BLANK WHEN ZERO clause specifies to display the screen item with space(s) instead of zero(s).
2. If the BLANK WHEN ZERO clause is specified in a numeric screen item, the category of the item is regarded as numeric edited.
3. Even if the BLANK WHEN ZERO clause is specified in an input item, the specification is ignored.

4. The BLANK WHEN ZERO clause is ignored when an ACCEPT statement is executed.

BLINK Clause

The BLINK clause specifies blinking of screen item contents.

[Format]

```
    BLINK
```

Syntax Rule

The BLINK clause can be specified only in an elementary item.

General Rules

1. When a screen item which specified the BLINK clause is to be displayed by execution of a DISPLAY statement, the screen area corresponding to the screen item blinks.
2. The BLINK clause does not carry meaning when an ACCEPT statement is executed.

COLUMN NUMBER Clause

The COLUMN NUMBER clause specifies the column where a screen item is mapped.

[Format]

$$\underline{\text{COLUMN}}\ \text{NUMBER IS}\ [\underline{\text{PLUS}}] \left\{ \begin{array}{l} \text{identifier-1} \\ \text{integer-1} \end{array} \right\}$$

Syntax Rules

1. The COLUMN NUMBER clause can be specified only in an elementary screen item.
2. identifier-1 must be an unsigned integer item.

General Rules

1. The COLUMN NUMBER clause specifies the display line number of a screen item when the screen item is displayed or input.
2. If PLUS is omitted, the column number of the area where a screen item is mapped is set to identifier-1 or integer-1. The meaning of the value set to identifier-1 or integer-1 depends on the description of an ACCEPT statement or DISPLAY statement. If the value of identifier-1 or integer-1 is n, the screen item is mapped to one of the following places:

   a. If AT COLUMN NUMBER phrase is omitted in a DISPLAY statement or ACCEPT statement, the first screen column operated by the statement is regarded as the first column of the physical screen. Therefore, the screen item which specified the COLUMN NUMBER clause is mapped to the nth column of the physical screen.

   b. When an AT COLUMN NUMBER phrase is written in a DISPLAY statement or ACCEPT statement, if the value of the column number specified in the AT COLUMN NUMBER phrase is x, the first column of the screen operated by the statement is regarded as the xth column of the physical screen. Therefore, the screen item which specified the COLUMN NUMBER clause is mapped to the n+xth column of the physical screen.

3. If PLUS is written, the relative column number from the right end of the previous screen item (screen item preceding the screen item which specified the COLUMN NUMBER clause) is set to identifier-1 or integer-1. The set value is used as the relative column number from the right end of the preceding screen item regardless of whether the preceding screen item is operated by an ACCEPT statement or DISPLAY statement.

4. A screen item which specified the COLUMN NUMBER clause with PLUS cannot be specified as the first screen item of a screen operated by an ACCEPT statement or DISPLAY statement.

5. The following clauses are assumed in a screen item which omitted the COLUMN NUMBER clause:

   a. If a LINE NUMBER clause is specified in the screen item, “COLUMN NUMBER 1” is assumed.

   b. If a LINE NUMBER clause is not specified in the screen item, when there is a preceding screen item, “COLUMN NUMBER PLUS 1” is assumed.

ERASE Clause

The ERASE clause specifies clearing part of a line or screen before a screen item is displayed.

[Format]

$$\underline{\text{ERASE}} \left\{ \begin{array}{l} \underline{\text{EOL}} \\ \underline{\text{EOS}} \end{array} \right\}$$

Syntax Rule

The ERASE clause can be specified only in an elementary screen item.

General Rules

1. When a screen item which specified the ERASE clause is to be displayed by a DISPLAY statement, one of the following screen areas is cleared before the screen item is displayed:

   a. If the ERASE EOL clause is specified, the display line of the screen item is cleared from the first column of the display area of the screen item to the right end of the line.

   b. If the ERASE EOS clause is specified, the screen is cleared from the first column of the display line of the screen item to the end of the screen.

2. If one of the following screen items is specified in a DISPLAY statement, only the area on the screen corresponding to the screen item is changed by the DISPLAY statement. Other areas on the screen remain the same as before execution of the DISPLAY statement.

   a. When a group screen item is specified in a DISPLAY statement and an elementary screen item with an effective BLANK clause (BLANK SCREEN clause or BLANK LINE clause) and ERASE clause does not depend on the group screen item.

   b. When an elementary screen item with an ineffective BLANK clause (BLANK SCREEN clause or BLANK LINE clause) and ERASE clause is specified in a DISPLAY statement

3. The ERASE clause does not carry meaning when an ACCEPT statement is executed.

FOREGROUND-COLOR Clause

The FOREGROUND-COLOR clause specifies the foreground color of a screen item.

[Format]

<u>FOREGROUND-COLOR</u> IS integer-1

Syntax Rules

1. The FOREGROUND-COLOR clause can be specified in an optional screen item.
2. The value of integer-1 must be from 0 to 7.

General Rules

1. If the FOREGROUND-COLOR clause is specified in a group screen item, it is applied to all elementary screen items which depend on the group screen item.

2. The FOREGROUND-COLOR clause is effective only for a color screen.

3. integer-1 specifies the value which indicates the foreground color of a screen item. The following table lists foreground colors corresponding to the value of integer-1.

Integer-1	**Foreground Color**
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	White

4. If the FOREGROUND-COLOR clause is omitted, the foreground color is white.

5. The FOREGROUND-COLOR clause is effective when DISPLAY and ACCEPT statements are executed. A screen item with an effective FOREGROUND-COLOR clause is displayed in the color specified in integer-1.

6. If a screen item with effective BLANK SCREEN and FOREGROUND-COLOR clauses is displayed with a DISPLAY statement, the default value of the foreground color changes to the color specified with the FOREGROUND-COLOR clause.

FULL Clause

The FULL clause specifies pressing input of the data of a screen item.

[Format]

FULL

Syntax Rules

1. The FULL clause can be specified in an elementary screen item other than literal item or in a group screen item.
2. If the FULL clause is specified in an elementary screen item, there must be no JUSTIFIED clause.

General Rules

1. If the FULL clause is specified in a group screen item, the FULL clause is applied to all input items and update items which depend on the group screen item.
2. When the input operation of a screen item with an effective FULL clause is made by an ACCEPT statement, pressing of the input key is ignored and the cursor is placed at the head of the screen item until the following data is input in the screen item.
    a. If the screen item is an alphanumeric screen item or alphanumeric edited screen item, until characters other than a space are input in the first and last character positions of the screen item area or until the entire screen item area is filled with spaces.

   b. If the screen item is a National screen item or National edited screen item, until national nonnumeric characters other than a space are input in the first and last national nonnumeric character positions of the screen item area or until the entire screen item area is filled with National spaces.

   c. If the screen item is a numeric screen item or numeric edited screen item, until numeric characters are input in all digit positions of the screen item. For a numeric edited screen item which specified the zero suppression, however, there is no need to input zeros above the significant digits.

3. If the function key is depressed to complete the input operation, rules for the FULL clause are not applied.

4. The FULL clause does not carry meaning when a DISPLAY statement is executed.

HIGHLIGHT Clause

The HIGHLIGHT clause specifies highlighting of a screen item.

[Format]

```
        HIGHLIGHT
        ---------
```

Syntax Rule

The HIGHLIGHT clause can be specified only in an elementary screen item.

General Rules

1. The HIGHLIGHT clause is effective when a DISPLAY statement and ACCEPT statement are executed. A screen item which specified the HIGHLIGHT clause is highlighted.
2. If a BACKGROUND-COLOR clause is effective for a screen item which specified the HIGHLIGHT clause, the background color of the screen item is highlighted. In the same way, if the FOREGROUND-COLOR clause is effective for a screen item which specified the HIGHLIGHT clause, the foreground color of the screen item is highlighted.

JUSTIFIED Clause

The JUSTIFIED clause specifies alignment of the data to the right end of a screen item.

[Format]

$$\left\{\begin{array}{l}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{array}\right\}\ \text{RIGHT}$$

Syntax Rules

1. The JUSTIFIED clause can be specified only in an alphabetic, alphanumeric, or National screen item.
2. JUST is a synonym of JUSTIFIED.

General Rules

1. The JUSTIFIED clause specifies the method of adjustment when data is transferred from a data item to a screen item. If the JUSTIFIED clause is specified in a screen item, data in the sending area is stored adjusted to the right end of the screen item. If the sizes of the sending data item and screen item are different, the following processing is done when the data is transferred:
    a. If the sending data item is larger than the screen item, the leftmost characters of the sending data item are truncated to fit the screen item.
    b. If the sending data item is smaller than the screen item, the unused character positions at the left end of the receiving data item are filled with spaces.

2. If the JUSTIFIED clause is omitted, the standard alignment rules for storing data in an elementary item are applied.
3. Even if the JUSTIFIED clause is specified in an input item, it is ignored.
4. The JUSTIFIED clause does not carry meaning when an ACCEPT statement is executed.

LINE NUMBER Clause

The LINE NUMBER clause specifies the line to which a screen item is mapped.

[Format]

$$\underline{\text{LINE}}\ \text{NUMBER IS}\ [\underline{\text{PLUS}}] \begin{Bmatrix}\text{identifier-1}\\ \text{integer-1}\end{Bmatrix}$$

Syntax Rules

1. The LINE NUMBER clause can be specified only in an elementary screen item.
2. The identifier-1 must be an unsigned integer item.

General Rules

1. The LINE NUMBER clause specifies the display line number of a screen item when the screen item is displayed or input.

2. If PLUS is omitted, the column number of the area to which a screen item is mapped is set to identifier-1 or integer-1. The meaning of the value set to identifier-1 or integer-1 depends on the description of an ACCEPT statement or DISPLAY statement. If the value of identifier-1 or integer-1 is n, the screen item is mapped to one of the following places:

   a. If AT LINE NUMBER phrase is omitted in the DISPLAY statement or ACCEPT statement, the first line of the screen operated by the statement is regarded as the first line of the physical screen. Therefore, the screen item which specified the LINE NUMBER clause is mapped to the nth line of the physical screen.

   b. When AT LINE NUMBER phrase is written in the DISPLAY statement or ACCEPT statement, if the value of the line number specified in the AT LINE NUMBER phrase is x, the first line of the screen operated by the statement is regarded as the xth line of the physical screen. Therefore, the screen item which specified the LINE NUMBER clause is mapped to the n+xth line of the physical screen.

3. If PLUS is written, the relative line number from the line number of the previous screen item (screen item which preceding the screen item which specified the LINE NUMBER clause) is set to identifier-1 or integer-1. The set value is used as the relative line number from the line number of the preceding screen item whether or not the preceding screen item is operated by an ACCEPT statement or DISPLAY statement.

4. The LINE NUMBER clause with PLUS cannot be specified as the first screen item.

5. A screen item which omitted the LINE NUMBER clause is mapped in the following places:

   a. If there is no preceding screen item, the screen item is mapped to the first line of the physical screen.

   b. If there is a preceding screen item, the screen item is mapped to the same line number as the preceding screen item.

LOWLIGHT Clause

The LOWLIGHT clause specifies dimming of a screen item.

[Format]

<u>LOWLIGHT</u>

Syntax Rule

The LOWLIGHT clause can be specified only in an elementary screen item.

General Rules

1. The LOWLIGHT clause is effective when a DISPLAY statement and ACCEPT statement are executed. A screen item which specified the LOWLIGHT clause is dimmed.
2. In the system which has only two intensities, standard intensity and low intensity are the same.

PICTURE Clause

The PICTURE clause specifies the category of an elementary screen item and edit format and associates a data item for interchanging the data with the area on the screen with the elementary item.

[Format]

$$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}\text{ IS character-string-1}$$

$$\left\{\begin{array}{l}\underline{\text{USING}}\text{ identifier-1}\\ \left[\underline{\text{FROM}}\left\{\begin{array}{l}\text{indentifier-2}\\ \text{literal-1}\end{array}\right\}\right]\ [\underline{\text{TO}}\text{ identifier-3}]\end{array}\right\}$$

Syntax Rules

1. A PICTURE clause can be specified only in an elementary screen item other than a literal item. The PICTURE clause and VALUE clause are mutually exclusive.
2. The identifier-1 to identifier-3 must be data items defined in the file, working-storage, linkage, or based-storage section. A data item defined in the constant section can also be specified in identifier-2.

3. One of following must be written in a PICTURE clause:
    a. USING phrase only
    b. FROM phrase only
    c. TO phrase only
    d. FROM phrase and TO phrase
4. PICTURE phrase, USING phrase, FROM phrase and TO phrase can be written in any order. Another clause can be written between PICTURE phrase and USING phrase, between PICTURE phrase and FROM phrase, or between PICTURE phrase and TO phrase.
5. PIC is a synonym of PICTURE.

General Rules

1. The category of an elementary screen item is specified by a combination of character-string-1. There are seven categories of an elementary screen item: alphabetic, numeric, alphanumeric, alphanumeric edited, numeric edited, national language, and National edited. Each category of elementary screen items is called “alphabetic screen item”, “numeric screen item”, “alphanumeric screen item”, “alphanumeric edited screen item”, “numeric edited screen item”, “national screen item”, and “national edited screen item.” Rules for character-string-1 are the same as those of a PICTURE character-string of a data description entry. For details, see the section titled “PICTURE clause.” However, if a numeric screen item is defined, the usage of the item is display and S must be specified in character-string-1.
2. If data is transferred from a data item to a screen item and displayed on the screen, a FROM phrase must be written. A screen item which specified only a FROM phrase is called “output item.”

3. If data is input from the screen and stored in another data item by a screen item, a TO phrase must be written. An item which specified only a TO phrase is called “input item.”
4. If data is transferred from a data item to a screen item, edited on the screen, and stored in another data item, both FROM phrase and TO phrase must be written. If data is transferred to a data item, updated on the screen, and stored in the same data item, a USING phrase must be written. A screen item which specified both FROM phrase and TO phrase or a screen item which specified a USING phrase is called “update item.”
5. The number of character positions of identifier-1, identifier-2, identifier-3, and literal-1 does not need to be the same as the number of character positions of the related screen item.
6. Transfer between a screen item and identifier-1, identifier-2, identifier-3, or literal-1 is done according to the rules for transfer.
7. A FROM phrase carries meaning only when a DISPLAY statement is executed.
8. A TO phrase carries meaning only when an ACCEPT statement is executed.
9. A USING phrase carries meaning when an ACCEPT statement or DISPLAY statement is executed.

REQUIRED Clause

The REQUIRED clause specifies that input in a screen item is required.

[Format]

REQUIRED

Syntax Rule

The REQUIRED clause can be specified in an elementary screen item or group screen item other than a literal item.

General Rules

1. If the REQUIRED clause is specified in a group screen item, it is applied to all input and update items which depend on the group screen item.
2. When the input operation of a screen item with an effective REQUIRED clause is made by an ACCEPT statement, pressing the input key is ignored and the cursor is placed at the head of the screen item until the following data is input in the screen item:
   a. If the screen item is an alphanumeric screen item or alphanumeric edited screen item, until one or more characters other than a space are input.
   b. If the screen item is a National screen item or National edited screen item, until one or more national nonnumeric characters other than a space are input.

   c. If the screen item is a numeric screen item or numeric edited screen item, until the value other than zero is input.

3. If a function key is pressed to complete an input operation, the rules for the REQUIRED clause do not apply.

4. The REQUIRED clause does not carry meaning when a DISPLAY statement is executed.

REVERSE-VIDEO Clause

The REVERSE-VIDEO clause specifies switching foreground color and background color of a screen item.

[Format]

<u>REVERSE-VIDEO</u>

Syntax Rule

The REVERSE-VIDEO clause can be specified only in an elementary screen item.

General Rule

The REVERSE-VIDEO clause carries meaning when DISPLAY statements and ACCEPT statements are executed. A screen item which specified the REVERSE-VIDEO clause is displayed with foreground color and background color reversed.

SECURE Clause

The SECURE clause specifies non-display of an input item.

[Format]

SECURE

Syntax Rule

The SECURE clause can be specified only in an input item or group screen item.

General Rules

1. If the SECURE clause is specified in a group screen item, it is applied to all input items which depend on the group screen item.
2. When an input operation of a screen item with an effective SECURE clause is done by an ACCEPT statement, input characters for the screen item are not displayed on the screen. At this time, the cursor remains at the head of the screen item.

SIGN Clause

The SIGN clause specifies the position and representation mode of an operational sign of a screen item.

[Format]

$$[\underline{\text{SIGN}}\ \text{IS}]\left\{\begin{array}{l}\underline{\text{LEADING}}\\ \underline{\text{TRAILING}}\end{array}\right\}[\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

Syntax Rule

The SIGN clause can be specified in an elementary screen item or group screen item other than a literal item. Other syntax rules are the same as those of the SIGN clause in a data description entry. For details, see the section titled “SIGN clause.”

General Rule

General rules are the same as those of the SIGN clause in a data description entry. For details, see the section titled “SIGN clause.”

UNDERLINE Clause

The UNDERLINE clause specifies underlining in a screen item.

[Format]

UNDERLINE

Syntax Rule

The UNDERLINE clause can be specified only in an elementary screen item.

General Rule

The UNDERLINE clause carries meaning when DISPLAY and ACCEPT statements are executed. A screen item which specified the UNDERLINE clause is displayed underlined.

USAGE Clause

The USAGE clause specifies the usage of a screen item.

[Format]

[USAGE IS] DISPLAY

Syntax Rules

1. The USAGE clause can be specified in an elementary item or group item other than a literal item.
2. If the USAGE clause is specified in a group screen item, it is also specified in an elementary screen item or group screen item which depends on the group screen item.

General Rules

1. If the USAGE clause is specified in a group screen item. it is applied to all elementary items which depend on the group screen item. An elementary item whose USAGE clause is effective is called “screen item for display.”
2. The USAGE clause specifies the representation format of a data item on the screen. The USAGE IS DISPLAY clause indicates that the representation format of a data item is standard data format.
3. If the USAGE clause is not specified in a group screen item, the usage of all elementary items which depend on the group screen item is assumed to be display. Also, the usage of an elementary screen item which omitted the USAGE clause is assumed to be display.
4. Other general rules are the same as those of the USAGE clause in a data description entry. For details, see the section titled “USAGE clause.”

VALUE Clause

The VALUE clause specifies the value of a literal item.

[Format]

VALUE IS literal-1

Syntax Rules

1. The VALUE clause can be specified only in a literal item. The VALUE clause and PICTURE clause must not be specified at the same time.
2. The literal-1 must be a nonnumeric literal, hexadecimal nonnumeric literal, or national nonnumeric literal.

General Rule

Integer-1 specifies the value of a literal item, that is the value to be displayed on the screen.

Report Description Entry

A report description entry defines report-name, layout of each page of the report, and a control data item. A report description entry can be written only in the report section.

[Format]

```
      RD  report-name-1
  [CODE clause]
  [CONTROL clause]
  [PAGE clause].
```

Syntax Rules

1. LEVEL indicator RD must be written at the beginning of a report description entry, followed by report-name-1.
2. report-name-1 must be written only in the REPORT clause.
3. The order of clauses which follow report-name-1 is optional.
4. The highest level identifier for the line-counter, page-counter, and data-name defined in the report section is report-name-1.

Special Register

If a report description entry is written, PAGE-COUNTER and LINE-COUNTER (special registers) are generated automatically. Special register PAGE-COUNTER is called “page-counter” and LINE-COUNTER is called “line-counter.”

Rules for Page Counter

1. The PAGE-COUNTER can be written in the following places:
   a. A SOURCE clause of a report group description entry of the report section.
   b. A place where a data item having an integer value can be written in the procedure division.
2. If two or more report description entry are written in a program, the PAGE-COUNTER must be written as follows:
   a. If the PAGE-COUNTER is written in the procedure division, it must be qualified with report-name.
   b. The PAGE-COUNTER can be written in a SOURCE clause without qualifying. If the PAGE-COUNTER is not qualified, it is qualified implicitly with report-name associated with a report group which contains the SOURCE clause. If the PAGE-COUNTER in another report is referenced, it must be qualified explicitly with report-name written in the report description entry.
3. The value is set to the PAGE-COUNTER by the report writer control system as follows:
   a. If an INITIATE statement is executed, 1 is set to the PAGE-COUNTER.
   b. The value of the PAGE-COUNTER is incremented by 1 whenever the report writer control system feeds a form.
4. The value of the PAGE-COUNTER can be changed by a statement of the procedure division.

Rules for LINE-COUNTER

1. The LINE-COUNTER can be written only in the following places:
    a. SOURCE clause of a report group description entry of the report division.
    b. A place where a data item having an integer value can be written in the procedure division.
2. A value cannot be set to the LINE-COUNTER. The value of the LINE-COUNTER is set by the report writer control system.
3. If two or more report description entry are written in a program, the LINE-COUNTER must be written as follows:
    a. If the LINE-COUNTER is written in the procedure division, it must be qualified with report-name.
    b. The LINE-COUNTER can be written in a SOURCE clause without qualifying. If the LINE-COUNTER is not qualified, it is qualified implicitly with report-name of a report description entry associated with a report group which contains the SOURCE clause. If the LINE-COUNTER in another report is referenced, it must be qualified explicitly with the report-name.
4. The value is set to the LINE-COUNTER by the report writer control system as follows:
    a. If an INITIATE statement is executed, 0 is set to the LINE-COUNTER.
    b. 0 is set to the LINE-COUNTER whenever the report writer control system feeds a form.
5. Even if a report group which does not have a printable item is processed or a group whose printing is suppressed by a SUPPRESS statement is processed, the value of the LINE-COUNTER does not change.

6. When each print line is displayed, the value of the LINE-COUNTER indicates the line number where the print line is displayed. The value of the LINE-COUNTER after display of a report group is determined by presentation rules for each report group. For details of the presentation rules for a report group, see the section titled “Presentation rules for report groups.”

CODE Clause

The CODE clause specifies a literal appended to a print line of a report.

[Format]

<u>CODE</u> literal-1

Syntax Rules

1. literal-1 must be a two-character nonnumeric literal.
2. If the CODE clause is specified in one of the reports in a certain file, it must be specified for all other reports in the file.

General Rules

1. If the CODE clause is specified, two characters of literal-1 are automatically appended to the left end of each record of the report.
2. With this function, when several kinds of reports are registered in one output unit at the same time, individual reports can be selected for generation. Do not use the CODE clause when reports are directly sent to a line printing device.

CONTROL Clause

The CONTROL clause specifies the control hierarchy of a report.

[Format]

$$\left\{\begin{array}{l}\underline{\text{CONTROL}}\text{ IS}\\ \underline{\text{CONTROLS}}\text{ ARE}\end{array}\right\}\left\{\begin{array}{l}\{\text{data-name-1}\}\ldots\\ \underline{\text{FINAL}}\ [\text{data-name-1}]\ldots\end{array}\right\}$$

Syntax Rules

1. data-name-1 must not be the one defined in the report section.
2. Occurrences of data-name-1 must not be the same data item.
3. Variable occurrence data items must not depend on data-name-1.

General Rules

1. data-name-1 and FINAL specify the control hierarchy. FINAL is the highest control. data-name-1 is a large control data item, the next occurrence of data-name-1 is a middle control data item, and the last occurrence of data-name-1 is a small control data item.
2. A data item specified in data-name-1 is called “control data item.” Changing value of the data item is called “control break.”

If a GENERATE statement is executed first for a certain report, the report writer control system saves the values of all control data items related to the report. When the next and after GENERATE statements are executed for the report, the report writer control system checks the change of the value of the control data item. Even if only a value of the control data item is changed, control break occurs. Control break is associated with the highest level where the change of the value is recorded.

3. The report writer control system checks control break by comparing contents of the control data item with previous contents of the control data item which was saved when during execution of the previous GENERATE statement for the same report. The report writer control system performs one of the following inequality comparison checks:

    a. If a control data item is a numeric data item, the check is comparison of two numeric operands.

    b. If a control data item is an index data item, the check is comparison of two index data items.

    c. If a control data item is National data item or national edited data item, the check is comparison of two National operands.

    d. If a control data item is not (a), (b), or (c), the check is comparison of two nonnumeric operands.

4. If there is no control data item corresponding to the most comprehensive control group in a report, FINAL is specified.

PAGE Clause

The PAGE clause specifies the page length and the vertical range of displaying a report group.

[Format]

$$\underline{\text{PAGE}} \begin{bmatrix} \text{LIMIT IS} \\ \text{LIMITS ARE} \end{bmatrix} \text{Integer-1} \begin{bmatrix} \text{LINE} \\ \text{LINES} \end{bmatrix}$$

$$[\underline{\text{HEADING}} \text{ integer-2}]\ [\underline{\text{FIRST}}\ \underline{\text{DETAIL}} \text{ integer-3}]$$

$$[\underline{\text{LAST}}\ \underline{\text{DETAIL}} \text{ integer-4}]\ [\underline{\text{FOOTING}} \text{ integer-5}]$$

Syntax Rules

1. The order of writing HEADING (limit of heading) phrase, FIRST DETAIL (upper limit of a detail report group) phrase, LAST DETAIL (lower limit of a detail report group) phrase, and FOOTING (limit of footing) is optional.
2. integer-1 must be up to 3 significant digits.
3. integer-2 must be 1 or more.
4. integer-3 must be larger than integer-2.
5. integer-4 must be larger than integer-3.
6. integer-5 must be larger than integer-4.
7. integer-1 must be larger than integer-5.

8. The following is the vertical range of a page in which a report group is indicated when the PAGE clause is written.

    a. If a report heading report group is displayed in one page, it is displayed in the range of lines from integer-2 to integer-1. If a report heading report group is not displayed in one page, it is displayed in the range of lines from integer-2 to the line above integer-3.

    b. If a page heading report group is specified, it is displayed in the range of lines from integer-2 to integer-3.

    c. If a control heading group or detail report group is specified, it is displayed in the range of lines from integer-3 to integer-4.

    d. If a control footing group is specified, it is displayed in the range of lines from integer-3 to integer-5.

    e. If a page footing report group is specified, it is displayed in the range of lines from the value of integer-5 plus 1 to integer-1.

    f. If a report footing report group is displayed in one page, it is displayed in the range of lines from integer-2 to integer-1. If a report footing report group is not displayed in one page, it is displayed in the range of lines from integer-5 plus 1 to integer-1.

9. An entire report group must be described to be displayed in one page. The report writer control system does not display a multiple-line report group that continues onto another page.

General Rules

1. The vertical layout of a page of a report is determined by the integer value specified by the PAGE clause. Meaning of each integer is as follows:

   a. integer-1 specifies effective lines of each page to determine the size of a page of a report.

   b. integer-2 of HEADING specifies the first line number to display a report heading report group or page heading report group.

   c. integer-3 of FIRST DETAIL specifies the first line number to display a body group. A report heading report group (without NEXT GROUP NEXT PAGE) is not displayed on or beyond the line number specified by integer-3.

   d. integer-4 of LAST DETAIL specifies the last line number to display a control heading group or detail report group.

   e. integer-5 of FOOTING specifies the last line number to display a control footing group. A report footing report group (without LINE integer-1 NEXT PAGE) and page footing group are displayed after the line number specified by integer-5.

2. If the PAGE clause is specified and also phrases in the PAGE clause are omitted, the following values are assumed:

   a. If a HEADING phrase is omitted, integer-2 is assumed 1.

   b. If a FIRST DETAIL phrase is omitted, integer-3 is assumed to be the same as integer-2.

   c. Both of a LAST DETAIL and FOOTING phrases are omitted, integer-4 and integer-5 are assumed to be the same as integer-1.

   d. If a FOOTING phrase is specified and LAST DETAIL phrase is omitted, integer-4 is assumed to be the same as integer-5.

   e. If a LAST DETAIL phrase is specified and FOOTING phrase is omitted, integer-5 is assumed to be the same as integer-4.

3. If the PAGE clause is omitted, a report consists of one page of undetermined length.

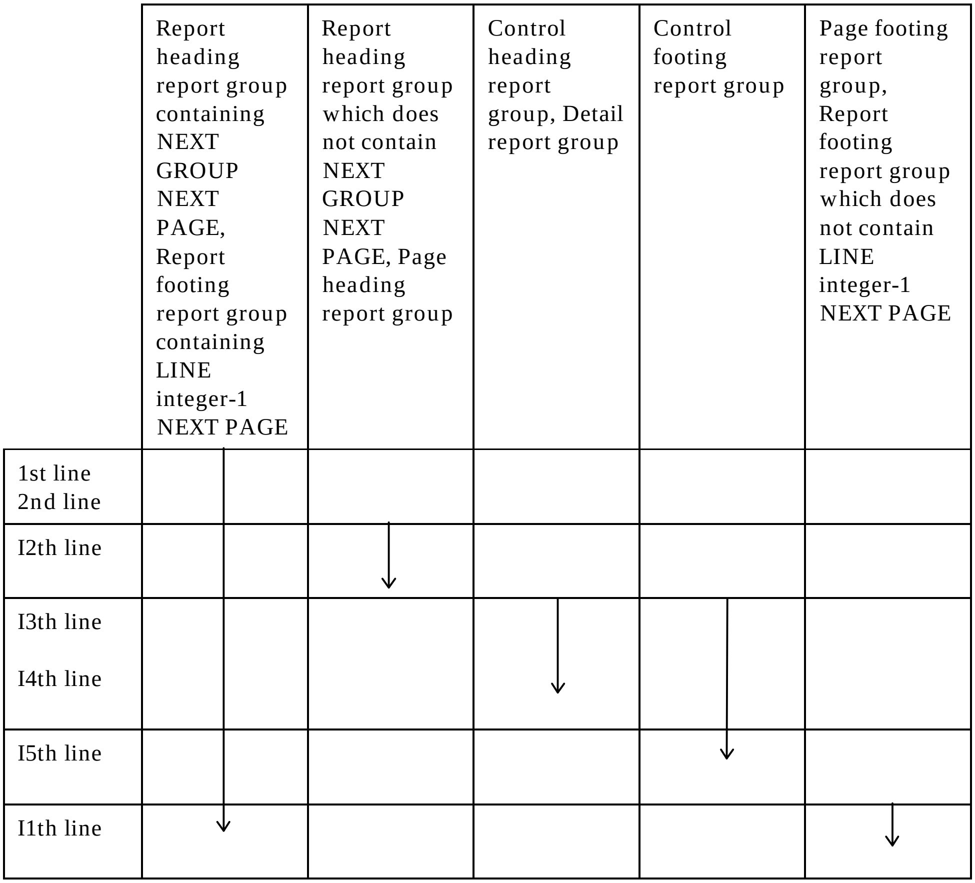
4. For report group presentation rules, see the section titled “Report group presentation rules.”

Page Area

The following table lists page areas set by the PAGE clause.

Report Group Indicated in the Area	**First Line Number in the Area**	**Last line Number in the Area**
Report heading report group containing NEXT GROUP NEXT PAGE, Report footing report group containing LINE integer-1 NEXT PAGE	integer-2	integer-1
Report heading report group which does not contain NEXT GROUP NEXT PAGE, Page heading report group	integer-2	(integer-3) - 1
Control heading group, Detail report group	integer-3	integer-4
Control footing group	integer-3	integer-5
Page footing group, Report footing report group which does not contain LINE integer-1 NEXT PAGE	(integer-5) + 1	integer-1

The following figure shows the page area.


Report heading report group containing NEXT GROUP NEXT PAGE, Report footing report group containing LINE integer-1 NEXT PAGE
Report heading report group which does not contain NEXT GROUP NEXT PAGE, Page heading report group
Control heading report group, Detail report group
Control footing report group
Page footing report group, Report footing report group which does not contain LINE integer-1 NEXT PAGE
1st line
2nd line
I2th line
I3th line
I4th line
I5th line
I1th line

I1: Value of integer-1 of the PAGE clause
I2: Value of integer-2 of the HEADING phrase
I3: Value of integer-3 of the FIRST DETAIL phrase
I4: Value of integer-4 of the LAST DETAIL phrase
I5: Value of integer-5 of the FOOTING phrase

Report Group Description Entry

A report group description entry defines character of a report group which depends on a report and character of each item in the report group. A report group description entry can be written only in the report section.

A report group description entry follows a report description entry. A report group description entry consists of layers 1, 2, and 3.

[Format 1] Report group description entry of the first layer

```
01        [data-name-1]
          [CHARACTER TYPE clause]
          [LINE NUMBER clause]
          [NEXT GROUP clause]
           TYPE clause
          [USAGE clause].
```

[Format 2] Report group description entry of the second layer

```
level-number        [data-name-1]
                    [CHARACTER TYPE clause]
                    [LINE NUMBER clause]
                    [USAGE clause].
```

[Format 3] Report group description entry of the third layer

```
level-number  [data-name-1]
              [BLANK WHEN ZERO clause]
              [CHARACTER TYPE clause]
              [ COLUMN NUMBER clause     ]
              [ PRINTING POSITION clause ]
              [GROUP INDICATE clause]
              [JUSTIFIED clause]
              [LINE NUMBER clause]
              PICTURE clause
              [SIGN clause]
              [USAGE clause]
              { SOURCE clause }
              { SUM clause    } .
              { VALUE clause  }
```

Syntax Rules

1. If data-name-1 is written at the beginning of a report group description entry, data-name-1 must directly follow level-number. The order of clauses other than data-name-1 is optional.
2. Level-number of format 2 must range from integer 02 to 48. Level-number of format 3 must range from integer 02 to 49.

3. A report group description entry consists of layers 1, 2 and 3. Rules for appending a layer are shown below:
    a. The first entry of a report group description entry must be format 1.
    b. An entry of format 2 or format 3 can be written immediately at the lower level of an entry of format-1.
    c. If an entry of format 2 is written, at least one entry of format 3 must be written immediately at the lower level.
    d. An entry of format 3 must be the one which defines an elementary item.
4. data-name-1 of format 1 is written only in the following cases:
    a. When a detail report group is referenced by a GENERATE statement
    b. When a detail report group is referenced by an UPON phrase of a SUM clause
    c. When a report group is referenced by a USER BEFORE REPORTING sentence.
    d. When the name of a control footing group is used for qualifying sum counter If data-name-1 is written, it is referenced by a GENERATE statement, UPON phrase of a SUM clause, USE BEFORE REPORTING statement, or qualifier of sum counter only.
5. At least one clause must be written in an entry of format 2.
6. If data-name-1 is written with format 2, it can be used only for qualifying sum counter.

7. A USAGE clause is written in the report section only for declaring the usage of a printable item.

   a. If a USAGE clause is written in an entry of format 3, the entry must be the one which defines a printable item.

   b. If a USAGE clause is written in an entry of format 1 or 2, a printable item must be defined in at least one of entries which depend on the entry.

8. An entry which specified a LINE NUMBER clause must not influence an entry which specified a LINE NUMBER clause.

9. An item defined in an entry of format 3 is called “printable item.” The following rules must be observed in format 3:

   a. A GROUP INDICATE clause can be written only in a detail report group.

   b. A SUM clause can be written only in a control footing group.

   c. An entry which contains a COLUMN NUMBER clause or PRINTING POSITION clause and does not contain a LINE NUMBER clause must depend on an entry which contains a LINE NUMBER clause.

   d. data-name-1 can be written in any entry. However, it can be referenced only when the entry defines sum counter.

   e. Either a COLUMN NUMBER clause or PRINTING POSITION clause must be written in an entry which specified a VALUE clause.

   f. An entry containing a COLUMN NUMBER clause and an entry containing a PRINTING POSITION clause must not be in one report group at the same time.

10. The following five combinations of clauses are allowed in an entry of format 3:

Clause	**Combination of Clauses**				
	1	**2**	**3**	**4**	**5**
PICTURE clause	Necessary	Necessary	Necessary	Necessary	Necessary
COLUMN NUMBER clause or PRINTING POSITION clause	-	Necessary	Applicable	Applicable	Necessary
SOURCE clause	-	-	Necessary	Necessary	-
SUM clause	Necessary	Necessary	-	-	-
VALUE clause	-	-	-	-	Necessary
JUST clause	-	-	Applicable	-	Applicable
BLANK WHEN ZERO clause	-	Applicable	-	Applicable	-
GROUP INDICATE clause	-	-	Applicable	Applicable	Applicable
USAGE clause	-	Applicable	Applicable	Applicable	Applicable
SIGN clause	Applicable	Applicable	Applicable	Applicable	Applicable
LINE clause	Applicable	Applicable	Applicable	Applicable	Applicable

Necessary: The clause must always be written.

Applicable: The clause is optional.

-: The clause must not be written.

General Rules

1. Format 1 is a report description entry. A report group is defined by the report description entry and all entries which depend on the entry.
2. BLANK WHEN ZERO, JUSTIFIED, and PICTURE clauses in a report group description entry are the same as BLANK WHEN ZERO, JUSTIFIED, and PICTURE clauses in a data description entry, respectively. For rules for these clauses, see the section titled “Data description entry.”
3. If two or more report group description entries of level-number 01 are written, different areas are allocated to each area.

CHARACTER TYPE Clause

The CHARACTER TYPE clause specifies format of characters when printing.

[Format 1] Specifies the character of National data item or national edited data items.

$$[\underline{\text{CHARACTER}}\ \underline{\text{TYPE}}\ \text{IS}]\ \left\{\begin{array}{l}\underline{\text{MODE-1}}\\ \underline{\text{MODE-2}}\\ \underline{\text{MODE-3}}\end{array}\right\}\ [\text{BY mnemonic-name-1}]$$

[Format 2] Specifies the character size, pitch, font, rotation amount, and from of National data items or national edited data items.

$$\underline{\text{CHARACTER}}\ \underline{\text{TYPE}}\ \text{IS mnemonic-name-2}$$

Syntax Rules

1. The CHARACTER TYPE clause can be specified only in a group item which influences National data items and national edited data items or a group item which influences national edited data items.
2. If the CHARACTER TYPE clause is specified in a group item, it is applied to all National data items and national edited data items which depend on the group item.
3. If the CHARACTER TYPE clause is specified both in a group item and a data item which depends on the group item, CHARACTER TYPE clauses having the same meaning must be specified.
4. mnemonic-name-1 must be associated with the following function-name in the special-names paragraph of the environment division:

   HSC, F0202, H0202, F0102, F0201
5. If mnemonic-name-1 is associated with the following function-name, only MODE-1 or MODE-2 can be specified.

   HSC, H0202
6. mnemonic-name-2 must be associated with function-name in the special-names paragraph of the environment division.

$$\text{XY-}\left\{\begin{array}{c}\text{12P}\\ \text{9P}\\ \text{7P}\end{array}\right\}\left[\ \text{-}\left\{\begin{array}{c}\text{12}\\ \text{21}\\ \text{22}\end{array}\right\}\ \right]\ \text{or}$$

$$\text{YX-}\left\{\begin{array}{c}\text{12P}\\ \text{9P}\end{array}\right\}\text{[-22]}\ \ \text{-H}$$

General Rules

General rules for the CHARACTER TYPE clause in a report group description entry is the same as those of format 1 and 2 in a data description entry. For details, see the section titled “CHARACTER TYPE clause.”

COLUMN NUMBER Clause

The COLUMN NUMBER clause specifies column on a print line.

[Format]

```
    COLUMN NUMBER IS integer-1
```

Syntax Rules

1. The COLUMN NUMBER clause can be written only in an elementary item in a report group. If the COLUMN NUMBER clause is written, it must be written in an entry which contains a LINE NUMBER clause or an entry which depends on the entry.
2. In one print line, printable items must be written from left to right sequentially. Columns of printable items in a line must not be overlapped.

General Rules

1. The COLUMN NUMBER clause indicates that sum counter defined by the object of a SOURCE or VALUE clause, or a SUM clause is displayed on a print line. If the COLUMN NUMBER clause is not written, the item is not displayed on the print line.
2. integer-1 indicates the leftmost column of a printable item.
3. The report writer control system puts spaces in all columns of a print line which is not occupied by a printable item.
4. The leftmost column of a print line is 1.

GROUP INDICATE Clause

The GROUP INDICATE clause specifies indicating a detail report group in which control break or form feed occurs.

[Format]

<u>GROUP</u> INDICATE

Syntax Rule

The GROUP INDICATE clause can be written only in a detail report group which defines a printable item.

General Rules

1. The GROUP INDICATE clause specifies that this printable item is indicated only for the first appearing detail report group after control break or form feed.
2. If the GROUP INDICATE clause is specified, a SOURCE clause and VALUE clause are ignored and padding with spaces is done except for the following cases:
   a. When a detail report group in the report is indicated in the first time
   b. When a detail report group is indicated in the first time after form feed
   c. When a detail report group is indicated in the first time after control break
3. If a PAGE clause and CONTROL clause are not written in a report description entry, a printable item for which the GROUP INDICATE clause is specified is indicated when this detail report group is indicated in the first time after an INITIATE statement is executed.

   Then, the item which contains a SOURCE clause or VALUE clause is padded with spaces.

LINE NUMBER Clause

The LINE NUMBER clause specifies the vertical position of a report group.

[Format]

$$\underline{\text{LINE}}\ \text{NUMBER IS} \left\{ \begin{array}{l} \text{integer-1 [ON } \underline{\text{NEXT}}\ \underline{\text{PAGE}}\text{]} \\ \underline{\text{PLUS}}\ \text{integer-2} \end{array} \right\}$$

Syntax Rules

1. integer-1 and integer-2 must be up to 3 significant digits. Neither integer-1 nor integer-2 must be used to indicate a printable item to the outside of the vertical range of a page defined by a PAGE clause for this report group type.

   Integer-2 can specify zero.

2. An entry which contains the LINE NUMBER clause must not depend on an entry which contains the LINE NUMBER clause.

3. The absolute LINE NUMBER clause must be written before the relative LINE NUMBER clause in one report group description entry.

4. A series of integers of the absolute LINE NUMBER clause must be in ascending order. They do not need to be consecutive.

5. If a PAGE clause is omitted, only the relative LINE NUMBER clause can be specified in a report group description entry of the report.

6. A NEXT PAGE phrase can be written only once in one report group description entry. If it is written, it must be used by the first LINE NUMBER clause.
7. The LINE NUMBER clause with a NEXT PAGE phrase can be written only in an entry of a body report group and report footing report group.
8. An entry which defines a printable item must contain the LINE NUMBER clause or depend on an entry which contains the LINE NUMBER clause.
9. The first LINE NUMBER clause of a page footing group must be the absolute LINE NUMBER clause.

General Rules

1. The LINE NUMBER clause must be written to specify line number of a print line of a report group.
2. The report writer control system does vertical positioning specified by the LINE NUMBER clause before a print line specified by the LINE NUMBER clause is indicated.
3. integer-1 specifies the absolute line number. Absolute line number indicates the line number where a print line is indicated.
4. integer-2 specifies the relative line number. If the relative LINE NUMBER clause is not the first LINE NUMBER clause of a report group description entry, the line number where a print line is indicated is calculated by adding the line number indicating the print line directly before the report group and integer-2 of the relative LINE NUMBER clause. If integer-2 is zero, a print line is indicated in the same line as the previous print line.

If the relative LINE NUMBER clause is the first LINE NUMBER clause in the report group description entry, the line number where a print line is indicated is determined according to the section titled “Report Groups Representation Rules.”

5. If a NEXT PAGE clause is written, the report group is indicated in the line number specified by the clause in a new page.

NEXT GROUP Clause

The NEXT GROUP clause specifies form feed to be done after the last line of a report group is indicated.

[Format]

$$\underline{\text{NEXT}}\ \underline{\text{GROUP}}\ \text{IS}\ \left\{\begin{array}{l}\text{integer-1}\\ \underline{\text{PLUS}}\ \text{integer-2}\\ \underline{\text{NEXT}}\ \underline{\text{PAGE}}\end{array}\right\}$$

Syntax Rules

1. The NEXT GROUP clause must not be specified in a report group description entry in which a LINE NUMBER clause is not specified.
2. integer-1 and integer-2 must be up to 3 significant digits.
3. If a PAGE clause is omitted in a report description entry, only the relative NEXT GROUP clause can be specified in a report group description entry of the report.
4. A NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a page footing group.
5. The NEXT GROUP clause must not be specified in a report footing report group or page heading report group.

General Rules

1. The line feed specified by the NEXT GROUP clause is done after indicating a report group containing the clause.
2. The report writer control system determines the new value of the line-counter by combining the information of line feed specified by the NEXT GROUP clause, the information of TYPE and PAGE clauses, and the value of the line-counter.
3. The NEXT PAGE clause written in a control footing group other than the highest level in which control break was detected is ignored.
4. The NEXT GROUP of a body group affects the position where the next body group is indicated. The NEXT GROUP clause of a report heading report group affects the position where a page heading report group is indicated. The NEXT GROUP clause of a page footing group affects the position where a report footing report group is indicated.

PRINTING POSITION Clause

The PRINTING POSITION clause specifies the absolute printing position in the horizontal direction when printing at a national processing device.

[Format]

PRINTING <u>POSITION</u> IS integer-1

Syntax Rules

When the PRINTING POSITION clause is to be specified in a National data item or national editing data item, the CHARACTER TYPE clause must be specified in the data item or in the group item which contains the data item.

General Rules

1. For integer-1, specify the printing start position of the data item to be printed with the high order end of a printable line counted as 1.
2. The printing position shown by integer-1 must be located to the right of the rightmost printing position of the previous data item in the record description entry containing that data item.
3. Some printing devices cannot accept the PRINTING POSITION clause. For the relationship between printing device and the PRINTING POSITION clause, refer to the "COBOL85 User's Guide."

SIGN Clause

The SIGN clause specifies the position of operational signs and the expression format.

[Format]

$$[\underline{\text{SIGN}}\ \text{IS}]\left\{\begin{array}{l}\underline{\text{LEADING}}\\ \underline{\text{TRAILING}}\end{array}\right\}\ \underline{\text{SEPARATE}}\ \text{CHARACTER}$$

Syntax Rules

1. The SIGN clause can be specified only in numeric data items that contain PICTURE symbol “S”.
2. The numeric data item with a SIGN clause applied must be used for explicit or implicit display.
3. When a SIGN clause is contained in a report description entry, the SEPARATE CHARACTER phrase must be written.

General Rules

1. The SIGN clause specifies the position of a sign and the expression format of a numeric data item. The SIGN clause written for a numeric data item will be applicable only to that particular item. The PICTURE symbol “S” indicates the presence of a sign, but not the expression form or the position.
2. A numeric data item with PICTURE symbol “S” but without a SIGN clause is handled in the same manner as a numeric data item with TRAILING written in the SIGN clause.

3. Since the SEPARATE CHARACTER phrase must be written in a SIGN clause in a report group description entry, and the signs should follow the rules provided below:

   a. An operational sign will be appended at the left end character position when LEADING is specified or at the right end character position when TRAILING is specified. This character position is different from the numeric position.

   b. PICTURE symbol “S” is counted in the size of this item in the standard data format.

   c. The operational signs for positive and negative are “+” and “-”, respectively. If data does not contain “+” or “-” during execution of a clause with SEPARATE CHARACTER, processing result are undefined.

4. A numeric data item with PICTURE symbol “S” is regarded as a numeric data item with a sign. When conversion for calculation or comparison is required for an item containing a SIGN clause, conversion is automatically done.

5. For the expression format for a data item with a SIGN clause specified, see the section titled “SIGN clause,” and the section titled “USAGE clause.”

SOURCE Clause

The SOURCE clause specifies a sending data item which provides values to be transferred to a printable item.

[Format]

<u>SOURCE</u> IS identifier-1

Syntax Rules

1. identifier-1 can be defined in any section of the data division. However, only the following identifiers are allowed for the report section:
   a. Page-counter
   b. Line-counter
   c. Sum counter in the same report as the SOURCE clause
2. identifier-1 specifies a sending data item of an implicit MOVE statement in order for the report writer control system to transfer data to a printable item. identifier-1 must be defined in accordance with the rules for a sending item of a MOVE statement. See the section titled “Transfer Rules,” for the transcriptions rules.
3. When a reference modification is made on identifier-1, high order end position and length of the reference modifier must be a data-name or literal.
4. When a subscript is to be appended to identifier-1, neither operator (+ or -) can be written in the subscript.

General Rules

The report writer control system creates the printable line(s) for a report group immediately before displaying the report group. At this time, it executes an implicit MOVE statement specified by the SOURCE clause. See the section titled “TYPE clause,” for the timing to display a report group.

SUM Clause

The SUM clause creates a sum counter and specifies the names of data items to be summed.

[Format]

$$\{\underline{\text{SUM}}\ \{\text{indentifier-1}\}\ \ldots\ [\underline{\text{UPON}}\ \{\text{data-name-1}\}\ \ldots]\}\ldots$$

$$\left[\underline{\text{RESET}}\ \text{ON}\left\{\begin{matrix}\text{data-name-2}\\ \underline{\text{FINAL}}\end{matrix}\right\}\right]$$

Syntax Rules

1. A data item which is the body of the report group description entry where a SUM clause appears must be defined with alphabetic characters. identifier-1 must refer to a numeric data item. When identifier-1 is in the report section, it must refer to the sum counter.

   When an UPON phrase is omitted and the identifier in the SUM clause is a sum counter, this identifier must be defined in one of the following locations:

   a. in the same control footing group as that containing this SUM clause

   b. in a control footing group in a lower control hierarchy within the same report

   When the UPON phrase is described, the identifier in the SUM clause must not be a sum counter.

2. data-name-1 must be the name of the detail report group within the same report as that containing a control footing group where the SUM clause is written. data-name-1 can be qualified with a report name.

3. A SUM clause can be written only for a control footing group.

4. data-name-2 must be one of the data-names written in the CONTROL clause in a report. data-name-2 must not be a control data name lower than that in the control hierarchy of the control footing group where a RESET phrase is written. When a FINAL is to be written in the RESET phrase, a FINAL must also be written in the CONTROL clause in this report.

5. The highest level qualifier allowed for a sum counter shall be a report name.

General Rules

1. The SUM clause specifies a sum counter. A sum counter is a numeric data item with an operational sign generated by a compiler. The size of the sum counter and decimal point are determined, according to the category of the data item specified by the report group description entry where the SUM clause has been specified. The size and decimal point are determined as follows:

   a. When the data item related is a numeric character, the size and decimal position of the sum counter will be the same as those of the data item.

   b. When the data item related is a numeric edited data item, the size of the sum counter will be the same as the digit position, and the decimal point the same as that of the related data item.

   c. When the data item related is an alphanumeric or alphanumeric edited data item, the size of the sum counter equals the size of the data item excluding the editing character or 18 characters, whichever is smaller. The sum counter is an integer.

2. The value of the data item in identifier-1 is added to the sum counter at the execution time by the report writer control system. This adding operation complies with the rules for an ADD statement.

3. Even if more than one SUM clause is written in one elementary item description entry for a report, only one sum counter is set.

4. If a SUM clause is written in the description entry for a printable item, the sum counter will be the source data item. Also, the report writer control system transfers the sum counter value to the printable item in accordance with the rules for a MOVE statement to display it.

5. A data-name can be written in an elementary item description entry containing a SUM clause. It is a sum counter name and not the name of the printable item.

6. The value of the sum counter can be modified by a statement in the procedure division.

7. The report writer control system adds the value of the data item specified by identifier-1 to the sum counter while executing GENERATE and TERMINATE statements. There are three kinds of sum operations for a sum counter: subtotaling, crossfooting, and rolling forward. Subtotaling will be executed after processing a control break while executing a GENERATE statement, and before displaying a detail report group.

   Crossfooting and rolling forward will be performed while processing a control footing group.

8. If the UPON phrase is written, subtotaling can be performed selectively for each repeated detail report group specified by identifier-1.

9. The report writer control system adds each value (repetition of identifier-1 in the SUM clause) to the sum counter in accordance with its nature. The method to add a value to a sum counter is explained below:

   a. If the addend is the sum counter value in the same control footing group, the sum operation of the value in the sum counter and the addend is called “crossfooting.”

      Crossfooting is performed when the control footing group is processed due to control break. The crossfooting operation is performed in the order such sum counters are defined in the control footing group.

      Namely, all the sum operations are performed for the first sum counter in the control footing group, then all the sum operations are performed for the second sum counter. This procedure will be repeated until all the crossfooting operations are completed.

If one of the added is the sum counter value defined in a data description entry where this SUM clause is written, the value of the sum counter immediately before this sum operation is used for this sum operation.

b. When the addend is the sum counter value in a lower level control footing group, the sum operation of the sum counter and the addend is called “rolling forward.” This operation is performed when a lower level control footing group is processed due to a control break.

c. When the addend is not a sum counter value, the addition of the addend and the sum counter is called “subtotaling.” If an UPON phrase is written in a SUM clause, the addend is added every time the GENERATE statement for a detail report group specified is executed. If any UPON phrase is not written in a SUM clause, the addend is added every time a GENERATE statement specifying a data-name for a report is executed against a report where the SUM clause is written.

10. If the same identifier is specified as the addend twice or more, the value is added to the sum counter by the number of times specified. If the same data-name is specified in the UPON phrase twice or more, the adding operation will be repeatedly performed every time the GENERATE statement that contains the data-name for the detail report group is executed and by the number of times it is specified.

11. See the section titled “GENERATE statement (creating reports),” for the subtotaling to be performed when a GENERATE statement specifying a report name is executed.

12. When a RESET phrase is omitted, the report writer control system sets “0” to the sum counter in processing the control footing group containing the sum counter. When the RESET phrase is written, the report writer control system sets “0” to the sum counter after processing the control footing group of the level where the RESET is specified. For the timing to process a report group, see the section titled “TYPE clause.”

    When a INITIATE statement is executed, the report writer control system sets all the sum counters for the report to “0”.

TYPE Clause

The TYPE clause specifies the type of the report group and the timing to display it, as shown in the following example.

[Format]

$$\underline{\text{TYPE}}\ \text{IS}\ \left\{\begin{array}{l}
\left\{\begin{array}{l}\underline{\text{REPORT}}\ \underline{\text{HEADING}}\\ \underline{\text{RH}}\end{array}\right\}\\
\left\{\begin{array}{l}\underline{\text{PAGE}}\ \underline{\text{HEADING}}\\ \underline{\text{PH}}\end{array}\right\}\\
\left\{\begin{array}{l}\underline{\text{CONTROL}}\ \underline{\text{HEADING}}\\ \underline{\text{CH}}\end{array}\right\}\ \left\{\begin{array}{l}\text{data-name-1}\\ \underline{\text{FINAL}}\end{array}\right\}\\
\left\{\begin{array}{l}\underline{\text{DETAIL}}\\ \underline{\text{DE}}\end{array}\right\}\\
\left\{\begin{array}{l}\underline{\text{CONTROL}}\ \underline{\text{FOOTING}}\\ \underline{\text{CF}}\end{array}\right\}\ \left\{\begin{array}{l}\text{data-name-2}\\ \underline{\text{FINAL}}\end{array}\right\}\\
\left\{\begin{array}{l}\underline{\text{PAGE}}\ \underline{\text{FOOTING}}\\ \underline{\text{PF}}\end{array}\right\}\\
\left\{\begin{array}{l}\underline{\text{REPORT}}\ \underline{\text{FOOTING}}\\ \underline{\text{RF}}\end{array}\right\}
\end{array}\right\}$$

Syntax Rules

1. The following terms are synonyms:
   a. REPORT HEADING and RH
   b. PAGE HEADING and PH
   c. CONTROL HEADING and CH
   d. DETAIL and DE
   e. CONTROL FOOTING and CF
   f. PAGE FOOTING and PF
   g. REPORT FOOTING and RF
2. Only one group for each of report heading group, page heading group, control heading group with the FINAL specified, control footing group with the FINAL specified, page footing group, and report footing group can be written in one report.
3. The page heading group and the page footing group can be written only when a PAGE clause is specified in the report description entry.
4. data-name-1, data-name-2 and FINAL must be specified in the CONTROL clause in the report description entry. For each data-name or FINAL specified in the CONTROL clause in the report description entry, only one control heading group and only one control footing group can be written. However, both can be omitted.
5. A SOURCE clause for a page heading group, a SOURCE clause for a page footing group, or a related USE procedure must not contain a reference to a control data item.

6. A SOURCE clause or a related USE procedure for a control footing group, a page heading group, a page footing group, and a report footing group, must not contain a reference to the following data items:
   a. A group item containing a control data item
   b. A subordinate data item of a control data item
   c. A data item which is created by redefining or renaming a part or the all of a control data item
7. Do not specify more than one detail report group in the report description entry when writing a GENERATE statement which specifies a report name in the procedure division. If a GENERATE statement specifying a data-name for this report is not written, the detail report group can be omitted.
8. A report description entry must contain at least one body group description.

General Rules

1. A detail report group is directly instructed to be created by a GENERATE statement, and processed by the report writer control system. Report groups other than detail report groups are automatically processed by the report writer control system.
2. The REPORT HEADING phrase is specified in the report group in a report to be processed only once by the report writer control system as the first report group of the report. The first report group in a report is called “report heading group.” The report heading group is processed while executing the first GENERATE statement for the report.

3. The PAGE HEADING phrase is to be specified in the report group which is to be processed by the report writer control system as the first report group of each page in the report, except those provided below. The first report group on a page of a report is called a "page heading group."

   a. When one page consists of only one report heading group or report footing group, and any page heading group is not processed

   b. When a report heading group that does not configure one page by itself has already displayed on the page, and the page heading report group is processed as the second report group

4. The CONTROL HEADING phrase is specified in the report group to be processed by the report writer control system at the beginning of the control group for a control data name. The first report group in a control group is called a "control heading group."

   If FINAL is specified, it is executed during processing of the first GENERATE statement for the report. When the report writer control system detects a control break during execution of a GENERATE statement, processes all control heading groups in all levels below the highest level.

5. The DETAIL phrase is specified in the report group to be processed by the report writer control system while executing the corresponding GENERATE statement. A report group which has a TYPE clause specified in the DETAIL phrase is called a "detail report group." A detail report group, a control heading group, and a control footing group are collectively called a "body group."

6. The CONTROL FOOTING phrase is specified in the report group to be processed by the report writer control system at the end of a control group with a control data name. The last report group in a control group is called a “control footing group.”

   If the FINAL is specified, it is processed once in a report as the last main group of the report. When the report writer control system detects a control break during execution of a GENERATE statement, it processes all control footing groups in all levels below the highest level. If the GENERATE statement is executed at least once for the report, all the control footing groups are displayed during execution of the TERMINATE statement.

7. The PAGE FOOTING phrase is specified in the report group to be processed by the report writer control system as the last report group of each page within the report except the cases provided below. The last report group on a page within a report is called a “page footing group.”

   a. When one page is configured by only one report heading group or report footing group, and any page footing group is not processed

   b. When a report footing group that does not configure one page by itself has already been displayed at the end of the page, and the page footing group is to be processed as the second report group

8. The REPORT FOOTING phrase is specified in a report group, which is processed once by the report writer control system, as the last report group in the report. The last report group in the report is called a “report footing group.” If the GENERATE statement is executed at least once for the report, all the control footing groups are displayed while executing the TERMINATE statement.

9. The report writer control system processes a report heading group, a page heading group, a control heading group, a page footing group, and a report footing group in the following order:

    a. If any USE BEFORE REPORTING procedure which refers to a data-name in the report group exists, the USE procedure is executed.

    b. If a SUPPRESS statement is executed or the report group is for printing, a printable line is created and report groups are displayed in compliance with the report group presentation rules.

    c. If a SUPPRESS statement is not executed, and the report group is for printing, a printing line is created and the report groups are displayed in compliance with the report group presentation rules.

10. The report writer control system processes the control footing groups in the following order:

    a. Calculates the totals of the sum counters cross-wise. Namely the sum counters defined in this report group and the operand of the SUM clause in the same report group are added to the main body sum counter in the SUM clause.

    b. The sum counters are summed by rolling forward. Namely, all the sum counters defined in this report group and the operands of the SUM clause in the higher level control footing group are added to the sum counter in that higher level control footing group.

    c. If any USE BEFORE REPORTING procedure that refers to a data-name in this report group exists, the USE procedure is executed.

    d. If a SUPPRESS statement is executed and the report group is not for printing, process (f) will be executed.

e. If a SUPPRESS statement is not executed or the report group is for printing, printable lines are created and report groups are displayed in compliance with the control footing group presentation rules.

f. In processing this level in the control hierarchy, the sum counter may be set to “0” as required.

When a control break occurs, the report writer control system creates and displays a control footing groups where control breaks were detected. This is done from the lowest level to the highest level in compliance with the GENERATE statement presentation rules. At this time, even if no control footing group has been defined for a control data item, the report writer control system performs process (f) so long as the control data name has been specified in the RESET phrase in the report description.

11. The processes to be performed by the report writer control system for a detail report group corresponding to a GENERATE statement with a data-name specified will be one of the descriptions (a) through (e) provided below.

When only one detail report group is written in the report description, the processes on the details to be performed by the report writer control system that corresponds to the GENERATE statement with a report name specified will be one of the descriptions (a) through (d). The control system processes as if a GENERATE statement with a data-name specified had been executed.

When no detail report group is written in a report description, the processes to be performed by the report writer control system for a corresponding GENERATE statement with a report name specified will be (a). The control system processes as if only a GENERATE statement with a data-name specified had been executed, where only one detail report group was written in the report description.

a. Executes all the subtotaling operations, specified in the detail report group.

b. If there is a USE BEFORE REPORTING procedure that references a data-name in this report group, the system executes the USE procedure.

c. If the SUPPRESS statement has been executed, or if the report group is not for printing, the system terminates the process for the report group.

d. If the process for the detail report group depends on the GENERATE statement, the system terminates the process for the report group.

e. If neither (c) nor (d), the report writer control system creates a printable line, and displays a report group in compliance with the detail report group presentation rules.

12. While processing a control heading group, control footing group, or detail report group in compliance with general rules (9) and (10), the report writer control system may execute a page break process. Also, the control system may execute the following page footing group and page heading report group processes before actually displaying the body group. The control system does this by suspending display of the body group after deciding to display that group.

13. The values obtained by referencing to a control data item are as shown below. However, the stored value for the control data item which the report writer control system uses to detect a control break is called an old value. See the section titled “CONTROL clause,” for control data items.

    a. The old value is used if a control data item is referenced in the USE procedure or in the SOURCE clause associated, in processing a control break of a control footing group.

    b. When a TERMINATE statement is executed, the report writer control system regards that the highest level control break is detected. Then the old value is used if a control data item is referred to in the SOURCE clause in the control footing group and report footing report group or in the associated USE procedure.

    c. If a data item is referred to in the report group and in the associated USE procedure, the current value contained in the data item in processing the report group will be used, except cases (a) and (b).

USAGE Clause

The USAGE clause specifies the expression format for a data item in a storage unit.

[Format]

[USAGE IS] DISPLAY

Syntax Rules

1. A USAGE clause can be written in any data description entry.
2. When a USAGE clause is written in a data entry of a group item, a USAGE clause can be written in its elementary item or group item.
3. In the USAGE clause for a report group entry, only USAGE IS DISPLAY can be specified.

General Rules

1. If a USAGE clause is written for a group item level, it will be applied to all the elementary items within the group.
2. A USAGE clause specifies the expression format of a data item within a storage unit.
3. The USAGE IS DISPLAY clause indicates that the expression format of a data item is the standard data format.
4. A USAGE clause specified for an elementary item or a group item containing elementary items is regarded to be for display purpose.

VALUE Clause

The VALUE clause specifies the value for a printing item.

[Format]

VALUE IS literal-1

Syntax Rules

1. A signed numeric literal must correspond to the PICTURE character-string which describes the signed numeric data item.
2. A numeric literal in a VALUE clause must be a value within a range specified by the PICTURE clause. It must not cause truncation of any numeric value other than 0. A nonnumeric literal in a VALUE clause must not exceed the size indicated in the PICTURE clause.

General Rules

1. The VALUE clause must not conflict with any other clauses written in a data description for a this item or a group item containing this item. In addition, the following rules apply:
   a. When the category is numeric character, literal-1 in the VALUE clause must be a numeric literal.

b. When the category is alphabetic character, alphanumeric character, alphanumeric editing data, or numeric editing data, the literal-1 in the VALUE clause must be a nonnumeric literal. This literal is handled so that the data item is described as alphanumeric character.

c. If the category is National or national edited data, the literal-1 in the VALUE clause must be a national nonnumeric literal. This literal is handled so that the data item is described as national language.

d. If the category is Boolean, literal-1 in the VALUE clause must be a Boolean literal.

e. If any editing character is included in a PICTURE clause to be counted in the size of the data item, it is ignored in giving an initial value to the data item. Therefore, the literal in the editing item should be written in the edited form.

f. In giving an initial value to an item, it will not be affected by any BLANK WHEN ZERO nor JUSTIFIED clause.

2. In a report section, a specified value is given to a printing item every time a report group is displayed if the elementary entry containing the VALUE clause does not contain a GROUP INDICATE clause. However, if the entry does contain this clause, the specified value is displayed only when the conditions specified in the “GROUP INDICATE clause” in this section are satisfied.

Report Group Presentation Rules

In this section the following items are described.

- Possible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report groups
- Requirements to use LINE NUMBER and NEXT GROUP clauses
- Processes for the LINE NUMBER and NEXT GROUP clauses to be performed by the report writer control system

How to Use Presentation Rule Tables

Presentation rule tables are provided below for each of the report heading group, the page heading group, the page footing group, and the report footing group. The detail report group, the control heading group, and the control footing group are also described in the presentation rule table of each respective body group.

The first and the second columns in the presentation rule tables show all possible combinations of the LINE NUMBER and the NEXT GROUP clauses for a given report group type. Therefore, for one combination of the LINE NUMBER and NEXT GROUP clauses, read the row in the presentation rule table from left to right.

Reading Applicable Rule Column

The applicable rule column in the presentation rule table is divided into two. The first part is applicable to the report descriptions containing a PAGE clause, while the second part is applicable to those not containing a PAGE clause. A hyphen (-) indicates that no rule is applicable. The names and purposes of the rules in the applicable rule column are as provided below:

Upper Limit and Lower Limit Rules

These rules specify the vertical range within a page where the specified report group is displayed. If no PAGE clause is contained, the report to be printed will not be split vertically. Therefore, no upper limit or lower limit rule for a report description without a PAGE clause appears in the table.

Page Break Rules

The page break rule is applicable to the body group. Thus, the page break rule only appears in the presentation rule table for the body group. The report writer control system determines whether the specified body group can be displayed on the current page of the report at the execution time by applying the page break rules. If a PAGE clause is not contained in a report description entry, the page break rules are not applicable to the body group either.

Printing Starting Line Position Rules

The printing starting line position rules specify where the report writer control system should display the first line of a report group on a report medium. The presentation rule table does not specify where the report writer control system displays the second and subsequent printing lines (if any) on the report medium. This is defined by the general rules for the LINE NUMBER clause.

Next Report Group Rules

The next report group rules specify how to use the NEXT GROUP clause.

Line-counter Final Value Setting Rules

The line-counter final value setting rules specify the final value to be set for the line-counter by the report writer control system after displaying the report group.

LINE NUMBER Clause Notation

The first column of the presentation rule table indicates the order of the LINE NUMBER clause within one report group. The symbols are defined below:

- “Absolute” means that a LINE NUMBER clause without a NEXT PAGE phrase is to be written. In the LINE NUMBER clause without a NEXT PAGE phrase, specify an absolute line position. LINE NUMBER clauses without a NEXT PAGE phrase can be written consecutively. After this, a LINE NUMBER clause with a PLUS phrase can also be written consecutively.
- “PLUS” indicates that a LINE NUMBER clause with the PLUS phrase is to be written. A LINE NUMBER clause with a PLUS phrase specifies a relative line position. LINE NUMBER clauses with PLUS can be written consecutively.
- “NEXT PAGE” indicates that a LINE NUMBER clause with a NEXT PAGE phrase is to be written. A LINE NUMBER clause with NEXT PAGE specifies an absolute line position after changing the page. Following this clause, LINE NUMBER clauses without NEXT PAGE or LINE NUMBER clause with PLUS can also be written consecutively.
- “Omit” means to omit the LINE NUMBER clause.

NEXT GROUP Clause Notation

The second column of the presentation rule table shows how to write a NEXT GROUP clause in one report group. Only one NEXT GROUP clause can be written in one report group. The meaning of the symbols is as shown below.

- “Absolute” indicates that a NEXT GROUP clause with integer-1 is to be written.
- “PLUS” indicates that a NEXT GROUP clause with PLUS is to be written.
- “NEXT PAGE” indicates that a NEXT GROUP clause with NEXT PAGE is to be written.
- “Omit” indicates that a NEXT GROUP clause is to be omitted.
- “Prohibited” indicates not to write a NEXT GROUP clause.

Integer Saving Item for Next Report Group

The integer saving item for next report group is the data item which can be processed by the report writer control system only. If an absolute line position which cannot be applicable to the current page is specified in a NEXT GROUP clause, the report writer control system sets the value to the integer saving item for next report group.

After performing the page break process, the report writer control system positions the next body group using the value set for the integer saving item for next report group.

Report Heading Group Presentation Rules

All the possible combinations of LINE NUMBER clauses and NEXT GROUP clauses for the report heading group and their presentation rules are shown in the table below.

Combination of LINE NUMBER Clause and NEXT GROUP Clause		**Applicable Rule**						
		With a PAGE Clause					**Without a PAGE Clause**	
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line position	Next report group	Line counter final value setting	Printing starting line position	Line counter final value setting
Absolute	Absolute	(1)	(2) (a)	(3) (a)	(4) (a)	(5) (a)	Invalid combination	
Absolute	PLUS	(1)	(2) (a)	(3) (a)	(4) (b)	(5) (b)	Invalid combination	
Absolute	NEXT PAGE	(1)	(2) (b)	(3) (a)	(4) (c)	(5) (c)	Invalid combination	
Absolute	Omit	(1)	(2) (a)	(3) (a)	-	(5) (d)	Invalid combination	
PLUS	Absolute	(1)	(2) (a)	(3) (b)	(4) (a)	(5) (a)	Invalid combination	
PLUS	PLUS	(1)	(2) (a)	(3) (b)	(4) (b)	(5) (b)	(3) (d)	(5) (b)
PLUS	NEXT PAGE	(1)	(2) (b)	(3) (b)	(4) (c)	(5) (c)	Invalid combination	
PLUS	Omit	(1)	(2) (a)	(3) (b)	-	(5) (d)	(3) (d)	(5) (d)
Omit	Prohibited	-	-	(3) (c)	-	(5) (e)	(3) (c)	(5) (e)

See the section titled “How to use the presentation rule tables,” to learn how to understand the table. Numbers in parentheses in the table above are explained below:

1. Upper limit rule

   The first line position where a report heading group can be displayed is to be specified with the HEADING phrase in a PAGE clause.

2. Lower limit rule

    a. The last line position where a report heading group can be displayed will be the value obtained by subtracting one from integer-3 specified in FIRST DETAIL.

    b. The last line position where a report heading group can be displayed will be specified with integer-1 in the PAGE clause.

3. Printing starting line position rule

    a. The first printable line for a report heading group is displayed at the line position specified by the integer in the PAGE clause.

    b. The first printable line for a report heading group is displayed at the line position indicated by the value obtained by adding the integer in the first LINE NUMBER clause and the value subtracting one from integer-2 in the HEADING phrase in a PAGE clause.

    c. Any report heading group is not displayed.

    d. The first printable line in a report heading group is displayed at a line position obtained by adding the integer in the first LINE NUMBER clause to the line-counter value (in this case “0”).

4. Next report group rule

    a. The integer in NEXT GROUP must be greater than the line position to display the last printable line of a report heading group. Also, it must be less than the line position indicated by integer-3 in the FIRST DETAIL phrase in a PAGE clause.

    b. The value added to the integer in NEXT GROUP and the line position where the last printable line of a report heading group is to be displayed must be less than integer-3 in the FIRST DETAIL phrase in a PAGE clause.

c. The NEXT GROUP NEXT PAGE indicates that a single report heading group is displayed on the first page of the report. The report writer control system does not process other report groups while any report is positioned at the first page.

5. Line-counter final value setting rule

   a. The report writer control system sets the integer in the NEXT GROUP clause to the line-counter as the final value.

   b. The report writer control system sets the value by adding the integer in a NEXT GROUP clause and the line position displaying the last printable line of a report heading group to the line-counter as the final value.

   c. The report writer control system sets “0” to the line-counter as the final value.

   d. The report writer control system sets the line position displaying the last printable line of a report heading group to the line-counter as the final value.

   e. the line-counter will not be affected by any processing of a report group containing no printable items.

Page Heading Group Presentation Rules

All the possible combinations of LINE NUMBER clauses and NEXT GROUP clauses for a page heading group and their presentation rules are shown in the table below.

Combination of LINE NUMBER Clause and NEXT GROUP Clause		Applicable Rule				
		With a PAGE Clause (*1)				
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Printing starting line position	Next report group	Line-counter final value setting
Absolute	Prohibited	(1)	(2)	(3) (a)	-	(4) (a)
PLUS	Prohibited	(1)	(2)	(3) (b)	-	(4) (a)
Omit	Prohibited	-	-	(3) (c)	-	(4) (b)

*1 Do not define any page heading group if any PAGE clause is not contained in a report description entry.

See the section titled “How to use the presentation rule tables,” to learn how to understand the table. Numbers in parentheses in the table above are explained below:

1. Upper limit rule

   If a report heading group has already been displayed on the page where a page heading report group is to be displayed, the first line position where a page heading group can be displayed will be the value which added “1” to the final value of the line-counter set for a report heading group.

   In other cases, the first line position where a page heading group can be displayed is specified by the HEADING phrase in a PAGE clause.

2. Lower limit rule

   The last line position where a page heading group can be displayed will be the value obtained by subtracting one from integer-3 specified in FIRST DETAIL in a PAGE clause.

3. Printing starting line position rule

   a. The first printable line for a page heading group is displayed at the line position specified by the integer in the LINE NUMBER clause for the group.

   b. If a report heading group has already been displayed on the page where a page heading group is to be displayed, the first printable line for a page heading group is displayed at the line position indicated by the value obtained by adding the integer in the first LINE NUMBER clause for the page heading group to the final value of the line-counter set by the report heading group .

      In other cases, the first printable line of a page heading group is displayed at the position indicated by the value adding the integer in the first LINE NUMBER clause in the page heading group, to the value obtained by subtracting one from integer-2 in the HEADING phrase in a PAGE clause.

   c. No page heading group is displayed.

4. Line-counter final value setting rule

   a. The report writer control system sets the line position displaying the last printable line of a page heading group to the line-counter as the final value.

   b. The line-counter will not be affected by any processing of a report group not containing printable items.

Body Group Presentation Rules

All the possible combinations of LINE NUMBER clauses and NEXT GROUP clauses for the control heading group, the detail report group and the control footing group, and their presentation rules are shown in the table below.

Combination of LINE NUMBER clause and NEXT GROUP clause		Applicable rule							
		With a PAGE clause						Without a PAGE clause	
LINE NUMBER clause	NEXT GROUP clause	Upper limit	Lower limit	Page break	Printing starting line position	Next report group	Line counter final value setting	Printing starting line position	Line counter final value setting
Absolute	Absolute	(1)	(2)	(3) (a)	(4) (a)	(5)	(6) (a)	Invalid combination	
Absolute	PLUS	(1)	(2)	(3) (a)	(4) (a)	-	(6) (b)	Invalid combination	
Absolute	NEXT PAGE	(1)	(2)	(3) (a)	(4) (a)	-	(6) (c)	Invalid combination	
Absolute	Omit	(1)	(2)	(3) (a)	(4) (a)	-	(6) (d)	Invalid combination	
PLUS	Absolute	(1)	(2)	(3) (b)	(4) (b)	(5)	(6) (a)	Invalid combination	
PLUS	PLUS	(1)	(2)	(3) (b)	(4) (b)	-	(6) (b)	(4) (d)	(6) (f)
PLUS	NEXT PAGE	(1)	(2)	(3) (b)	(4) (b)	-	(6) (c)	Invalid combination	
PLUS	Omit	(1)	(2)	(3) (b)	(4) (b)	-	(6) (d)	(4) (d)	(6) (d)
NEXT PAGE	Absolute	(1)	(2)	(3) (c)	(4) (a)	(5)	(6) (a)	Invalid combination	
NEXT PAGE	PLUS	(1)	(2)	(3) (c)	(4) (a)	-	(6) (b)	Invalid combination	
NEXT PAGE	NEXT PAGE	(1)	(2)	(3) (c)	(4) (a)	-	(6) (c)	Invalid combination	
NEXT PAGE	Omit	(1)	(2)	(3) (c)	(4) (a)	-	(6) (d)	Invalid combination	
Omit	Prohibited	-	-	-	(4) (c)	-	(6) (e)	(4) (c)	(6) (e)

See the section titled “How to use the presentation rule tables,” to learn how to understand the table. Numbers in parentheses in the table above are explained below:

1. Upper limit rule

   The first line position where a body group can be displayed is to be specified with the FIRST DETAIL phrase in a PAGE clause.

2. Lower limit rule

   The last line position where a control heading group or a detail report group can be displayed is to be specified with the LAST DETAIL phrase in a PAGE clause.

   The last line position where a control footing group can be displayed is to be specified with the FOOTING phrase in a PAGE clause.

3. Page break rule

   a. If the value of the line-counter is less than the integer in the first absolute LINE NUMBER clause, the body group is displayed on the current page.

      In other cases, the report writer control system performs the page break process. If a page heading group has been defined, the report writer control system checks whether a value has been set to the integer for next report group saving item when the last body group of the previous page is displayed after the page break process. (See (6)(a).) If no value has been set to the integer saving item for next report group, the body group is displayed on the current page. If a value has been set to the integer saving item for next report group, the report writer control system transfers the value in the integer saving item for next report group to the line-counter, sets “0” to the integer saving item for next report group, and applies (3)(a) again.

b. If the body group has already been displayed on the current page of the report, the report writer control system calculates a testing total by adding the contents of the line-counter and the integers in all the LINE NUMBER clauses for the report group. If the total is less than or equal to the integer of the lower limit for the body group, the report group is displayed on the current page.

If this total exceeds the integer of the lower limit for the body group, the report writer control system performs the page break process. If a page heading report group is defined, the report writer control system applies (3)(b) again after the process.

If the body group has not been displayed on the current page of the report, the report writer control system checks whether a value has been set to the integer saving item for next report group when the last body group is displayed on the previous page. (See (6)(a).)

If no value has been set to the integer saving item for next report group, the body group is displayed on the current page of the report. If a value has been set to the integer saving item for next report group, the report writer control system transfers the value in the integer saving item for next report group to the line-counter, sets “0” to the integer saving item for next report group, and calculates a testing total.

This total is calculated by adding 1 and integers in all the LINE NUMBER clauses excluding the first one in the body group to the line-counter. If this total does not exceed the lower limit integer for the body group, the body group is displayed on the current page. If this total exceeds the lower limit integer for the body group, the report writer control system performs the page break process. If any page heading group has been defined, it displays the body group on the page after the process.

   c. If the body group has been displayed on the current page of the report, the report writer control system performs the page break process. If a page heading group has been defined, it applies (3)(c) after the process. If any body group has not been displayed on the current page of the report, the report writer control system checks whether a value has been set to the integer saving item for next report group when the last body group of the previous page is displayed after the page break process. (See (6)(a).) If no value has been set to the integer saving item for next report group, the body group is displayed on the current page. If a value has been set to the integer saving item for next report group, the re2port writer control system transfers the value in the integer for next report group saving item to the line-counter, and sets “0” to the integer saving item for next report group. Then if the value in the line-counter is less than the integer in the first mandatory LINE NUMBER clause, the body group is displayed on the current page. Otherwise, the report writer control system performs the page break process. If a page heading group has been defined, the system displays the body group on the page after the process.

4. Printing starting line position rule

   a. The first printable line in the body group is displayed at the line position specified by the integer in the LINE NUMBER clause for the group.

   b. If the value in the line-counter exceeds the line position specified in the FIRST DETAIL phrase in the PAGE clause, and no body group has been displayed on the current page, the first printable line for the current body group will be displayed on the next line of the line indicated by the value for the line-counter.

If the value of the line-counter is greater than the line position specified in the FIRST DETAIL phrase in the PAGE clause, and the body group has already been displayed on the current page, the first printable line of the current body group will be displayed on the line indicated by the value adding the line-counter value and the integer in the first LINE NUMBER clause for the current body group. If the value in the line-counter is less than the line position specified by the FIRST DETAIL phrase in the PAGE clause, the first printable line of the current body group will be displayed on the line specified in the FIRST DETAIL phrase.

c. The body group will not be displayed.

d. The addition of the value of the line-counter and the integer in the first LINE NUMBER clause specifies the line position where the first printable line is to be displayed.

5. Next report group rule

   The integer in an absolute NEXT GROUP clause shall be larger than the line position specified by the FIRST DETAIL phrase in a PAGE clause, and be smaller than the line position specified by the FOOTING phrase in the PAGE clause.

6. line-counter final value setting rule

   a. When the body group displayed immediately before is a control footing group, and if it is not the highest level among the control footing groups corresponding to the control break, the final value of the line-counter will be the line position which displayed the last printable line of the control footing group.

In all other cases, the report writer control system compares the line position displaying the last printable line of the body group, and the integer specified by the NEXT GROUP phrase. If the former is less than the latter, the report writer control system sets the integer specified by the NEXT GROUP to the line-counter as the final value. If the former is greater than the latter, the report writer control system sets the line position specified by the FOOTING phrase in the PAGE clause to the line-counter as the final value, and further, it sets the integer in the NEXT GROUP clause to the integer for next report group saving item.

b. When the body group displayed immediately before is a control footing group, and if it is not the highest level among the control footing groups corresponding to the control break, the final value of the line-counter shall be the line position which displayed the last printable line of the control footing group.

In all other cases, the report writer control system adds the integer in the NEXT GROUP clause and the line position displaying the last printable line of the body group, calculating a testing total. If the total is smaller than the line position specified in the FOOTING phrase in the PAGE clause, the report writer control system sets the total to the line-counter as the final value. If the total is greater than the line position specified in the FOOTING phrase in the PAGE clause, the report writer control system sets the line position specified in the FOOTING phrase in the PAGE clause to the line-counter as the final value.

c. When the body group displayed immediately before is a control footing group, and if it is not the highest level among the control footing groups corresponding to the control break, the final value of the line-counter will be the line position which displayed the last printable line of the control footing group.

   In all other cases, the report writer control system sets the line position specified in the FOOTING phrase in the PAGE clause to the line-counter as the final value.

d. The final value of the line-counter shall be the line position displaying the last printable line of the body group.

e. The line-counter will not be affected by any process of the body group that does not contain a printable item.

f. When the body group displayed immediately before is a control footing group, and if it is not the highest level among the control footing groups corresponding to the control break, the final value of the line-counter will be the line position which displayed the last printable line of the control footing group.

   In all other cases, the report writer control system sets the sum of the value of the line position where the last printable line was displayed and the integer in the NEXT GROUP clause to the line-counter as the final value.

Page Footing Group Presentation Rules

All the possible combinations of LINE NUMBER clauses and NEXT GROUP clauses for a page footing group and their presentation rules are shown in the following table.

Combination of LINE NUMBER clause and NEXT GROUP clause		**Applicable rule**				
		With a PAGE clause (*1)				
LINE NUMBER clause	**NEXT GROUP clause**	**Upper limit**	**Lower limit**	**Printing starting line position**	**Next report group**	**Line-counter final value setting**
Absolute	Absolute	(1)	(2)	(3) (a)	(4) (a)	(5) (a)
Absolute	PLUS	(1)	(2)	(3) (a)	(4) (b)	(5) (b)
Absolute	Omit	(1)	(2)	(3) (a)	-	(5) (c)
Omit	Prohibited	-	-	(3) (b)	-	(5) (d)

*1 Do not define any page footing group if any PAGE clause is not contained in a report description entry.

See the section titled “How to use the presentation rule tables,” to learn how to understand the table. Numbers in parentheses in the table above are explained below:

1. Upper limit rule

   The first line position where a page footing group can be displayed shall be the value adding one to integer-5 in the FOOTING phrase in a PAGE clause.

2. Lower limit rule

   The last line position where a page footing group can be displayed shall be the value specified by integer-1 in the PAGE clause.

3. Printing start line position rule

   a. The first printable line of a page footing group is to be displayed at the line position specified by the integer in the LINE NUMBER clause for the group.

   b. Any page footing group is not displayed.

4. Next report group rule

   a. The integer in the NEXT GROUP clause must be greater than the line position where the last printable line of the page footing group is displayed. Also, it must be less than the line position specified by integer-1 in the PAGE clause.

   b. The sum of the integer in the NEXT GROUP and the line position where the last printable line of the page footing group is displayed must be less than the line position specified by integer-1 in the PAGE clause.

5. line-counter final value setting rule

   a. The report writer control system sets the integer in the NEXT GROUP clause to the line-counter as the final value.

   b. The report writer control system sets the sum of the integer in the NEXT GROUP clause and the line position displayed the last printable line of a page footing group to the line-counter as the final value.

   c. The report writer control system sets the line position where the last printable line of the page footing group was displayed to the line-counter as the final value.

   d. The line-counter will not be affected by a process of a report group that does not contain a printable item.

Report Footing Group Presentation Rule

All the possible combinations of LINE NUMBER clauses and NEXT GROUP clauses for the report footing group, and their presentation rules are shown in the table below.

Combination of LINE NUMBER clause and NEXT GROUP clause		**Applicable rule**						
		With a PAGE clause					**Without a PAGE clause**	
LINE NUMBER clause	**NEXT GROUP clause**	**Upper limit**	**Lower limit**	**Printing starting line position**	**Next report group**	**Line counter final value setting**	**Printing starting line position**	**Line counter final value setting**
Absolute	Prohibited	(1) (a)	(2)	(3) (a)	-	(4) (a)	Invalid combination	
PLUS	Prohibited	(1) (a)	(2)	(3) (b)	-	(4) (a)	(3) (d)	(4) (a)
NEXT PAGE	Prohibited	(1) (b)	(2)	(3) (c)	-	(4) (a)	Invalid combination	
Omit	Prohibited	-	-	(3) (e)	-	(4) (b)	(3) (e)	(4) (b)

See the section titled “How to use the presentation rule tables,” to learn how to understand the table. Numbers in parentheses in the table above are explained below:

1. Upper limit rule

   a. If any page footing group has already been displayed on the current page of the report, the first line position where a report footing group can be displayed will be the position indicated by the value adding one to the final value of the line-counter set for the page footing group.

   b. In other cases, the first line position where a report footing group can be displayed will be the position indicated by a value adding one to integer-5 in the PAGE clause.

   c. The first line position where a report footing group can be displayed will be specified with the HEADING phrase in a PAGE clause.

2. Lower limit rule

   The last line position where a report footing group can be displayed will be specified by integer-1 in a PAGE clause.

3. Printing start line position rule

   a. The first printable line in a report footing group will be displayed at the line position specified by the integer in the LINE NUMBER clause for the group.

   b. If any page footing group has already been displayed on the current page of the report, the sum of final value of the line-counter determined by the page footing group, and the integer in the first LINE NUMBER clause for the report footing group specifies the line position where the first printable line of the report footing group is to be displayed. If a page footing group has not been displayed, the sum of the integer in the first LINE NUMBER clause for the report footing group and integer-5 in the FOOTING phrase in a PAGE clause will indicate the line position where the first printable line of a report footing group is to be displayed.

   c. If a NEXT PAGE phrase is written in a LINE NUMBER clause, the single report footing group configures one page. The first printable line of a report footing group will be displayed at the line position specified by the integer in the LINE NUMBER clause for the group.

   d. The sum of the value of the line-counter and the integer in the LINE NUMBER clause indicates the line position where the first printable line is to be displayed.

   e. No report footing group will be displayed.

4. line-counter final value setting rule

   a. The report writer control system sets the line position where the last printable line of a report footing group was displayed to the line-counter as the final value.

   b. The line-counter will not be affected by a process of a report group not containing a printable item.

Chapter 6. Procedure Division

The procedure division contains the object program. The division comes after the data division. The procedure division can be omitted if no object program is required.

Procedure Division Structure

The procedure division consists of a declarative and a procedure.

A declarative begins with DECLARATIVES, and ends with END DECLARATIVES.

A procedure consists of one paragraph, a set of several consecutive paragraphs, one section, or a set of several consecutive sections. A section contains all paragraphs belonging to the section.

[Format 1] If the declarative is omitted (1):

```
PROCEDURE DIVISION.
{section-name SECTION.              }
[paragraph-name.                    } Procedure
     [sentence ] … ] … } …          }
```

[Format 2] If the declarative is omitted (2):

```
PROCEDURE DIVISION.
[paragraph-name.                    } Procedure
     [sentence ] … ] …              }
```

[Format 3] If the declarative is included:

```
PROCEDURE DIVISION.
DECLARATIVES.                       }
{section-name SECTION.              }
     USE statement. (*1)            } Declarative
[paragraph-name.                    }
     [sentence ] … ] … } …          }
END DECLARATIVES.                   }
{section-name SECTION.              }
[paragraph-name.                    } Procedure
     [sentence ] … ] … } …          }
```

*1: A USE statement also includes a USE BEFORE REPORTING statement.

For this compiler, the names of the sections and paragraphs in the procedure can be omitted.

Section

A section consists of a section header (“section-name SECTION.”) and any number of paragraphs. Paragraphs can be omitted. A section begins at a section header and ends at one of the following locations:

- Immediately before the next section
- When a declarative, at END DECLARATIVE
- END DECLARATIVES for the declarative

Paragraph

A paragraph consists of a paragraph name and any number of sentences. A paragraph name must be followed by a period (separator). A sentence can be omitted. A paragraph begins at the paragraph name and ends at one of the following locations:

- Immediately before the next paragraph name
- Immediately before the next section-name
- End of the procedure division
- When a declarative, END DECLARATIVES

Statement

A sentence consists of any number of statements, and ends with a period (separator). The statements in the procedure division are executed in the order in which they are written.

Statements include conditional, imperative, and compiler-directing statements.

A conditional statement determines the next operation based on the condition truth value of the condition.

An imperative statement determines the next operation during compilation.

A compiler-directing statement specifies the operation to be performed during compilation. This statement is meaningless during execution. Compiler-directing statements include the COPY, REPLACE, and USE statements. The COPY and REPLACE statements (compiler-directing statements) can be written in other than the procedure division. Chapter 7, “Source Text Manipulation,” explains the COPY and REPLACE statements.

A conditional statement can be changed to an imperative statement by putting an explicit scope terminator at the end of the statement. An explicit scope terminator is a word in form of “END-verb.” For example, an ADD statement containing ON SIZE ERROR, but not containing an END-ADD phrase is a conditional statement, and that containing END-ADD is an imperative statement.

Several consecutive imperative statements can be written. A separator can be put between imperative statements. Where the format of the imperative-statement-n is shown, several consecutive imperative statements not separator period can be written.

The following table lists conditional and imperative statements:

Statement	Difference Between Conditional and Imperative Statements
ACCEPT statement	Conditional statement (*1) if it contains ON EXCEPTION phrase or NOT ON EXCEPTION phrase. Imperative statement otherwise.
ADD statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
ALTER statement	Imperative statement
CALL statement	Conditional statement (*1) if it contains ON OVERFLOW phrase, ON EXCEPTION phrase, or NOT ON EXCEPTION phrase. Imperative statement otherwise.
CANCEL statement	Imperative statement
CLOSE statement	Imperative statement
COMPUTE statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
CONTINUE statement	Imperative statement
DELETE statement	Conditional statement (*1) if it contains INVALID KEY phrase or NOT INVALID KEY phrase. Imperative statement otherwise.
DISPLAY statement	Conditional statement (*1) if it contains ON EXCEPTION phrase or NOT ON EXCEPTION phrase. Imperative statement otherwise.
DIVIDE statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
ENTRY statement	Imperative statement otherwise.
EVALUATE statement	Conditional statement (*1)
EXIT statement	Imperative statement otherwise.
EXIT PERFORM statement	Imperative statement otherwise.
EXIT PROGRAM statement	Imperative statement otherwise.
GENERATE statement	Imperative statement otherwise.
GO TO statement	Imperative statement otherwise.
IF statement	Conditional statement (*1)
INITIALIZE statement	Imperative statement otherwise.
INITIATE statement	Imperative statement otherwise.
INSPECT statement	Imperative statement otherwise.
MERGE statement	Imperative statement otherwise.
MOVE statement	Imperative statement otherwise.
MULTIPLY statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
OPEN statement	Imperative statement otherwise.

Statement	Difference Between Conditional and Imperative Statements
PERFORM statement	Imperative statement otherwise.
READ statement	Conditional statement (*1) if it contains AT END phrase, NOT AT END phrase, INVALID KEY phrase, or NOT INVALID KEY phrase. Imperative statement otherwise.
RELEASE statement	Imperative statement otherwise.
RETURN statement	Conditional statement (*1)
REWRITE statement	Conditional statement (*1) if it contains INVALID KEY phrase or NOT INVALID KEY phrase. Imperative statement otherwise.
SEARCH statement	Conditional statement (*1)
SET statement	Imperative statement otherwise.
SORT statement	Imperative statement otherwise.
START statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
STOP statement	Imperative statement otherwise.
STRING statement	Conditional statement (*1) if it contains ON OVERFLOW phrase or NOT ON OVERFLOW phrase. Imperative statement otherwise.
SUBTRACT statement	Conditional statement (*1) if it contains ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase. Imperative statement otherwise.
SUPPRESS statement	Imperative statement otherwise.
TERMINATE statement	Imperative statement otherwise.
UNLOCK statement	Imperative statement otherwise.
UNSTRING statement	Conditional statement (*1) if it contains ON OVERFLOW phrase or NOT ON OVERFLOW phrase. Imperative statement otherwise.
WRITE statement	Conditional statement (*1) if it contains INVALID KEY phrase, NOT INVALID KEY phrase, END-OF-PAGE phrase, or NOT END-OF-PAGE phrase. Imperative statement otherwise.

*1: This statement is changed to an imperative statement by putting an explicit scope terminator at the end of the statement.

A statement that explicitly transfers control to the following non executable statements is called a procedure branching statement. Procedure branching statements include the following:

- ALTER statement
- CALL statement
- EXIT statement
- EXIT PERFORM statement
- EXIT PROGRAM statement
- GO TO statement
- MERGE statement containing OUTPUT PROCEDURE phrase
- PERFORM statement
- SORT statement containing INPUT PROCEDURE phrase or OUTPUT PROCEDURE phrase

Sentence

The three types of sentence are conditional, imperative, and compiler-directing sentences. Each sentence type is constructed with different types of statements.

A conditional sentence is a conditional statement that ends with a period (separator). One conditional statement can be combined with several previous imperative statements to form one conditional sentence.

An imperative sentence is an imperative statement that ends with a period (separator). Several imperative statements can be combined to form one imperative sentence.

A compiler-directing sentence is a compiler-directing statement that ends with a period (separator). A compiler-directing sentence must consist of only one compiler-directing statement and a period (separator).

Scope of Statements

Use an explicit scope terminator or a period (separator) to explicitly specify the scope of a statement. Explicit scope terminators include the following words:

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

An explicit scope terminator terminates the scope of the statements corresponding to the terminator.

An example of specifying the scope of statements with explicit scope terminators is shown below.

```
IF A=1 THEN             … 1
   MOVE A TO B
   IF X=1 THEN          … 2                                       Scope of
      MOVE X TO Y                                    Scope of     IF statement 1
   ELSE                                              IF statement 2
      MOVE Z TO Y
   END IF               … Explicit terminator for IF statement 2
END IF                  … Explicit terminator for IF statement 1
```

A period (separator) terminates the scope of all statements before the period. An example of specifying the scope of statements with a period (separator) is shown below.

```
IF A=1 THEN             … 1
   MOVE A TO B
   IF X=1 THEN          … 2                 Scope of
      MOVE X TO Y          Scope of         IF statement 1
   ELSE                    IF statement 2
      MOVE Z TO Y.
```

Procedure Division Header

The procedure division must begin with a procedure division header. The USING phrase receives parameters from the calling program.

[Format]

```
      PROCEDURE DIVISION
           ⎧ C       ⎫
      [WITH⎨ PASCAL  ⎬LINKAGE]
           ⎩ STDCALL ⎭
      [USING {data-name-1} … ].
```

Win The WITH phrase can be specified.

Syntax Rule

1. A procedure division header can contain a USING phrase only if the program is called by a CALL statement with a USING phrase. However, for this compiler, a procedure division header can also contain a USING phrase if the program executed first in the run unit receives execution time parameters.
2. In the USING phrase, specify the data-names corresponding to the parameters passed by the calling program. The items in USING of a CALL statement in the calling program correspond to the data-names in USING of the procedure division header. This is done in sequence from left to right, as they are specified in the USING phrase.

3. data-name-1 must observe the following rules:

    a. data-name-1 must be a data item defined in the linkage section.

    b. The level number of the data item for data-name-1 must be 01 or 77.

    c. The data description entry for data-name-1 must not contain the REDEFINES clause. However, data-name-1 can be written as the object of a REDEFINES clause.

4. Each data-name in a list must be unique.

5. For details of the maximum number of times data-name-1 can be written in a USING phrase, see Appendix B, “Quantitative System Restrictions.”

Rules for the WITH Phrase

1. The WITH phrase specifies the calling conventions for a called program. If the WITH phrase is omitted, the COBOL linkage conventions are applied.

2. The WITH phrase must not be written in the Procedure Division header of a contained program.

General Rules

1. The parameters in a BY CONTENT phrase of a CALL statement, once executed, can be referenced by data-name-1 in the called program. However, the calling program cannot receive the value assigned to data-name-1 in the called program. The following must be the same in the use and character position count:

   a. A data item in USING of the procedure division header

   b. The corresponding data item in a BY CONTENT phrase of a CALL statement

2. The parameters in a BY REFERENCE phrase of a CALL statement, once executed, can be referenced by data-name-1 in the called program. The calling program can receive the value assigned to data-name-1 in the called program.

   The following must be the same in the character position count:

   a. A data item in USING of the procedure division header

   b. The corresponding data item in BY REFERENCE of a CALL statement

3. The parameters in a BY VALUE phrase of a CALL statement cannot be received by the USING phrase of the procedure division header.

4. The contents of data-name-1 are always referenced based on the data description of data-name-1.

5. A data item defined in the linkage section of the called program can be used in the procedure division under any of the following conditions:

   a. The data item is written in the USING phrase of the procedure division header or an ENTRY statement.

   b. The data item is dependent on the data item satisfying the condition in (a).

   c. A data item satisfying the condition in (a) or (b) is assigned as object side of the REDEFINES phrase or RENAMES clause. The data item is defined in the above linkage section.

d. The data item is dependent on the data item satisfying the condition in (c).

e. The data item is a condition-name or an index-name for a data item satisfying one of the conditions in (a) to (d).

Rules for the WITH Phrase

1. The calling conventions between the calling and called programs must be the same. If they are different, the execution result is undefined.
2. In a single separately compiled program, the Procedure Division header of the outermost program must be the same as the WITH phrase in the ENTRY statement.

Common Statement Rules

This section explains common statement rules and common specifications in statements.

Arithmetic Expressions

An arithmetic expression is formed by combining arithmetic operators, and identifiers or literals. There are the following five types:

1. Arithmetic expression consisting of a numeric data item, a floating-point data item, a numeric function, an integer function, a numeric literal, or the figurative constant ZERO
2. Arithmetic expression formed by combining elements in (1) using arithmetic operators

3. Arithmetic expression formed by combining two arithmetic expressions using an arithmetic operator
4. Parenthesized arithmetic expression
5. Arithmetic expression preceded by a unary arithmetic operator

Arithmetic Operator

Arithmetic operators include binary and unary arithmetic operators. The following table lists arithmetic operators:

Classification	**Operator**	**Meaning**
Binary arithmetic operator	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	**	Exponentiation (*1)
Unary arithmetic Operator	+	Same as multiplication by +1 (numeric literal).
	-	Same as multiplication by -1 (numeric literal).

*1: If positive and negative real numbers are produced by evaluating an exponentiation expression, the positive number overrides the negative number.

Rules for Writing an Arithmetic Expression

1. The following table lists combinations of identifiers, literals, arithmetic operators, and parentheses permitted in an arithmetic expression:

Subsequent Element **Preceding Element**	**Identifier or Literal**	**(**	**Unary Arithmetic Operator +,-**	**Binary Arithmetic Operator +,- ,*,/,****	**)**
Unary arithmetic operator +,-	o	o	-	-	-
(	o	o	o	-	-
Binary arithmetic operator +, -, *, /, **	o	o	o	-	-
)	-	-	-	o	o
Identifier or literal	-	-	-	o	o

o: The subsequent element can be written.
-: The subsequent element cannot be written.

2. An arithmetic expression must begin with a unary arithmetic operator, a left parenthesis, an identifier, or a literal. It must end with a right parenthesis, an identifier, or a literal.

3. An identifier in an arithmetic expression must be a numeric data item, a floating-point data item, a numeric function, or an integer function. A literal in an arithmetic expression must be a numeric literal or the figurative constant ZERO.

4. An arithmetic operator must be preceded and followed by a space. However, a space between an arithmetic operator and a parenthesis can be omitted.

5. One left parenthesis must correspond to one right parenthesis. A left parenthesis must be written before the corresponding right parenthesis. A unary arithmetic operator may appear first in an arithmetic expression following an identifier or another operator. If so, the unary arithmetic operator must be preceded by a left parenthesis.

Rules for Evaluating an Arithmetic Expression

1. The order of evaluating an arithmetic expression can be specified by using parentheses. Without parentheses, arithmetic operations are performed in the following order of precedence:

   First: + and - (unary arithmetic operator)

   Second: **

   Third: * and /

   Forth: + and - (binary arithmetic operator)

2. Use parentheses to:

   a. Change the order of performing arithmetic operations on the same level.

   b. Change the order of performing arithmetic operations on different levels.

3. A parenthesized arithmetic expression is evaluated first. If there are parentheses within parentheses, the innermost parenthesized arithmetic expression is evaluated first.

4. Consecutive arithmetic operations on the same level are performed in order from left to right.

Boolean Expression

A Boolean expression is formed by combining Boolean operators, and identifiers or literals. It has one of the following five types:

1. Boolean expression consisting of a Boolean item, a Boolean literal, the figurative constant ZERO, or ALL Boolean literal.
2. Boolean expression formed by combining elements in (1) using Boolean operators
3. Boolean expression formed by combining two Boolean expressions using a Boolean operator
4. Parenthesized Boolean expression
5. Boolean expression preceded by the unary Boolean operator NOT

Boolean Operators

Boolean operators include binary and unary Boolean operators. The following table lists Boolean operators:

Classification	**Operator**	**Meaning**
Binary Boolean operator	AND	Boolean product
	OR	Boolean sum
	EXOR	Exclusive Boolean sum
Unary Boolean operator	NOT	Boolean negation

Rules for Writing a Boolean Expression

1. The following table lists combinations of identifiers, literals, Boolean operators, and parentheses permitted in a Boolean expression:

Subsequent Element **Preceding Element**	**Identifier or Literal**	**(**	**Unary Boolean Operator NOT**	**Binary Boolean Operator AND,OR, EXOR**	**)**
Unary Boolean operator NOT	o	o	-	-	-
(	o	o	o	-	-
Binary Boolean operator AND,OR,EXOR	o	o	o	-	-
)	-	-	-	o	o
Identifier or literal	-	-	-	o	o

o: The subsequent element can be written.
-: The subsequent element cannot be written.

2. A Boolean expression must begin with a unary Boolean operator, a left parenthesis, an identifier, or a literal. It must end with a right parenthesis, an identifier, or a literal.

3. An identifier in a Boolean expression must be a Boolean item. A literal in a Boolean expression must be a Boolean literal, the figurative constant ZERO, or ALL Boolean literal.

4. All Boolean items or literals in a Boolean expression must have the same length.

5. A Boolean operator must be preceded and followed by a space. However, a space between a Boolean operator and a parenthesis can be omitted.

6. One left parenthesis must correspond to one right parenthesis. A left parenthesis must be written before a right parenthesis.

Rules for Evaluating a Boolean Expression

1. The order of evaluating a Boolean expression can be specified by using parentheses. Without parentheses, Boolean operations are performed in the following order of precedence:

   First: NOT

   Second: AND

   Third: OR and EXOR

2. Use parentheses to:

   a. Change the order of performing Boolean operations on the same level.

   b. Change the order of performing Boolean operations on different levels.

3. A parenthesized Boolean expression is evaluated first. If there are parentheses within parentheses, the innermost parenthesized Boolean expression is evaluated first.

4. Consecutive Boolean operations on the same level are performed in order from left to right.

5. AND, OR, EXOR, and NOT in the following are Boolean operators, and not logical operators:

   a. Two Boolean expressions combined only by AND, OR, or EXOR

   b. One Boolean expression beginning with NOT

Conditional Expression

A conditional expression takes on the truth value indicating whether the condition is satisfied (true) or not (false). A conditional expression controls the flow of a program by examining the truth value. A conditional expression can be written in the EVALUATE, IF, PERFORM, and SEARCH statements.

Conditional expressions have the following types:

$$\text{Conditional expression} \begin{cases} \text{Simple condition} \begin{cases} \text{Relation condition} \\ \text{Class condition} \\ \text{Condition-name condition} \\ \text{Switch-status condition} \\ \text{Sign condition} \end{cases} \\ \\ \text{Complex condition} \end{cases}$$

Relation Condition

A relation condition is for comparing two operands. An identifier, a literal, an arithmetic expression, or an index-name can be written as an operand.

[Format]

```
                           { IS [NOT] GREATER THAN                    }
                           { IS [NOT] LESS THAN                       }
                           { IS [NOT] EQUAL TO                        }
{ identifier-1           } { IS [NOT] >                               } { identifier-2           }
{ literal-1              } { IS [NOT] <                               } { literal-2              }
{ arithmetic-expression-1} { IS [NOT] =                               } { arithmetic-expression-2}
{ index-name-1           } { IS GREATER THAN OR EQUAL TO              } { index-name-2           }
                           { IS >=                                    }
                           { IS LESS THAN OR EQUAL TO                 }
                           { IS <=                                    }
```

Note: >, <, =, >=, and <= are not underlined for avoiding confusion with other symbols.

1. An operand (identifier-1, literal-1, arithmetic-expression-1, or index-name-1) on the left of a relational operator is called “left side of a condition.” An operand (identifier-2, literal-2, arithmetic-expression-2, or index-name-2) on the right of a relational operator is called “right side of a condition.”
2. A relation condition must contain at least one variable.
3. A reserved word making up a relational operator must be preceded and followed by one or more spaces.
4. A truth value is determined by comparing the values on the right and left sides of a relation condition. This is done according to a relational operator. The type of relation condition comparison is determined by a relational operator. The following table lists the meanings of relational operators:

Relational Operator	**Meaning**
IS [NOT] GREATER THAN and IS [NOT] >	The value on the left side (subject) of an expression is greater than that on the right side (object). [The value on the left side (subject) of an expression is not greater than that on the right side (object).] (*1)
IS [NOT] LESS THAN and IS [NOT] <	The value on the left side of an expression is less than that on the right side. [The value on the left side (subject) of an expression is not less than that on the right side (object).] (*1)
IS [NOT] EQUAL TO and IS [NOT]=	The value on the left side of an expression equals that on the right side. [The value on the left side (subject) of an expression is not equal to that on the right side (object).] (*1)
IS GREATER THAN OR EQUAL TO and IS >=	The value on the left side of an expression is greater than or equal to that on the right side object. (*2)
IS LESS THAN OR EQUAL TO and IS <=	The value on the left side subject of an expression is less than or equal to that on the right side object. (*3)

*1: Brackets [ ] indicate that NOT is written within the brackets.

*2: IS GREATER THAN OR EQUAL TO is equivalent to IS NOT LESS THAN. IS >= is equivalent to IS NOT <.

*3: IS LESS THAN OR EQUAL TO is equivalent to IS NOT GREATER THAN. IS <= is equivalent to IS NOT >.

5. For details of the rules for relation condition comparison, see the section titled “Comparison rules.”

Class Condition

A class condition is for examining the class of a data item.

[Format]

$$\text{identifier-1 IS [}\underline{\text{NOT}}\text{]}\left\{\begin{array}{l}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\\ \underline{\text{ALPHABETIC-LOWER}}\\ \underline{\text{ALPHABETIC-UPPER}}\\ \underline{\text{BOOLEAN}}\\ \text{class-name}\end{array}\right\}$$

1. In a class condition, the content of identifier-1 is examined according to a class examination specification, then a truth value is determined. The following table lists the conditions under which class examinations show that an expression is true:

Type of Class Examination	**Condition Where a Class Examination Shows that an Expression Is True**
NUMERIC examination	identifier-1 contains only digits 0 to 9, or digits 0 to 9 with operational signs. (*1)
ALPHABETIC examination	identifier-1 contains upper case alphabetic characters A to Z, and blanks, lower case alphabetic characters a to z, and blanks, or upper case and lower case alphabetic characters, and blanks.
ALPHABETIC-LOWER examination	identifier-1 contains lower case alphabetic characters a to z and blanks.
ALPHABETIC-UPPER examination	identifier-1 contains upper case alphabetic characters A to Z, and blanks.
BOOLEAN examination	identifier-1 contains only Boolean characters 0 and 1.
class-name examination	identifier-1 contains a set of the characters specified in the CLASS phrase of a special-names paragraph

*1: For details, see item (6).

2. identifier-1 must be one of the following:
   a. Data item to be displayed
   b. Alphanumeric function identifier
   c. Packed decimal item
3. For the NUMERIC examination, the following data items cannot be specified in identifier-1:
   a. Alphabetic data item
   b. Group item containing signed numeric data items
   c. National data item
   d. National edited data item
   e. Boolean item

4. For the ALPHABETIC, ALPHABETIC-LOWER, ALPHABETIC-UPPER, and class-name examinations, the following data items cannot be specified in identifier-1:

    a. Numeric data item

    b. National data item

    c. National edited data item

    d. Boolean item

5. For the BOOLEAN examination, the following data items cannot be specified in identifier-1:

    a. Numeric data item

    b. Alphabetic data item

    c. National data item

    d. National edited data item

6. The NUMERIC examination shows that an expression is true under any of the following conditions:

    a. identifier-1 (data item) contains only digits without operational signs. Alternatively, identifier-1 (packed decimal item) contains hexadecimal F as a sign.

    b. identifier-1 (data item) contains only digits with a valid operational sign. An operational sign is valid under one of the following conditions:

        A SEPARATE CHARACTER phrase is specified in the SIGN clause of identifier-1 (data item), and the sign is "+" or "-" expressed in the standard data format.

        A SEPARATE CHARACTER phrase is not specified in the SIGN clause of identifier-1 (zoned decimal item), and the sign is hexadecimal 4, 5, or 3.

identifier-1 (packed decimal item) contains hexadecimal C, D, or F as a sign.

7. NOT and a key word are combined to form one class condition. An expression with NOT has the truth value opposite to that of the expression without NOT. For example, NOT NUMERIC is true if the operand is not a number.

Condition-name Condition

A condition-name condition is for checking whether the value of a conditional variable equals that of a condition-name.

[Format]

condition-name-1

1. A condition-name condition is true under either of the following conditions:

    a. THRU is not written in the VALUE clause of condition-name-1 (data description entry). Also, the value of a conditional variable equals that specified in the VALUE clause.

    b. THRU is written in the VALUE clause of condition-name-1 (data description entry). Also, the value of a conditional variable is within the range (including the lower and upper limits specified in THRU) specified in the VALUE clause.

2. Compare the value of a conditional variable with that of condition-name-1 according to the rules for relation condition comparison. For details of the comparison rules, see the section titled “Comparison rules.”

3. Examples of condition-name conditions are shown below.

   a. If conditional variables and condition-names are defined as follows:

```
02 MONTH PICTURE 99.                 ...Definition of the conditional variable MONTH
   88 SPRING-M    VALUE 3 THRU 5.    ⎫
   88 SUMMER-M    VALUE 6 THRU 8.    ⎪ Definitions of the condition-names SPRING-M,
   88 FALL-M      VALUE 9 THRU 11.   ⎬ SUMMER-M, FALL-M, and WINTER-M
   88 WINTER-M    VALUE 12, 1, 2.    ⎭
```

   b. In order to check the value of the conditional variable MONTH, you can use condition-name conditions to write the following IF statements:

1. IF SPRING-M: Equivalent to IF MONTH >= 3 AND <= 5.
2. IF SUMMER-M: Equivalent to IF MONTH >= 6 AND <= 8.
3. IF FALL-M: Equivalent to IF MONTH >= 9 AND <= 11.
4. IF WINTER-M: Equivalent to IF MONTH = 12 OR 1 OR 2.

Switch-status Condition

A switch-status condition is for checking whether an external switch is on.

[Format]

condition-name-1

1. Define an external switch in the special-names paragraph of the environment division. Associate the condition-name to the on or off status of the switch.
2. If the external switch is set to condition-name-1, the switch-status condition is true.

Sign Condition

A sign condition is for checking whether the algebraic value of an arithmetic expression is greater than, less than, or equal to zero.

[Format]

$$\text{arithmetic-expression-1 IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{POSITIVE}} \\ \underline{\text{NEGATIVE}} \\ \underline{\text{ZERO}} \end{array} \right\}$$

1. arithmetic-expression-1 must contain at least one identifier.
2. In a sign condition, the algebraic value of an arithmetic expression is examined, and the truth value is determined. This is done based on whether POSITIVE, NEGATIVE, or ZERO is specified. The following table lists the meanings of these key words:

Keyword	**Meaning**
POSITIVE	Greater than zero
NEGATIVE	Less than zero
ZERO	Equal to zero

3. NOT and a key word are combined to form one sign condition. An expression with NOT has the truth value opposite to that of the expression without NOT. For example, NOT ZERO is true if the value of an arithmetic expression is not zero (i.e., positive or negative).

Complex Condition

A complex condition is formed by combining one or more simple conditions and logical operator.

A complex condition consists of negated and combined conditions. A negated condition is a simple or complex condition preceded by the logical operator NOT. A combined condition is simple or complex conditions combined by the logical operator AND or OR.

The truth value of a complex condition is obtained by evaluating individual conditions and performing all logical operations sequentially.

The following table lists the meanings of logical operators:

Logical Operator	**Meaning**	**Truth Value**
NOT	Logical negation	If the condition following NOT is false, the truth value is true. If the condition following NOT is true, the truth value is false.
AND	Logical product	If both conditions on the right and left of AND are true, the truth value is true. If either or both conditions on the right and left of AND are false, the truth value is false.
OR	Logical sum	If either or both conditions on the right and left of OR are true, the truth value is true. If both conditions on the right and left of OR are false, the truth value is false.

Negated Condition

A negated condition is for reversing the truth value of a condition.

[Format]

$\underline{\text{NOT}}$ condition-1

Combined Condition

A combined condition is for obtaining a logical product or sum.

[Format]

$$\text{condition-1}\left\{\left\{\begin{array}{l}\underline{\text{AND}}\\ \\ \underline{\text{OR}}\end{array}\right\}\text{ condition-2}\right\}\ldots$$

Rules for Writing a Complex Condition Expression

1. The following table lists combinations of simple conditions, logical operator, and parentheses permitted in a complex condition expression:

Subsequent Element **Preceding Element**	**Simple Condition**	**AND**	**OR**	**NOT**	**(**	**)**
Simple condition	-	o	o	-	-	o
AND	o	-	-	o	o	-
OR	o	-	-	o	o	-
NOT	o	-	-	-	o	-
(	o	-	-	o	o	-
)	-	o	o	-	-	o

o: The subsequent element can be written.

-: The subsequent element cannot be written.

2. A complex condition must begin with a simple condition, NOT, or left parenthesis and end with a simple condition or right parenthesis.
3. A logical operator must be preceded and followed by a space. However, a space between a logical operator and a parenthesis can be omitted.
4. One left parenthesis must correspond to one right parenthesis. A left parenthesis must be written before the corresponding right parenthesis.

Rules for Evaluating a Complex Condition

1. The order of evaluating a complex condition can be specified by using parentheses. Without parentheses, logical operations are performed in the following order of precedence:

   First: NOT

   Second: AND

   Third: OR

2. To change the order of performing logical operations, enclose the range of the conditions combined by a logical operator in parentheses. The parenthesized conditions combined by a logical operator are evaluated first, or a parenthesized logical operation is performed first. If there are parentheses within parentheses, the innermost parenthesized condition is evaluated first, or the innermost parenthesized logical operation is performed first.

3. Consecutive logical operations on the same level are performed in order from left to right.

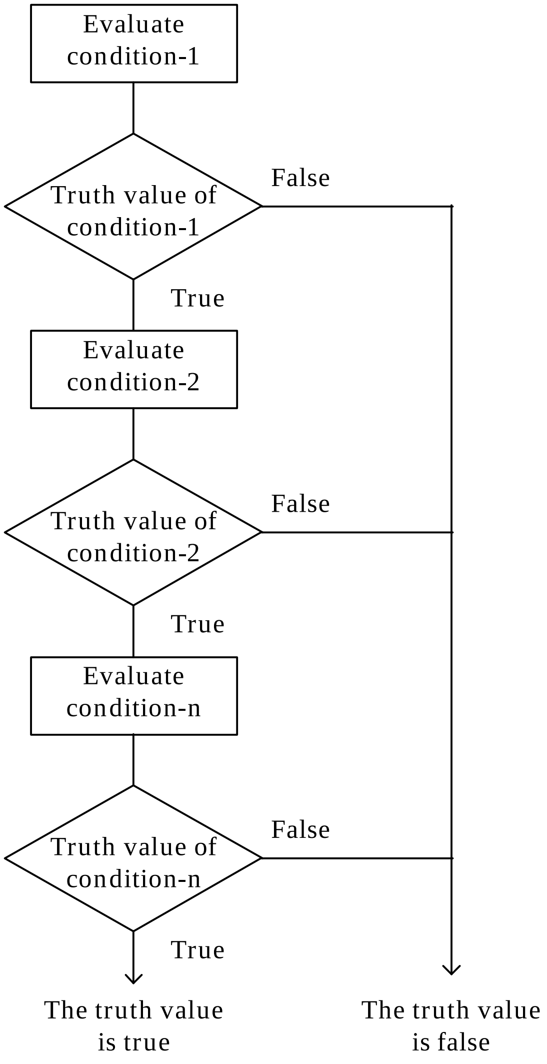
4. Evaluation of the individual conditions of a complex condition terminates when the truth value of the complex condition is determined. This is done regardless of whether all the individual conditions are evaluated.

5. The values of the arithmetic expressions and functions in a complex condition expression are determined when the expression is evaluated.

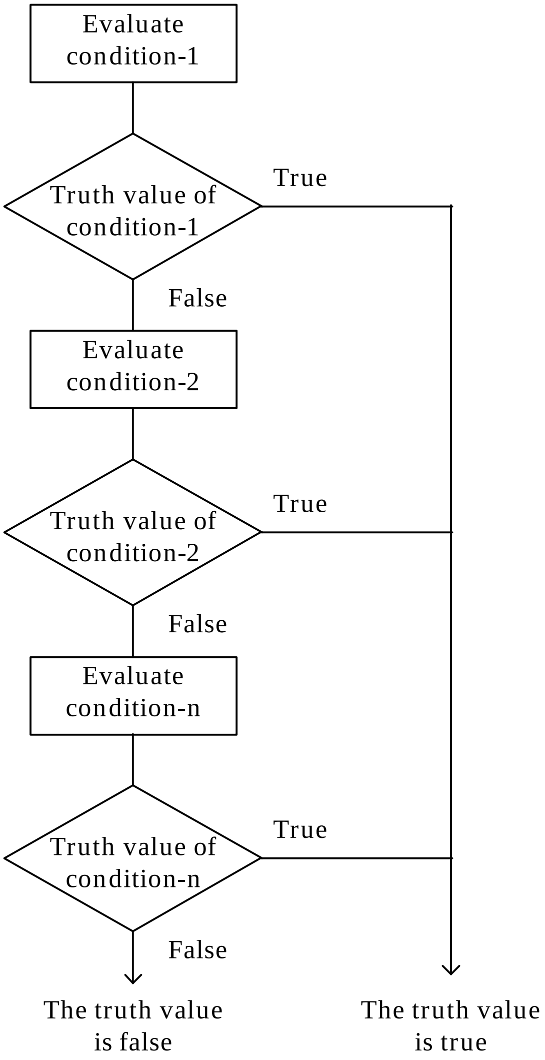
6. A negated condition is evaluated when the condition combined with the logical operator NOT is evaluated.

Example of the Order of Evaluating a Complex Condition

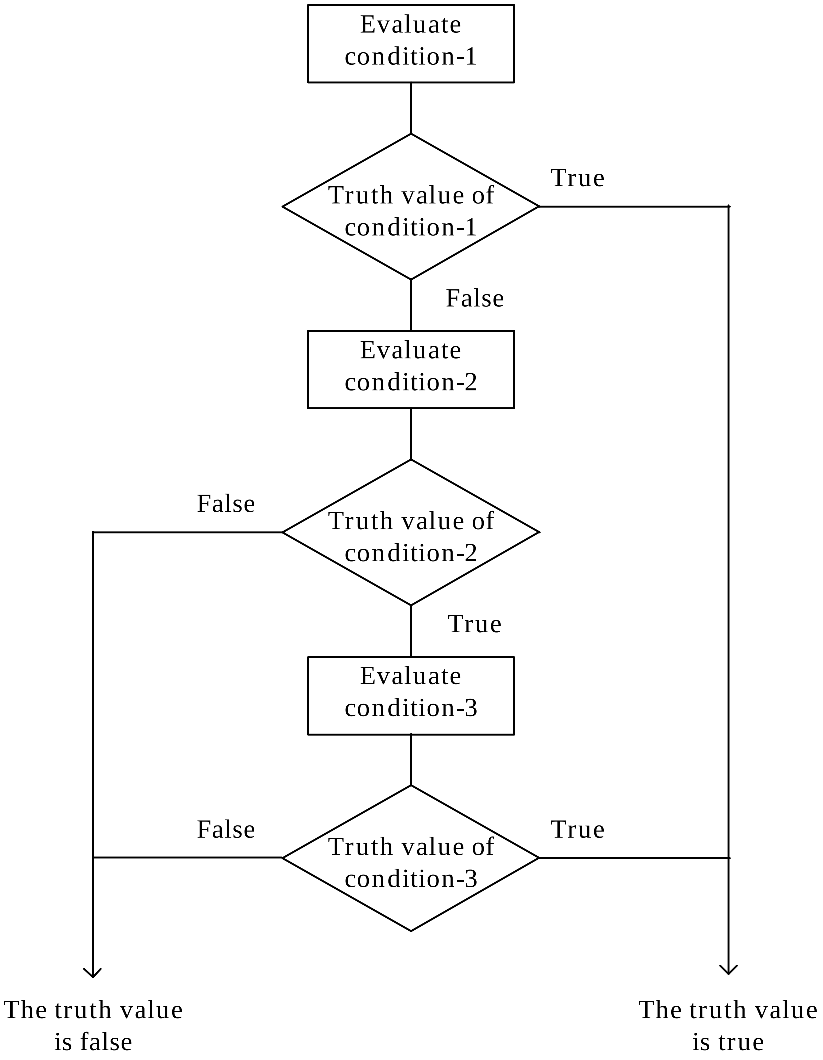
The following figure shows the order of evaluating “condition-1 AND condition-2 AND ... condition-n”:



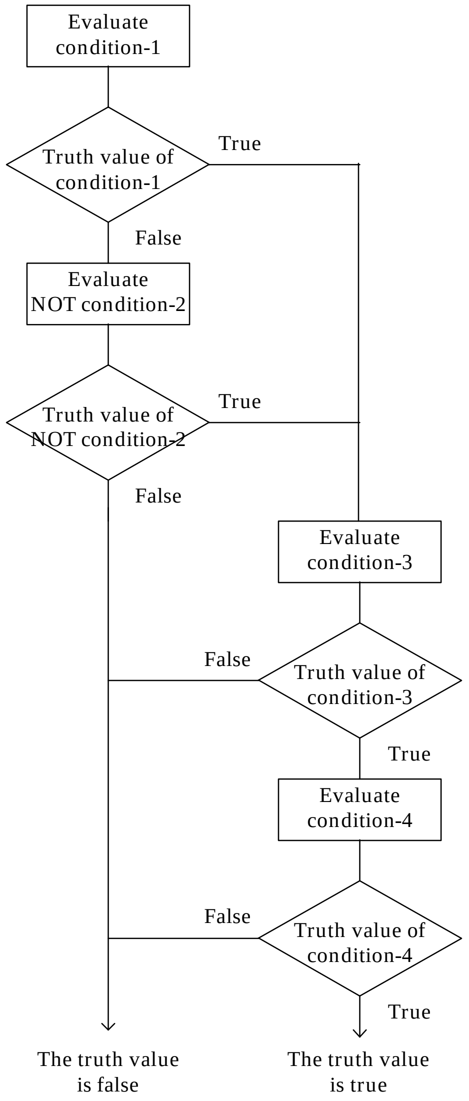
The following figure shows the order of evaluating “condition-1 OR condition-2 OR ... condition-n”:



The following figure shows the order of evaluating “condition-1 OR condition-2 AND condition-3”:



The following figure shows the order of evaluating "(condition-1 OR NOT condition-2) AND condition-3 AND condition-4":



Abbreviating a Combined Relation Condition

A complex condition formed by combining a relation condition and a logical operator is called a combined relation condition. A combined relation condition expression may not contain a parenthesis that changes the order of performing logical operations. If so, part of the combined condition can be omitted.

Part of a combined condition can be omitted as follows:

a. The subject of a relation condition can be omitted if it is the same as the subject of the preceding relation condition.

b. The subject and relational operator of a relation condition can be omitted if they are the same as the subject and relational operator.

[Format]

$$\text{relation condition}\left\{\begin{Bmatrix}\underline{\text{AND}}\\ \\ \underline{\text{OR}}\end{Bmatrix}[\underline{\text{NOT}}]\ [\text{relational operator}]\ \text{object}\right\}\ldots$$

1. These two methods of omission can be used for a series of relation conditions.

2. The subject of the preceding relational condition is used in place of the omitted subject. The preceding relational operator is used in place of the omitted relational operator. The omitted operands and relational operators are repeatedly compensating for until an unabbreviated simple condition appears in the complex condition expression. The results of compensating for the subject and relational operator must observe the rules provided in the section titled, “Complex condition.”

3. NOT in the expression for an abbreviated combined relation condition is interpreted as follows:

   a. NOT followed by GREATER, >, LESS, <, EQUAL, or = is treated as part of the relational operator.

   b. NOT other than that in (a) is treated as a logical operator. Therefore, the result of compensating for any omitted subject or omitted relational operator is a negated condition.

4. The following table lists examples of abbreviated combined relation conditions:

Abbreviated Combined Relation Condition	**Unabbreviated Combined Relation Condition**
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT (((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d)))

_: Operand on the left side and relational operator that can be omitted

Comparison Rules

This section explains the rules for relation condition comparison. One of the following rules is applied based on a combination of the subject and object of a relation condition:

a. Nonnumeric comparison

b. Numeric comparison

c. Comparison of national characters

d. Boolean comparison

e. Comparison of pointer data

f. Index comparison

The rules for Boolean comparison apply to the subject or object Boolean item of a relation condition. The rules for pointer data comparison apply to the subject or object pointer data item or ADDR function of the condition. The rules for index comparison apply to the subject or object index-name or index data item of the condition.

The following table lists combinations of the operands to which the rules for character, numeric, and national character comparison are applied:

Subject or object of condition Object or subject of condition	Group	Alphabetic character Alphanumeric(*1) Alphanumeric edited Numeric edited	National National Edited	Zoned Decimal	Binary Packed Decimal	Floating Point	Arithmetic expression(*2)
Group Item	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	Nonnumeric comparison	-
Alphabetic data item Alphanumeric data item (*1) Alphanumeric edited data item Numeric edited data item Nonnumeric literal(*3)	Nonnumeric comparison	Nonnumeric comparison	-	Nonnumeric comparison	-	-	-
National data item(*4) National edited data item National literal(*5)	Nonnumeric comparison	-	Comparison of national characters	-	-	-	-
Figurative constant SPACE HIGH-VALUE LOW-VALUE	Nonnumeric comparison	Nonnumeric comparison	Comparison of national characters	Nonnumeric comparison	-	-	-
Figurative constant QUOTE Symbolic-character	Nonnumeric comparison	Nonnumeric comparison	-	Nonnumeric comparison	-	-	-
Zoned decimal item	Nonnumeric comparison	Nonnumeric comparison	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
Binary item packed decimal item	Nonnumeric comparison	-	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
Numeric literal	Nonnumeric comparison	Nonnumeric comparison (*6)	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
Figurative constant ZERO	Nonnumeric comparison	Nonnumeric comparison	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
Floating-point data item Floating-point literal	Nonnumeric comparison	-	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison
Arithmetic expression(*2)	-	-	-	Numeric comparison	Numeric comparison	Numeric comparison	Numeric comparison

-: Incomparable combination

*1: Alphanumeric data items include alphanumeric functions.

*2: Arithmetic expressions include numeric and integer functions.

*3: Character literal include ALL character literals.

*4: National data items include national functions .

*5: National nonnumeric literal include ALL national nonnumeric literals.

*6: An alphanumeric function cannot be compared with a numeric literal.

Nonnumeric Comparison

The rules for nonnumeric comparison are as follows:

1. Two operands are compared based on the collating sequence.
2. The number of the character positions for two operands may be the same. If so, each pair of the characters in the corresponding positions is sequentially compared from left to right. The result of comparison is determined as follows:
   a. If all the corresponding characters are the same, the two operands are equal.
   b. Each pair of the characters in the corresponding positions is compared until a pair of different characters appears. At this point, comparison determines which operand including a high position character is the largest in the collating sequence.
3. The number of the character positions for two operands may differ. If so, the shorter operand is treated as if it were padded on to the right with spaces to the length of the longer operand. The method of comparison is the same as for (2).
4. Suppose the following are compared:
   a. Numeric operand (zoned decimal item, binary item, packed decimal item, numeric literal, figurative constant ZERO, floating-point data item, or floating-point literal).
   b. Nonnumeric operand (group item, alphabetic data item, alphanumeric data item, alphanumeric edited data item, numeric edited data item, nonnumeric literal, national data item, national edited item, national literal , figurative constant SPACE, HIGH-VALUE, LOW-VALUE, or QUOTE, or symbolic-character).

c. The following rules are applied:

- The numeric operand must be an integer.
- The nonnumeric operand may be an elementary item or a nonnumeric literal. If so, the numeric operand is treated as if it were copied to an alphanumeric data item having the same length as the nonnumeric operand. This alphanumeric data item is compared with the nonnumeric operand.
- The nonnumeric operand may be a group item. If so, first the numeric operand is treated as if it were copied into a group item having the same length as the nonnumeric operand. Then, this group item is compared with the nonnumeric operand.

5. For details of the collating sequence, see the section titled “PROGRAM COLLATING SEQUENCE clause.”

Numeric Comparison

The rules for numeric comparison are as follows:

1. Two operands are compared based on an algebraic value.
2. The value of zero is compared as zero regardless of whether a sign is included.
3. An unsigned operand is treated as if the sign were positive for comparison.

Comparison of National Nonnumeric Characters

The rules for comparing national nonnumeric characters are as follows:

1. Two operands are compared based on the collating sequence of national nonnumeric characters.

2. The number of the national nonnumeric characters for two operands may be the same. If so, each pair of the national nonnumeric characters in the corresponding positions is sequentially compared from left to right. The result of comparison is determined as follows:

   a. If all the corresponding national nonnumeric characters are the same, the two operands are equal.

   b. Each pair of the national nonnumeric characters in the corresponding positions is compared until a pair of different national nonnumeric characters appears. At this point, comparison determines which operand including a high-position national character is largest in the national character collating sequence.

3. The number of the national character positions for two operands may differ. If so, the shorter operand is treated as if it were padded on the right with national spaces to the length of the longer operand. The method of comparison is the same as for (2).

Boolean Comparison

The rules for Boolean comparison are as follows:

1. For Boolean comparison, a Boolean item can be compared with the following operands:

   a. Boolean literal (including ALL Boolean literals)

   b. Figurative constant ZERO

2. Boolean items to be compared need not be for the same purpose.

3. A relational operator must be either of the following:

   a. IS [NOT] EQUAL TO

   b. IS [NOT] =

4. The length (number of Boolean characters) of two operands must be the same.

5. Each pair of the Boolean characters in the corresponding positions of two operands is sequentially compared from left to right. If all the corresponding Boolean characters are the same, the two operands are equal.

6. A Boolean expression cannot be written as an operand.

Comparison of Pointer Data

The rules for comparing pointer data are as follows:

1. For pointer data comparison, a pointer data item or an ADDR function can be compared with the following operands:
   a. Pointer data item
   b. ADDR function
   c. Figurative constant ZERO
2. Pointer data comparison can be written only in the relation condition expression of the IF or EVALUATE statement.
3. A relational operator must be either of the following:
   a. IS [NOT] EQUAL TO
   b. IS [NOT] =

Index Comparison

The following table lists the rules for index comparison:

Subject or Object of a Condition / Object or Subject of a Condition	**Index-name**	**Index Data Item**	**Numeric Literal (Integer Only)**	**Numeric Data Item (Integer Only)**
Index-name	Comparison of occurrence numbers	Comparison without conversion	Comparison between an occurrence number and an integer	Comparison between an occurrence number and an integer
Index data item	Comparison without conversion	Comparison without conversion	-	-

-: Incomparable combination

Comparison of the occurrence numbers: The occurrence numbers corresponding to the index-names are compared.

Comparison between the occurrence number and the integer: The occurrence number corresponding to the index-name is compared with another operand.

Comparison without conversion: Actual values are compared as they are.

Rules for Copying

This section explains the rules for copying using the MOVE statement.

Execution of a statement may copy a data item, a literal, or the result of an arithmetic operation into a data item. The rules for copying apply not only to such implicit MOVE statements, but also to explicit MOVE statements.

The types of copying rules include elementary and group item copying. If either or both of sending and receiving sides are group items, the rules for copying group items are applied. Otherwise, the rules for copying elementary items are applied.

Copying of Elementary Items

For copying of an elementary item, one of the following rules is applied depending on the category and use of the receiving side:

a. Alphabetic copying

b. Alphanumeric and alphanumeric edited copying

c. Numeric and numeric edited copying

d. Floating-point copying

e. National and national edited copying

f. Boolean copying

g. Pointer data copying

For copying of an elementary item, the internal representation format is converted, edited, or de-edition, if necessary.

The following table lists combinations of operands for copying elementary items:

Receiving Side Sending Side		Alphabetic character	Alphanumeric and Alpha-Numeric Edited	Numeric and Numeric Edited	Floating Point	National and National Edited	Boolean	Pointer Data
Alphabetic item		1	2	-	-	-	-	-
Alphanumeric item (*1) Nonnumeric literal (*2)		1	2	3	4	-	6 (*8)	-
Alphanumeric edited data item		1	2	-	-	-	-	-
Numeric item and literal	Integer	-	2	3	4	-	-	-
	Non-integer	-	-	3	4	-	-	-
Numeric edited data item		-	2	3	4	-	-	-
Floating-point item Floating-point literal		-	-	3	4	-	-	-
National item (*3) National edited item National literal (*4)		-	-	-	-	5	-	-
Boolean item Boolean literal (*5)		-	2 (*7)	-	-	-	6	-
Pointer data item (*6)		-	-	-	-	-	-	7
Figurative constant ZERO		-	2	3	4	-	6	7
Figurative constant SPACE		1	2	-	-	5	-	-
Figurative constants HIGH-VALUE LOW-VALUE		-	2	-	-	5	-	-
Figurative constant QUOTE Symbolic-character		-	2	-	-	-	-	-

1) to 7): Combination of items that can be copied

1) to 7) correspond to the following header numbers:
-: Combination of items that cannot be copied

*1 Alphanumeric data items include alphanumeric functions.
*2 Nonnumeric literals include ALL nonnumeric literals.
*3 National data item also refers to national function.
*4 National nonnumeric literals include ALL national nonnumeric literals.
*5 Boolean literal include ALL Boolean literals.
*6 Pointer data items include ADDR functions.
*7 A Boolean item cannot be copied into an alphanumeric edited data item.
*8 A nonnumeric literal cannot be copied into a Boolean item.

Alphabetic Copying: 1

If the receiving side is an alphabetic item, data is aligned, and the remaining portion is padded with blanks as required. This is done according to the standard alignment rule.

Alphanumeric and Alphanumeric Edited Copying: 2

If the receiving side is an alphanumeric data item or an alphanumeric edited data item, the following rules are applied:

1. Data is aligned, and the remaining portion is padded with blanks as required. This is done according to the standard alignment rule.

2. If the sending side is a signed numeric data item, the sign is not copied. If the SIGN clause in the sending item contains a SEPARATE phrase, the sign is not copied. Therefore, in the edited data format, the number of the digits of the sending side is treated as if it were one digit less.

3. If the sending side is a numeric edited data item, data is not de-editing (removal of edited characters).

4. If use of the sending side differs from that of the receiving side, the representation of the sending side is converted to the internal representation of the receiving side.

5. If the PICTURE clause sending numeric data item contains P, the digit position indicated by P is treated as a zero. P is included in a calculation of the length of the sending side.

Numeric and Numeric Edited Copying: 3

If the receiving side is a numeric data item or a numeric edited data item, the following rules are applied:

1. Data is aligned by the decimal point, and the remaining portion is padded with zeros as required. This is done according to the standard alignment rule. The zeros may be converted to other characters according to the description of the PICTURE phrase.
2. If the sending side is a numeric edited data item, first data is de-editing, then an unsigned unedited number is obtained. The unedited number is copied into the receiving side.
3. If the receiving side is a signed numeric data item, the receiving side has the same sign as the sending side. At this point, the representation format of the sign is converted, if necessary. If the sending side does not have a sign, the positive sign is assigned to the receiving side.
4. If the receiving side is an unsigned numeric data item, the absolute value of the sending side is copied. A sign is not assigned to the receiving side.
5. If the category of the sending side is alphanumeric, the send data is treated as an unsigned integer, and copied.

Floating-point Copying: 4

If the receiving side is a floating-point data item, data is treated as if it were copied into a non-integer numeric data item.

National and National Edited Copying: 5

If the receiving side is a national or national edited item, data is aligned, and the remaining portion is padded with blanks as required. This is done according to the standard alignment rule.

Boolean Copying: 6

If the receiving side is a Boolean item, the following rules are applied:

1. Data is aligned, and the remaining portion is padded with Boolean character zeros as required. This is done according to the standard alignment rule.
2. If the sending side is an alphanumeric data item, the sending side is treated as an external Boolean item.

Pointer Data Copying: 7

If the receiving side is a pointer data item, the following rules are applied:

1. If the sending side is a pointer data item or an ADDR function, the contents of the sending side are copied as they are.
2. If the sending side is the figurative constant ZERO, a 0 is copied.

Group Item Copying

If either or both of the sending and receiving sides are group items, the following rules are applied:

1. A Boolean item, a pointer data item, or a floating-point literal cannot be copied into a group item.

2. A group item cannot be copied into a Boolean item or a pointer data item.

3. A group item is copied in the same way in which an elementary item is copied to and from an alphanumeric data item.

4. The internal representation format is not converted.

5. The individual elementary items belonging to a group item and group items are not considered. The entire group item is copied just like one alphanumeric data item. If the OCCURS clause is specified in a group item or in a data item belonging to a group item, another rule is applied. For details of the rules for copying a group item containing the OCCURS clause, see the section titled “OCCURS clause.”

Arithmetic Statement

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements are called arithmetic statements. The following rules are common to arithmetic statements:

1. The data description entries of the operands in an arithmetic statement need not be the same. During computation, data is converted, and aligned by a decimal point as required.
2. An arithmetic operation may require the data item to contain a temporary operation result. This temporary data item is called an intermediate result. The storage field for an intermediate result is provided by a compiler as a signed numeric data item. The number of the digits of an intermediate result is based on the algorithm explained in Appendix D, “Intermediate Results.” The operation result temporarily stored in the field during execution is copied into a data item for storage according to the MOVE statement rules.

More Than One Answer of an Arithmetic Statement

An arithmetic statement can contain one or more resultant identifiers (data items to contain results). In this case, the results of an arithmetic statement are computed as follows:

1. In a statement, all data items to be initially evaluated are computed as required. The result is stored in a temporary data item.
2. Then, the temporary data item obtained in (1) is computed for each resultant identifier, and the results are stored. This computation is done in the order in which the identifiers are specified (from left to right).

An example of obtaining more than one answer is shown below. temp indicates a temporary storage field provided by a compiler.

Example 1:

“ADD a b c TO c d(c) e” is computed in the same way in which the following statements are executed sequentially:

```
ADD a b c GIVING temp
ADD temp TO c
ADD temp TO d(c)
```

... The value of c was changed by the preceding addition.

```
ADD temp TO e
```

Example 2:

“MULTIPLY a(i) BY i a(i)” is computed in the same way in which the following statements are executed sequentially:

```
MOVE a(i) TO temp
MULTIPLY temp BY I
MULTIPLY temp BY a(i)
```

... The value of i was changed by the preceding multiplication.

ROUNDED Phrase

A ROUNDED phrase can be written in arithmetic statements.

The number of the decimal places obtained by an arithmetic operation may be greater than that of a resultant identifier. The following processing is performed depending on whether the ROUNDED phrase is present:

1. Without the ROUNDED phrase, the fraction part obtained by the arithmetic operation is rounded down to the length of the resultant identifier.
2. With the ROUNDED phrase, the value of the highest digit of the truncated part may be 5 or more. If so, the absolute value of the lowest digit of the resultant identifier is incremented by one.

The lowest digit of the integer part of a resultant identifier may be defined by the PICTURE symbol P. If so, the low order end of the integer part to which a storage field is actually allocated is truncated or rounded down.

ON SIZE ERROR Phrase

Execution of an arithmetic statement may cause a size error condition, which can be detected by written an ON SIZE ERROR phrase in the arithmetic statement.

Conditions Where a Size Error Condition Occurs

A size error condition occurs under any of the following conditions:

1. The base of exponentiation is zero, and the exponent is zero or less.
2. The result of evaluating exponentiation is not a real number.
3. The divisor is zero.
4. The absolute value obtained by an operation exceeded the maximum value that can be stored in a resultant identifier.

The maximum value that can be stored in a resultant identifier in (4) is that specified in the PICTURE character-string. If a resultant identifier is a binary item, this maximum value is also that specified in the PICTURE character-string. It is not the maximum value that can be stored in a storage field.

A size error condition in (4) occurs only when the final result is stored by one arithmetic operation. It does not occur when an intermediate result is stored. With ROUNDED phrase, a size error condition in (4) is checked after rounding off.

If two or more resultant identifiers are written, a size error condition is checked each time the result of the corresponding arithmetic operation is obtained.

Operation Performed if a Size Error Condition Occurs

The value of a resultant identifier in which a size error condition occurred is as follows:

1. With the ON SIZE ERROR phrase or NOT ON SIZE ERROR phrase, the value of an identifier in which a size error condition occurred remains unchanged. (The value before execution of an arithmetic statement)
2. Without the ON SIZE ERROR phrase and NOT ON SIZE ERROR phrase, the value of an identifier in which a size error condition occurred is undefined.
3. In an identifier in which no size error condition occurred, the result of the arithmetic operation is stored. This is done regardless of whether an ON SIZE ERROR phrase and a NOT ON SIZE ERROR phrase are written.

After completion of the arithmetic operation, that is, after all the values of resultant identifiers have been determined, control is transferred according to the following rules:

1. Control is transferred to an imperative statement in which the ON SIZE ERROR phrase is written. After the statement is executed, control is transferred to the end of the arithmetic statement. However, in the imperative statement, a procedure branching statement or a conditional statement that causes explicit control transfer may be executed. If so, control is transferred according to the rules of the statement.
2. Without the ON SIZE ERROR phrase, control is transferred to the end of the arithmetic statement.

Operation Performed if No Size Error Condition Occurs

If no size error condition occurs, after completion of an arithmetic operation, control is transferred according to the following rules:

1. Control is transferred to an imperative statement in which NOT ON SIZE ERROR phrase is written. After the statement is executed, control is transferred to the end of the arithmetic statement. However, in the imperative statement, a procedure branching statement or a conditional statement that causes explicit control transfer may be executed. If so, control is transferred according to the rules of the statement.
2. Without NOT ON SIZE ERROR, control is transferred to the end of the arithmetic statement.

CORRESPONDING Phrase

In the MOVE, ADD, and SUBTRACT statements, a CORRESPONDING phrase can be written. The CORRESPONDING phrase associates data items having the same name and belonging to a group item with each other.

When a CORRESPONDING phrase is written in the following format:

CORRESPONDING d1 TO d2

Then, d1, d2 and the data items to be associated with must satisfy the following conditions:

1. d1 and d2 must be group items. A data item having a level number of 66, 77, or 88 must not be specified as d1 and d2.
2. A data item containing the USAGE IS INDEX clause must not be specified as d1 and d2.

3. d1 and d2 must not be referenced partially.
4. The names of the data items to be associated with must be unique by appending implicit qualifiers to them.
5. For the MOVE statement, at least either of the data items to be associated with must be an elementary item. A combination of the data items to be associated with must obey the copying rules.
6. For the ADD and SUBTRACT statements, the data items to be associated with must be numeric data items.

If “CORRESPONDING d1 TO d2” is written, data items belonging to d1 are associated with those belonging to d2. Also, the data items must satisfy all the following conditions:

1. The data-names are the same except the FILLER item.
2. The series of the names of the previous qualifiers for d1 are the same as for d2.
3. The REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is not specified. The data items do not belong to those containing the REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause.

Overlapping of Operands

The data items defined in different data description entries may be specified in the sending and receiving items of one statement. Also, the data items may share the partial or entire storage field. If so, the result of executing the statement is undefined. The data item defined in the same data description entry may be specified in the sending and receiving items of one statement. If so, the result of executing the statement may not be defined. The rules for this are explained in the item under “General rules for individual statements.”

INVALID KEY Phrase

The DELETE statement, or the READ, REWRITE, START, or WRITE statement for random access may be executed for a relative or indexed file. If so, an invalid key condition may occur. An invalid key condition can be detected by written INVALID KEY phrase in these input-output statements. The NOT INVALID KEY phrase can check whether an input-output statement was executed successfully without an invalid key condition.

This section explains the operation of these input-output statements that may cause an invalid key condition under the following three headings:

- Operation to be performed if an invalid key condition occurs
- Operation to be performed if an exception condition other than an invalid key condition occurs
- Operation to be performed if no invalid key condition or no other exception condition occurs

Operation to be Performed if an Invalid Key Condition Occurs

If an invalid key condition occurs, an input-output statement is executed unsuccessfully. After the value indicating an invalid key condition is stored in the input-output status, control is transferred depending on whether:

a. INVALID KEY phrase is written in the input-output statement.

b. The related USE AFTER STANDARD EXCEPTION procedure is provided.

The following table shows where control is transferred if an invalid key condition occurs:

Whether INVALID KEY Phrase is Specified	**Whether the USE AFTER STANDARD EXCEPTION Procedure is Provided**	**Where Control is Transferred if an Invalid Key Condition Occurs**
Specified	Provided or not provided	Control is transferred to an imperative statement with INVALID KEY phrase. After the imperative statement is executed, control is transferred to the end of an input-output statement. (*1)
Not specified	provided	Control is transferred to the USE AFTER STANDARD EXCEPTION procedure. Control is transferred according to the rules of the USE statement.
Not provided	Not provided	If the file contains the FILE STATUS clause, control is transferred to the end of an input-output statement. Otherwise, the execution result is undefined.

*1 In the imperative statement, a procedure branching statement or a conditional statement that causes explicit control transfer may be executed. If so, control is transferred according to the rules of the statement.

Operation to be Performed if an Exception Condition Other Than an Invalid Key Condition Occurs

If an exception condition other than an invalid key condition occurs, an input-output statement is executed unsuccessfully. After the value indicating an exception condition is stored in the input-output status, control is transferred according to the following rules:

1. If the related USE AFTER STANDARD EXCEPTION procedure is written, control is transferred to the procedure. Then, control is transferred according to the rules of the USE statement.

2. The related USE AFTER STANDARD EXCEPTION procedure may not be written, and the file may contain the FILE STATUS clause. If so, control is transferred to the end of the input-output statement. If the FILE STATUS clause is not specified, the execution result is undefined.

Operation to be Performed if No Invalid Key Condition or No Other Exception Condition Occurs

If no invalid key condition or no other exception condition occurs, an input-output statement is executed successfully. After the value indicating the fact is stored in the input-output status, control is transferred according to the following rules:

1. If the input-output statement contains a NOT INVALID KEY phrase, control is transferred to an imperative statement with a NOT INVALID KEY phrase. After the imperative statement is executed, control is transferred to the end of the input-output statement. However, in the imperative statement, a procedure branching statement or a conditional statement that causes explicit control transfer may be

executed. If so, control is transferred according to the rules of the statement.

2. If the input-output statement does not contain a NOT INVALID KEY phrase, control is transferred to the end of the input-output statement.

AT END Phrase

The READ statement for sequential access may be executed for a sequential, relative, or indexed file. If so, an at end condition may occur. An at end condition can be detected by written AT END phrase in the READ statement. The NOT AT END phrase can check whether an input-output statement was executed successfully without an at end condition.

This section explains the operation of the READ statement under the following three headings:

a. Operation to be performed if an at end condition occurs

b. Operation to be performed if an exception condition other than an at end condition occurs

c. Operation to be performed if no at end condition or no other exception condition occurs

Operation to be Performed if an At End Condition Occurs

If an at end condition occurs, the READ statement is executed unsuccessfully. After the value indicating an at end condition is stored in the input-output status, control is transferred depending on whether:

a. AT END phrase is specified in the READ statement.

b. The related USE AFTER STANDARD EXCEPTION procedure is provided.

The following table shows where control is transferred if an at end condition occurs:

Whether AT END Phrase is Specified	**Whether the USE AFTER STANDARD EXCEPTION Procedure is Provided**	**Where Control is Transferred if an At End Condition Occurs**
Specified	Provided or not provided	Control is transferred to an imperative statement with an AT END phrase. After the imperative statement is executed, control is transferred to the end of the READ statement. (*1)
Not specified	provided	Control is transferred to the USE AFTER STANDARD EXCEPTION procedure. After the USE AFTER STANDARD EXCEPTION procedure is executed, control is transferred to the end of the READ statement.
Not specified	Not provided	If the file contains the FILE STATUS clause, control is to the end of the READ statement. Otherwise, the execution result is undefined.

*1: In the imperative statement, a procedure branching statement or a conditional statement that causes explicit control transfer may be executed. If so, control is transferred according to the rules of the statement.

Operation to be Performed if an Exception Condition Other than an At End Condition Occurs

If an exception condition other than an at end condition occurs, the READ statement is executed unsuccessfully. After the value indicating an exception condition is stored in the input-output status, control is transferred according to the following rules:

1. If the related USE AFTER STANDARD EXCEPTION procedure is written, control is transferred to the procedure. Then, control is transferred according to the rules of the USE statement.

2. The related USE AFTER STANDARD EXCEPTION procedure may not be written, and the file may contain the FILE STATUS clause. If so, control is transferred to the end of the READ statement. If the FILE STATUS clause is not specified, the execution result is undefined.

Operation to be Performed if No At End Condition or No Other Exception Condition Occurs

If no at end condition or no other exception condition occurs, the READ statement is executed successfully. After the value indicating the fact is stored in the input-output status, control is transferred according to the following rules:

1. If the READ statement contains a NOT AT END phrase, control is transferred to an imperative statement with a NOT AT END phrase. After the imperative statement is executed, control is transferred to the end of the READ statement. However, in the imperative statement, a procedure branching statement or a conditional statement that causes explicit control transfer may be executed. If so, control is transferred according to the rules of the statement.

2. If the READ statement does not contain a NOT AT END phrase, control is transferred to the end of the READ statement.

Statements

This section explains each statement.

ACCEPT Statement (Nucleus)

The ACCEPT statement enters small amounts of data.

[Format 1] To enter data:

$$\underline{\text{ACCEPT}}\ \text{identifier-1}\ \{[\text{FORM mnemonic-name-1}]\}$$

[Format 2] To obtain date, day of the week, and time:

$$\underline{\text{ACCEPT}}\ \text{identifier-2}\ \underline{\text{FORM}} \left\{ \begin{array}{l} \underline{\text{DATE}} \\ \underline{\text{DAY}} \\ \underline{\text{DAY-OF-WEEK}} \\ \underline{\text{TIME}} \end{array} \right\}$$

Syntax Rules

1. identifier-1 must be an alphabetic item, alphanumeric data item, zoned decimal item, packed decimal item, binary item, external Boolean item, or fixed-length group item.
2. Associate mnemonic-name-1 with the function-name CONSOLE or SYSIN in the special-names paragraph of the environment division. For details of the maximum length of identifier-1 that can be used if CONSOLE is specified as mnemonic-name-1, see Appendix B, “Quantitative System Restrictions.”

3. identifier-2 must be an alphanumeric data item, alphanumeric edited data item, numeric edited data item, zoned decimal item, packed decimal item, binary item, or fixed-length group item.

General Rules

Rules for Format 1

1. The ACCEPT statement reads data from either of the following hardware devices, then stores it in identifier-1. The read data is not edited, and not checked for errors.

   a. CONSOLE (system logical console)

   b. SYSIN (system logical input unit)

2. If the FROM phrase is omitted, the mnemonic-name for SYSIN is assumed.

3. If the input-output device specified as mnemonic-name-1 is the same as the input device specified in the READ statement, the result is undefined.

4. If the mnemonic-name corresponding to CONSOLE is specified as mnemonic-name-1, processing is performed in the following order:

   a. A message generated by the system is automatically displayed on the system logical console, and execution of the ACCEPT statement is interrupted.

b. When the user enters a message on the system logical console, the ACCEPT statement restarts. The message is stored in identifier-1 from left regardless of the description of the PICTURE clause. If the input message is shorter than identifier-1, the remaining portion of identifier-1 is padded with blanks. If the input message is longer than identifier-1, the message is truncated on the right to the length of identifier-1.

5. If the mnemonic-name corresponding to SYSIN is specified as mnemonic-name-1, records are repeatedly read, and sequentially stored in receiving data items. This processing is performed until the receiving data items are filled with input data, or until all the records have been read.

Rules for Format 2

1. The ACCEPT statement for format 2 copies information described in the FROM phrase into the data item for identifier-2. This is done according to the rules for the MOVE statement.

   DATE (year, month, and day), DAY (year and day), DAY-OF-WEEK (day of the week), and TIME (time) are virtual data items. These can be used only in the ACCEPT statement, and not in any other part of a COBOL program.

2. DATE is treated as an unsigned six-digit zoned decimal integer item. If DATE is written, the last two digits of year, the month, then the day are copied into identifier-2 (A total of six digits). For example, if the date is October 1, 1994, the value copied to identifier-2 is 941001.

3. DAY is treated as an unsigned five-digit zoned decimal integer item. If DAY is written, the following are copied into identifier-2:

   a. The last two digits of a year

   b. Then, the total number of the days since January 1

      (A total of five digits). For example, if the date is October 1, 1994, the value copied to identifier-2 is 94274.

4. TIME is treated as an unsigned eight-digit zoned decimal integer item. If TIME is written, the following are copied into identifier-2:

   a. Hour (24-hour format) - Minute

   b. Second

   c. Hundredths of a second

      (A total of eight digits). For example, if the time is precisely 2:41 p.m., the value copied to identifier-2 is 14410000.

5. DAY-OF-WEEK is treated as an unsigned one-digit zoned decimal integer item. If DAY-OF-WEEK is written, one digit indicating a day of the week is copied into identifier-2. If day of the week is Monday, 1 is copied. If it is Tuesday, 2 is copied. If it is Sunday, 7 is copied.

ACCEPT Statement (Screen Operation)

The ACCEPT statement enters data on the screen.

[Format]

```
ACCEPT data-name-1
   [     { | LINE NUMBER   { identifier-1 }   | } ]
   [ AT  { |               { integer-1    }   | } ]
   [     { |                                  | } ]
   [     { | COLUMN NUMBER { identifier-2 }   | } ]
   [     { |               { integer-2    }   | } ]
  [ON EXCEPTION imperative-statement-1]
  [NOT ON EXCEPTION imperative-statement-2]
  [END-ACCEPT]
```

Syntax Rules

1. data-name-1 must be a screen item defined in the screen section. The screen item must be either of the following:

   a. Elementary screen item for which the TO phrase or the USING phrase is written in the PICTURE clause

   b. Group screen item containing an elementary screen item for which the TO phrase or the USING phrase is written in the PICTURE clause

2. data-name-1 can be qualified.

3. integer-1 and integer-2 must be unsigned integers.

4. identifier-1 and identifier-2 must be unsigned integer items.

5. The LINE NUMBER phrase and COLUMN NUMBER phrase can be written in any order.

General Rules

1. The ACCEPT statement reads data from the screen field corresponding to the screen item for data-name-1. Then, it copies the read data into the data item (written in TO or USING of the PICTURE clause) for data-name-1. If a group screen item is specified as data-name-1, all the input and update items belonging to the screen item are copied. Data is copied from a screen item into a data item according to the copying rules.
2. If a group screen item is specified as data-name-1, the operation is as follows:
    a. Once the ACCEPT statement has been executed, data can be entered to all the input and update items belonging to the group screen item.
    b. To enter data, move the cursor. Do this in order of the screen positions specified in the LINE NUMBER and COLUMN NUMBER clauses of the screen data description entry.
    c. If the AUTO clause is specified as data-name-1, the cursor is automatically moved to the next input or update field.
3. The ACCEPT statement is completed by pressing the input key.
4. If the AUTO clause is valid for the screen item, the ACCEPT statement is completed when:
    a. Data is entered to the field of the last screen item of the group screen item for data-name-1.

   b. Data is entered to the field of the elementary screen item for data-name-1.

5. The LINE NUMBER phrase specifies the screen line number corresponding to the screen item for data-name-1. In doing this, the first line of the physical screen is considered to be 1.

6. The COLUMN NUMBER phrase specifies the screen column number corresponding to the screen item for data-name-1. In doing this, the first column of the physical screen is considered to be 1.

7. If the AT phrase is omitted, data is entered starting at the first line and first column of the screen.

8. If the ON EXCEPTION phrase is written, and input does not terminate normally, imperative-statement-1 is executed.

9. If the NOT ON EXCEPTION phrase is written, and input terminates normally, imperative-statement-2 is executed. For details of the normal termination status, see the section titled “CRT STATUS clause.”

ACCEPT Statement (Operation on Command Line Arguments and Environmental Variables)

The ACCEPT statement enters the number or values of command line arguments or the values of environmental variables.

[Format]

```
ACCEPT identifier-1 FROM mnemonic-name-1
    [ON EXCEPTION imperative-statement-1]
    [NOT ON EXCEPTION imperative-statement-2]
    [END-ACCEPT]
```

Syntax Rules

1. Associate mnemonic-name-1 with one of the following function-names in the special-names paragraph of the environment division:

   a. ARGUMENT-NUMBER

   b. ARGUMENT-VALUE

   c. ENVIRONMENT-VALUE

2. identifier-1 must be a data item that satisfies the following conditions:

   a. The mnemonic-name corresponding to the function-name ARGUMENT-NUMBER may be specified as mnemonic-name-1. If so, identifier-1 must be an unsigned integer item.

b. The mnemonic-name corresponding to the function-name ARGUMENT-VALUE or ENVIRONMENT-VALUE may be specified as mnemonic-name-1. If so, identifier-1 must be a group item not containing a variable occurrence data item, or an alphanumeric data item.

3. The mnemonic-name corresponding to the function-name ARGUMENT-NUMBER may be specified as mnemonic-name-1. If so, the ON EXCEPTION and NOT ON EXCEPTION phrases must not be written.

General Rules

Rule Applied if Mnemonic-name-1 is Associated with the Function-name ARGUMENT-NUMBER

The number of the arguments specified in the command line is copied into identifier-1 according to the copying rules.

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ARGUMENT-VALUE

1. The value of the argument indicated by the current argument position indicator is copied into identifier-1 according to the copying rules.

2. Execution of the ACCEPT statement increments the value of the argument position indicator by one.

3. The value of the argument indicator before execution of the ACCEPT statement may have exceeded the position of the last argument in the command line. If so, and the ON EXCEPTION phrase is written, imperative-statement-1 is executed.

4. The exception condition in (3) may not occur. If so, and the NOT ON EXCEPTION phrase is written, imperative-statement-2 is executed.

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ENVIRONMENT-VALUE

1. Before execution of the ACCEPT statement, the DISPLAY statement where the mnemonic-name corresponding to the function-name ENVIRONMENT-NAME is specified must be executed. Execution of the ACCEPT statement copies the value of the environment variable positioned by the preceding DISPLAY statement into identifier-1. This is done according to the copying rules.
2. If the ON EXCEPTION phrase is written, imperative-statement-1 is executed under either of the following conditions:
   a. Before execution of the ACCEPT statement, the DISPLAY statement for positioning the environmental variable is not executed.
   b. The environmental variable specified in the preceding DISPLAY statement does not exist.
3. The exception condition in (2) may not occur. If so, and the NOT ON EXCEPTION phrase is written, imperative-statement-2 is executed.

ADD Statement (Nucleus)

The ADD statement obtains the result of addition.

[Format 1] To replace the result of addtion with the augend:

$$\begin{array}{l}
\underline{\text{ADD}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\}\ldots\ \underline{\text{TO}}\ \{\text{identifier-2}\ \ [\underline{\text{ROUNDED}}]\}\ \ldots\\
\qquad [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-1}]\\
\qquad [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\
\qquad [\underline{\text{END-ADD}}]
\end{array}$$

[Format 2] To store the result of addtion in a data item different from that for the augend:

$$\begin{array}{l}
\underline{\text{ADD}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\}\ldots\ \text{TO}\ \left\{\begin{array}{l}\text{identifier-2}\\ \text{integer-2}\end{array}\right\}\ \underline{\text{GIVING}}\\
\qquad \{\text{identifier-3}\ \ [\underline{\text{ROUNDED}}]\}\ \ldots\\
\qquad [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-1}]\\
\qquad [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\
\qquad [\underline{\text{END-ADD}}]
\end{array}$$

[Format 3] To add the corresponding data items belonging ti two group items:

$$\begin{array}{l}
\underline{\text{ADD}} \left\{\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right\}\ \text{identifier-1}\ \ \underline{\text{TO}}\ \ \text{identifier-2}\\
\qquad [\underline{\text{ROUNDED}}]\\
\qquad [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-1}]\\
\qquad [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\
\qquad [\underline{\text{END-ADD}}]
\end{array}$$

Syntax Rules

1. For formats 1 and 2, identifier-1 and identifier-2 must be numeric data items. For format 3, identifier-1 and identifier-2 must be group items.
2. identifier-3 must be a numeric data item or a numeric edited data item.
3. literal-1 and literal-2 must be numeric literals.
4. CORRESPONDING is synonymous with CORR. Either can be written.

General Rules

Rules for Format 1

The ADD statement for format 1 adds the sum of the individual operands before TO to identifier-2, then stores the result in identifier-2. This addition is done in the order in the written sequence of identifier-2.

Rules for Format 2

The ADD statement for format 2 adds the sum of the individual operands before TO to the operand after TO, then stores the result in identifier-3. The result of this addition is stored in the written sequence of identifier-3.

Rules for Format 3

The ADD statement for format 3 obtains the sum of a data item belonging to identifier-1 and the corresponding data item belonging to identifier-2. The added data items must have the same name qualifier series. A data item belonging to identifier-1 is treated as an addend, and that belonging to identifier-2 is treated as an augend. The sum is stored in the data item belonging to identifier-2. This produces the same result of writing the ADD statement for each corresponding identifier.

Rules Common to Format 1 to Format 3

1. The END-ADD phrase delimits the scope of the ADD statement.
2. For details of the ROUNDED, ON SIZE ERROR, and CORRESPONDING phrases, and the operation and copying rules, see the section titled “Common Statement Rules.”

ALTER Statement (Nucleus)

The ALTER statement changes the predefined processing order. It is an obsolete language element.

[Format]

```
ALTER
    {procedure-name-1  TO  [PROCEED  TO]
                                procedure-name-2} …
```

Syntax Rules

1. procedure-name-1 must be a paragraph name. The paragraph must consist of only one sentence containing one GO TO statement without DEPENDING.
2. procedure-name-2 must be a paragraph or section-name of the procedure division.

General Rule

The ALTER statement changes the destination of the GO TO statement written in the paragraph for procedure-name-1 to procedure-name-2.

CALL Statement (Inter-program Communication)

The CALL statement transfers control to another program in the run unit.

[Format 1] ON OVERFLOW phrase

```
CALL { identifier-1 } [WITH { C       } LINKAGE]
     { literal-1    }       { PASCAL  }
                            { STDCALL }

  [       { [BY REFERENCE] {identifier-2} ...              }     ]
  [ USING { BY CONTENT { identifier-2 } ...                } ... ]
  [       {            { literal-2    }                    }     ]
  [       { BY VALUE { identifier-3} ...                   }     ]
  [ON OVERFLOW imperative-statement-1]
  [END-CALL]
```

[Format 2] ON EXCEPTON phrase

```
CALL { identifier-1 } [WITH { C       } LINKAGE]
     { literal-1    }       { PASCAL  }
                            { STDCALL }

  [       { [BY REFERENCE] {identifier-2} ...              }     ]
  [ USING { BY CONTENT { identifier-2 } ...                } ... ]
  [       {            { literal-2    }                    }     ]
  [       { BY VALUE {identifier-3} ...                    }     ]
  [ON EXCEPTON imperative-statement-1]
  [NOT ON EXCEPTON imperative-statement-2]
  [END-CALL]
```

Win The WITH phrase can be specified.

Syntax Rules

1. identifier-1 must be an alphanumeric data item. The value of identifier-1 must observe the rules for describing program names.
2. identifier-2 and identifier-3 must be data items defined in the file, working-storage, or linkage section.
3. identifier-2 must be a data item at level 01 or 77, or an elementary item at any level. However, for this compiler, a group item at any level can be written.
4. The internal Boolean item to be specified as identifier-2 must be defined so that it starts on a byte boundary.
5. identifier-3 must be one of the following:
   a. Nine-digit-or-less numeric data item for which use is COMPUTATIONAL-5
   b. One-character alphabetic item
   c. One-character alphanumeric data item
   d. One-digit zoned decimal integer item without SEPARATE
6. literal-1 must be a nonnumeric literal. The value of literal-1 must observe the rules for describing program names.
7. literal-2 must be a nonnumeric literal, hexadecimal nonnumeric literal, or national nonnumeric literal.
8. For details of the rules for describing program names, see the section titled “Program-ID paragraph.” For details of the maximum number of the operands in the USING phrase, see Appendix B, “Quantitative System Restrictions.”

General Rules

1. The CALL statement calls another program in the run unit. The program containing the CALL statement is called a calling program. The program called by execution of the CALL statement is called a called program. The name of a called program is specified as identifier-1 or literal-1. To pass parameters to a called program, write the USING phrase. To check whether a called program can be executed, write the ON OVERFLOW, ON EXCEPTION, or NOT ON EXCEPTION phrase.

2. If a called program is a COBOL program, the contents of literal-1 and identifier-1 must be either of the following:

   a. The program name written in the program-ID paragraph of the called program

   b. The literal written in the ENTRY statement

3. When the CALL statement is executed, if the program specified in the CALL statement can be executed, control is transferred to the called program.

4. Where control is transferred when the CALL statement is executed depends on whether the ON OVERFLOW, ON EXCEPTION, and NOT ON EXCEPTION phrases are specified. The following table shows where control is transferred when the CALL statement is executed.

Whether the ON EXCEPTION or ON OVERFLOW phrase is specified	Whether the NOT ON EXCEPTION phrase is specified	Operation of the CALL statement	
		If the called program cannot be executed	If the called program can be executed
Specified	Specified	1. Control is transferred to imperative-statement-1. 2. After imperative-statement-1 is executed, control is transferred to the end of the CALL statement. (*1)	Control is transferred to the called program. After having been returned by the called program, control is transferred to imperative-statement-2. After imperative-statement-2 is executed, control is transferred to the end of the CALL statement. (*2)
Specified	Not specified	1. Control is transferred to imperative-statement-1. 2. After imperative-statement-1 is executed, control is transferred to the end of the CALL statement. (*1)	Control is transferred to the called program. After having been returned by the called program, control is transferred to the end of the CALL statement.
Not specified	Specified	Operation of the CALL statement is undefined.	Control is transferred to the called program. After having been returned by the called program, control is transferred to imperative-statement-2. After imperative- statement-2 is executed, control is transferred to the end of the CALL statement. (*2)
Not specified	Not specified	Operation of the CALL statement is undefined.	Control is transferred to the called program. After having been returned by the called program, control is transferred to the end of the CALL statement.

*1 In imperative-statement-1, a procedure branching statement or a conditional statement that causes explicit transfer of control may be written. If so, control is transferred according to the rules for the statement.

*2 In imperative-statement-2, a procedure branching statement or a conditional statement that causes explicit transfer of control may be written. If so, control is transferred according to the rules for the statement.

5. Program names can be duplicated in a run unit. If the program name specified in the CALL statement is duplicated in a run unit, the program to be called is determined in the following order. This is done according to the rules for the scope of names.

   a. The programs directly included in the program containing the CALL statement are checked. If one of these programs is specified in the CALL statement, the program is called.

   b. The program containing the CALL statement is directly or indirectly included in some programs. These programs are checked to see if they directly include programs having the same attributes as the program containing the CALL statement. If one of these programs with the same attributes is specified in the CALL statement, the program is called.

   c. Another compilation unit is checked to see if it contains the program specified in the CALL statement. If such a program is found, the program is called.

6. The called program may not have the initialization attribute. If so, the program, and the programs directly or indirectly included in it are initialized under either of the following conditions:

   a. They are called for the first time in the run unit.

   b. They are called for the first time after the CANCEL statement is executed for the called program.

   c. Otherwise, the following program has not been changed since the last time they returned control to the calling program:

      - program(s) called for the second time and after in the run unit

      - program(s) directly or indirectly included in such program(s)

7. If the called program has the initialization attribute, the program, and the individual programs directly or indirectly included in it are always initialized when called.

8. If the called program is initialized, the files for its internal file connector are not open. Otherwise, the statuses and positioning of the files for the internal file connector have not been changed since the program execution terminated for the last time.

   For details of the initial program status, see the section titled “Initial program status.”

9. No call from a program and no return from the called program change the statuses and positioning of the files for external file connectors.

10. The USING phrase can be written in the CALL statement only if the USING phrase is written in either of the following of the called COBOL program:

    a. The header of the procedure division

    b. ENTRY statement

    In this case, the number of the operands in the USING phrase of the calling program must equal to that for the called program.

11. In the operands of the USING phrase in the CALL statement, specify the parameters to be passed to the called program. The called program, if it is a COBOL program, receives parameters using the header of the procedure division or the USING phrase of the ENTRY statement. Each parameter is associated with other parameters in the order in which it is written in the USING phrase. For example, to pass two parameters to a COBOL program, write them in the USING phrase of the CALL statement. The first and second parameters are associated with the first and second data-names in the header of the procedure division of the called program.

12. The values of the parameters written in the USING phrase of the CALL statement can be used in the called program. This can be done when the CALL statement is executed.

13. One or more parameters can be written in the BY CONTENT, BY REFERENCE, and BY VALUE phrases. These phrases can also be combined. If these phrases are combined, one phrase is valid for parameters until another corresponding BY CONTENT, BY REFERENCE, or BY VALUE phrase appears. If the first parameter is not preceded by the BY CONTENT, BY REFERENCE, or BY VALUE phrase, the BY REFERENCE phrase is assumed.

14. The BY REFERENCE phrase may be specified explicitly or implicitly. If this phrase is written, the parameters for the called program and those for the calling program are assumed to occupy the same storage fields. The parameter character position count for the calling program must be the same as for the called program.

15. If the BY CONTENT phrase is written, the values of the parameters specified in the calling program can be referenced by the called program. This can be done by using the data items written in the header of the procedure division or the ENTRY statement of the called program. However, if the called program changes these data items, the values are not reflected in the corresponding parameters of the calling program. The parameter use and character position count for the calling program must be the same as for the called program.

16. literal-2 may be specified in the BY CONTENT phrase. If so, the corresponding data item attribute and character position count for the called program must be the same as those for literal-2.

17. If the BY VALUE phrase is written, the values of parameters are passed directly to the called program. The called program must be written in a language that enables value parameters to be received. The called program may change the passed values using the BY VALUE phrase. If so, the values of the corresponding data items for the calling program are not changed.

18. The called program cannot execute the CALL statement that directly or indirectly calls the calling program.

19. The CALL statement in the declarative must not directly or indirectly call the incomplete program to which control was passed.

20. The END-CALL phrase delimits the scope of the CALL statement.

21. To transfer control to the beginning of the procedure division of the called program, specify the name of the called program as literal-1 or identifier-1. You may want to transfer control to entry point other than the beginning of the procedure division. If so, specify the secondary entry name specified in the ENTRY statement as literal-1 or identifier-1.

22. When you intend to call a program that can reference a global name, do not write the name in the USING phrase of the CALL statement.

Rules for the WITH Phrase

1. The WITH phrase indicates that the generated program is called according to the specified calling conventions. If the WITH phrase is omitted, the COBOL linkage conventions are applied.

2. The calling conventions between the calling and called programs must be the same. If they are different the execution result is undefined.

3. The WITH phrase must not be used in calling a contained program or a program that might be a contained program.

CANCEL Statement (Inter-program Communication)

The CANCEL statement initializes a program to be called next time.

[Format]

$$\underline{\text{CANCEL}} \left\{ \begin{matrix} \text{identifier-1} \\ \text{literal-1} \end{matrix} \right\} \ldots$$

Syntax Rules

1. identifier-1 must be an alphanumeric data item. The value of identifier-1 must observe the rules for describing program names.
2. literal-1 must be a nonnumeric literal. The value of literal-1 must observe the rules for describing program names.
3. For details of the rules for describing program names, see the section titled “Program-ID paragraph.”

General Rules

1. The CANCEL statement cancels the logical relation between a run unit and a program. Specify the name of the program to be canceled as identifier-1 or literal-1. Execution of the CANCEL statement cancels the logical relation between the following:

   a. The program specified as identifier-1 or literal-1

   b. The run unit for the program containing the CANCEL statement

2. The contents of identifier-1 and literal-1 must be the names of programs that can be called from the program containing the CANCEL statement. For details of the programs that can be called, see the section titled “CALL statement (inter-program communication).”

3. Execution of the CANCEL statement cancels the program specified in the CANCEL statement, and all the programs called by the program., This produces the same result as executing CANCEL statements in the sequence from called program to calling program of the nesting written in separately compiled programs.

4. After the CANCEL statement has been executed successfully, the program specified in the CANCEL statement may be called again from the run unit. If so, the program is initialized. These rules are also applied to the implicit CANCEL statement.

5. The program specified in the CANCEL statement must not directly or indirectly reference the called program in which the EXIT PROGRAM statement is not executed. For example, CANCEL”A” cannot be executed in the program called during execution of program A.

6. The logical relation with the program canceled by executing the CANCEL statement is established only when the CALL statement is re-executed.

7. The program called by the CALL statement is canceled under any of the following conditions:

   a. The CANCEL statement for the program is executed.

   b. The run unit including the program terminates.

   c. The program has the initialization attribute, and executes the EXIT PROGRAM statement.

   For cases (b) and (c), the CANCEL statement is called the implicit CANCEL statement.

CLOSE Statement (Sequential, Relative, Indexed, and Presentation Files, and Report Writer Module)

The CLOSE statement terminates file processing.

[Format 1] Sequential and report files

$$\underline{\text{CLOSE}} \left\{ \text{file-name-1} \left[\begin{array}{l} \left\{\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right\} [\text{ FOR } \underline{\text{REMOVAL}}] \\ \underline{\text{WITH}} \left\{\begin{array}{l}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{array}\right\} \end{array} \right] \right\} \ldots$$

[Format 2] Relative, indexed, and presentation file

$$\underline{\text{CLOSE}}\ \{\text{file-name-1}\ \ [\text{WITH}\ \ \underline{\text{LOCK}}]\,\}\ \ldots$$

Syntax Rules

Rules Common to Format 1 and Format 2

In a list of file-name-1, a file specified differently in the ORGANIZATION or ACCESS MODE clause can be specified.

Rules for Format 1

1. If the WITH NO REWIND phrase is written, file-name-1 must be a record sequential file.
2. If the REEL or UNIT phrase is written, file-name-1 must be a record sequential file or a print file without the FORMAT clause.

General Rules

Operation of the CLOSE statement depends on the storage medium. Files are explained under the following three headings:

1. Unit record (sequential, relative, indexed, presentation, and report files)

   Files on an input-output medium for which the concepts of rewinding and volumes are meaningless

2. Sequential single volume (sequential and report files)

   Sequential file included entirely in one volume

3. Sequential multi-volume (sequential and report files)

   Sequential file spanning two or more volumes

Rules Common to Individual Files

1. Execution of the CLOSE statement updates the input-output status for file-name-1.
2. The CLOSE statement in which two or more file-name-1 are written may be executed. If so, this produces the same result as executing the separate CLOSE statement for each file in the written sequence of file-name-1.

Rules Common to Relative, Indexed, and Presentation Files, and Sequential and Report Files for Unit Records

1. The file for file-name-1 must be opened before execution of the CLOSE statement.
2. The following processing is performed for the file for file-name-1.
   1. Closing of files

      The standard termination procedure is executed.
   2. Locking of files

      If the LOCK phrase is written, a file is locked, and reopened in the run unit.
3. After the CLOSE statement has been executed successfully, the file for file-name-1 is put in the following status:
   1. The record fields for file-name-1 cannot be referenced.
   2. All the records retained by the file connector for file-name-1 and the file for file-name-1 are unlocked for other run units.
   3. The open file for file-name-1 is removed, and is not associated with the file connector for file-name-1 any more.
4. If the record fields are referenced after the CLOSE statement has been executed unsuccessfully, the result is undefined.
5. For sequential, relative, and indexed files, an undefined file opened in the input mode may not exist. If so, the file does not terminate, and the file position indicator remains unchanged.

Rules for Sequential and Report Files for Volumes

1. The following table lists the results of executing the CLOSE statement for volume files.

Method of Writing the CLOSE Statement	Sequential Single Volume Unit	Sequential Multi-Volume Unit
CLOSE	(c), (g)	(a), (c), (g)
CLOSE WITH LOCK	(c), (e), (g)	(a), (c), (e), (g)
CLOSE WITH NO REWIND	(b), (c)	(a), (b), (c)
CLOSE REEL / UNIT	(f), (g)	(f), (g)
CLOSE REEL / UNIT FOR REMOVAL	(d), (f), (g)	(d), (f), (g)

The parenthesized alphabetic characters in the table correspond to the following. If execution results depend on the open mode, there is an explanation for each open mode. Otherwise, the explanation applies to the files opened in any open mode.

a. Influence on the preceding volume

A file may be opened in the input or input-output mode. If so, all the previous volumes are closed except when the file was previously processed by the CLOSE statement with the REEL / UNIT phrase. If the current volume is not the last volume of the file, the subsequent volumes are not processed.

A file may be opened in the output mode. If so, all the previous volumes are closed except when the file was previously processed by the CLOSE statement with the REEL / UNIT phrase.

b. The current volume is not rewound.

The current volume is not rewound, and left in the same position.

c. Closing of files

If a file is opened in the input or input-output mode: If the file is in the termination position, and a label record is specified, the label is processed based on the standard label procedure. Then, the standard termination procedure is executed.

Only the standard termination procedure is executed, and termination label processing is not performed under either of the following conditions:

- The file is in the termination position, and no label record is specified
- The file is not in the termination position.

If a file is opened in the output mode: If a label record is specified, the label is processed based on the standard label procedure. Then, the standard termination procedure is executed. If no label record is specified, the standard termination procedure is executed without processing a label.

If a file is opened in the extended mode: The standard termination procedure is executed.

d. Removal of volumes

The current volume is rewound, if possible, and logically removed from this run unit. To process the volume again in the proper file order, execute the CLOSE statement without the REEL / UNIT phrase. Then, execute the OPEN statement. For this compiler, the FOR REMOVAL phrase is treated as a comment.

e. Locking of files

A file is locked, and is not reopened in the run unit.

f. Closing of volumes

If a file is opened in the input or input-output mode:

- The current volume may be the last volume or the only volume of the file. Alternatively, the reel may be on a unit record storage medium. If so, the volume is not replaced, and the volume indicator remains unchanged.
- Another volume in the file, if any, is replaced, and the volume indicator is updated so that it indicates the next volume of the file. Then, the standard start volume label procedure is executed. If the current volume does not contain a data record, the volume is replaced again.

If a file (volume storage medium) is opened in the output mode:

- The standard termination volume label procedure is executed.
- The volume is replaced. The volume indicator is updated so that it indicates a new volume.
- The standard start volume label procedure is executed.
- The next WRITE statement prepares for writing of a record in the next position of the next volume.

g. Rewinding

The current volume is physically positioned at the beginning.

2. Execution of the CLOSE statement unlocks all the records retained by the file connector for file-name-1 and the file for file-name-1. This is done for other run units.

3. If there is no undefined file opened in the input mode, no file termination and volume procedures are executed for the file. The file position indicator and the volume indicator remain unchanged.

4. If the REEL and UNIT phrases are omitted, once the CLOSE statement has been executed successfully, the file for file-name-1 enters the following status:

    a. The record field for file-name-1 cannot be referenced. If the record field is referenced after the CLOSE statement has been executed unsuccessfully, the result is undefined.

    b. The open file for file-name-1 is removed, and it is not associated with the file connector for file-name-1 any more.

Rules for Report Files

Before execution of the CLOSE statement, all the report file reports that have started must be terminated by executing the TERMINATE statement.

COMPUTE Statement (Nucleus)

The COMPUTE statement evaluates an arithmetic or Boolean expression.

[Format 1] To evaluate an arithmetic expression:

```
COMPUTE {identifier-1 [ROUNDED]} ... = arithmetic-expression-1
 [ON SIZE ERROR imperative-statement-1]
 [NOT ON SIZE ERROR imperative-statement-2]
 [END-COMPUTE]
```

[Format 2] To evaluate a Boolean expression:

```
COMPUTE {identifier-2} ... = Boolean-expression-1
```

Note: "=" is a key word, but not underlined for avoiding confusion with other symbols.

Syntax Rules

1. identifier-1 must be a numeric data item or a numeric edited data item.

2. identifier-2 must be a Boolean item.

General Rules

Rules for Format 1

1. The COMPUTE statement for format 1 evaluates arithmetic-expression-1. Then, it stores the result in identifier-1 in the written sequence of identifier-1.

2. If only one identifier or literal is written in arithmetic-expression-1, the value of the identifier or the literal is stored in identifier-1.

Rules for Format 2

1. The COMPUTE statement for format 2 evaluates Boolean-expression-1. Then, it stores the result in identifier-2 in the written sequence of identifier-2 is written.
2. If only one identifier or literal is written in Boolean-expression-1, the value of the identifier or the literal is stored in identifier-2.

Rules Common to Format 1 and Format 2

1. The END-COMPUTE phrase delimits the scope of the COMPUTE statement.
2. See the section titled “Common Statement Rules” for details of the following:
   a. ROUNDED and ON SIZE ERROR phrases

   b. Arithmetic and Boolean expressions

   c. Rules for operations and copying

CONTINUE Statement (Nucleus)

The CONTINUE statement indicates that there is no executable statement.

[Format]

CONTINUE

Syntax Rule

The CONTINUE statement can be written in any place where a conditional or imperative statement can be written.

General Rule

The CONTINUE statement does not affect program execution.

DELETE Statement (Relative and Indexed Files)

The DELETE statement logically removes a record from a mass storage file.

[Format]

```
DELETE file-name-1 RECORD
   [INVALID KEY imperative-statement-1]
   [NOT INVALID KEY imperative-statement-2]
   [END-DELETE]
```

Syntax Rules

1. If a sequential access file is specified as file-name-1, the INVALID KEY or NOT INVALID KEY phrase must not be written.

2. A random access or dynamic access file may be specified as file-name-1. Also, the USE AFTER STANDARD EXCEPTION procedure for file-name-1 may not be written. If so, the INVALID KEY phrase must be written. However, for this compiler, the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

3. For an indexed file, file-name-1 must not be a random access file for which the RECORD KEY clause with the DUPLICATES phrase is specified.

General Rules

Rules Common to Relative and Indexed Files

1. The file for file-name-1 must be a mass storage file.

2. The file for file-name-1 must be opened in the input-output mode before execution of the DELETE statement.

3. If a sequential access file is specified as file-name-1, the READ statement must be executed for the file before execution of the DELETE statement. If the READ statement is executed successfully, execution of the DELETE statement removes the record read by the READ statement from a file.

4. An attempt to delete a record locked by another file connector forces the DELETE statement to be executed unsuccessfully. Then, the value indicating record locking is stored in the input-output status for file-name-1.

5. After the DELETE statement has been executed successfully, the applicable record is logically removed from a file. After that, the record cannot be referenced.

6. If the DELETE statement is executed, the file position indicator is not changed.

7. If the DELETE statement is executed, the following are not changed:

   a. The contents of the record field

   b. The contents of the data item specified in DEPENDING ON of the RECORD clause of the file description entry for file-name-1.

8. Execution of the DELETE statement updates the value of the input-output status for file-name-1.

9. If LOCK MODE IS AUTOMATIC is specified for the file for file-name-1 after the DELETE statement has been executed successfully, the existing records are unlocked.

10. An invalid key condition may occur during execution of the DELETE statement. If so, after the value indicating an invalid key condition is stored in the input-output status for file-name-1, control is transferred. This is done according to the rules described in the section titled “INVALID KEY phrase.”

11. No exception condition may occur during execution of the DELETE statement. If so, after the input-output status for file-name-1 is stored, control is transferred according to the rules described in the section titled “INVALID KEY phrase.”

12. The END-DELETE phrase delimits the scope of the DELETE statement.

Rules for Relative Files

A random access or dynamic access file may be specified as file-name-1. If so, the relative number of the record to be deleted must be stored in the relative key item before execution of the DELETE statement. Execution of the DELETE statement logically removes the record having the relative record number from a file. If the file does not contain the record having the relative number, an invalid key condition occurs, and execution of the DELETE statement is unsuccessful.

Rules for Indexed Files

A random access or dynamic access file may be specified as file-name-1. If so, the key value of the record to be deleted must be stored in the main record key item before execution of the DELETE statement. Execution of the DELETE statement logically removes the record having the key value from a file. If the file does not contain the record having the key value, an invalid key condition occurs, and the execution of DELETE statement is unsuccessful.

DISPLAY Statement (Nucleus)

The DISPLAY statement displays small amounts of data.

[Format]

$$\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} [\underline{\text{UPON}}\ \ \text{mnemonic-name-1}]$$

$$[\text{WITH}\ \ \underline{\text{NO}}\ \ \underline{\text{ADVANCING}}]$$

Syntax Rules

1. The numeric literal written in literal-1 must be an unsigned integer.

   However, for this compiler, a numeric literal with a sign or a fraction part can be specified as literal-1.

2. mnemonic-name-1 must be associated with one of the following function-names in the special-names paragraph of the environment division:

   a. CONSOLE

   b. SYSOUT

c. SYSERR

d. SYSPUNCH

General Rules

1. The DISPLAY statement transfers the contents of identifier-1 or literal-1 to a hardware device in the order in which they are specified.

2. If a figurative constant is written in literal-1, only one repetition of the figurative constant is displayed.

3. If identifier-1 is written, its content is displayed in the following format:

    a. A group item or display elementary item (excluding a signed zoned decimal item without the SEPARATE phrase) may be specified as identifier-1. If so, the content of identifier-1 is displayed as it is.

    b. A binary item, packed decimal item, index data item, or signed zoned decimal item without the SEPARATE phrase may be specified as identifier-1. If so, it is converted to an zoned decimal item with the SIGN LEADING SEPARATE phrase, and displayed in the following format:

If identifier-1 is signed:	s9(n) 9(m)
If identifier-1 is unsigned:	9(n) 9(m)
If identifier-1 is an index data item:	9(n)

    where “s” indicates a sign (+ or -), “9(n)” indicates the number of the digits in the integer part, and “9(m)” indicates the number of digits in the fraction part. For an index data item, the value of n is 9.

    c. An internal Boolean item specified as identifier-1 is converted to an external Boolean item, and displayed.

d. An internal floating-point data item specified as identifier-1 is converted to an external floating-point data item, and displayed in the following format:

   For a single-precision internal floating-point data item: s.9(8) Es99

   For a long-precision internal floating-point data item: s.9(17) Es99

   where “s” indicates a sign (+ or -), and “9(8)”, “9(17)” and “99” indicate the number of digits.

e. A national data item or a national edited data item specified as identifier-1 is displayed as if one of the following were specified as identifier-1:

   CHARACTER TYPE IS MODE-2 clause without a mnemonic-name

   CHARACTER TYPE IS MODE-1 clause without a mnemonic-name

f. A pointer data item specified as identifier-1 is displayed in hexadecimal.

4. If two or more operands are written after DISPLAY, the total byte count (after conversion, if conversion is necessary) of all operands is the transfer byte count.

5. Execution of the DISPLAY statement transfers and displays the contents of identifier-1 or literal-1 in the order in which they are written. How this is done is shown below. Transfer data means the contents of the operands written before UPON combined in the order in which they are written.

   a. If the transfer byte count is the same as the number of bytes that can be transferred at a time, data is transferred and displayed as is.

b. If the transfer byte count is greater than the number of bytes that can be transferred at a time, data is repeatedly transferred and displayed. Each repetition consists of aligning the left end of data to the leftmost position, then transferring the data. When the last data is transferred, the rule in (a) or (c) is applied.

c. The transfer byte count is less than the number of bytes that can be transferred at a time, the left end of data is aligned at the leftmost position, then the data is transferred and displayed. If the device does not accept variable-length records, the remaining portion is padded with spaces.

6. If the UPON phrase is omitted, the mnemonic-name corresponding to SYSOUT is assumed to be specified as mnemonic-name-1.

7. The WITH NO ADVANCING phrase may be omitted. If so, after the last operand has been transferred to the hardware device, it is positioned on the left end of the next line.

8. If the WITH NO ADVANCING phrase is written, after the last operand has been displayed, line feed is not performed. At this point, the hardware device performs line feed as follows:

   a. A hardware device that can be positioned at a particular character remains positioned immediately after the last character of the operand displayed last.

   b. A hardware device that cannot be positioned at a particular character only controls vertical line feed, if possible. If the hardware device can print new data over the old data, data is displayed over the line already displayed.

DISPLAY Statement (Screen Operation)

The DISPLAY statement displays data on the screen.

[Format]

```
DISPLAY data-name-1
    [     {| LINE NUMBER { identifier-1 }       |} ]
    [     {|             { integer-1    }       |} ]
    [ AT  {|                                    |} ]
    [     {| COLUMN NUMBER { identifier-2 }     |} ]
    [     {|               { integer-2    }     |} ]
   [END-DISPLAY]
```

Syntax Rules

1. data-name-1 must be a screen item defined in the screen section. The screen item must be one of the following:
   a. Elementary screen item for which the VALUE clause is specified
   b. Elementary screen item for which the FROM or USING phrase is written in the PICTURE clause.
   c. Group screen item containing the elementary screen item in (a) or (b)
2. data-name-1 can also be qualified.
3. integer-1 and integer-2 must be unsigned integers.
4. identifier-1 and identifier-2 must be unsigned integer items.
5. The LINE NUMBER and COLUMN NUMBER phrases can be written in any order. Either or both phrases can be written.

General Rules

1. The DISPLAY statement displays the following data in the screen field corresponding to the screen item for data-name-1.

   a. If a literal item is specified as data-name-1, the value specified in the VALUE clause is displayed.

   b. An output or update item may be specified as data-name-1. If so, the value of the data item written in the FROM or USING phrase of the PICTURE clause is copied into the screen item. Then, the data is displayed. This is done according to the copying rules.

   c. If a group screen item is specified as data-name-1, all the screen items belonging to the screen item are displayed.

2. The LINE NUMBER phrase specifies the screen line number corresponding to the screen item for data-name-1. In doing this, the first line of the physical screen is assumed to be 1.

3. The COLUMN NUMBER phrase specifies the screen column number corresponding to the screen item for data-name-1. In doing this, the first column of the physical screen is assumed to be 1.

4. If the AT phrase is omitted, data is displayed starting at the first line and column of the screen.

DISPLAY Statement (Operation on Command Line Arguments and Environmental Variables)

The DISPLAY statement positions a command line argument and an environmental variable, then stores a value in the environmental variable.

[Format]

$$\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{integer-1} \end{array} \right\} \underline{\text{UPON}} \text{ mnemonic-name-1}$$

$$\begin{array}{l} \text{[ON } \underline{\text{EXCEPTION}} \text{ imperative-statement-1]} \\ \text{[}\underline{\text{NOT}} \text{ ON } \underline{\text{EXCEPTION}} \text{ imperative-statement-2]} \\ \text{[}\underline{\text{END-DISPLAY}}\text{]} \end{array}$$

Syntax Rules

1. mnemonic-name-1 must be associated with one of the following function-names in the special-names paragraph of the environment division:

   a. ARGUMENT-NUMBER

   b. ENVIRONMENT-NAME

   c. ENVIRONMENT-VALUE

2. identifier-1 and literal-1 must satisfy the following conditions:

   a. If the mnemonic-name corresponding to the function-name ARGUMENT-NUMBER is specified as mnemonic-name-1, identifier-1 must be an unsigned integer item. Also, literal-1 must be an unsigned integer.

b. The mnemonic-name corresponding to the function-name ENVIRONMENT-NAME or ENVIRONMENT-VALUE may be specified as mnemonic-name-1. If so, identifier-1 must be a group item not containing a variable occurrence data item, or an alphanumeric data item. Also, literal-1 must be a nonnumeric literal.

3. The ON EXCEPTION and NOT ON EXCEPTION phrases can be written only if the mnemonic-name corresponding to the function-name ENVIRONMENT-VALUE is specified as mnemonic-name-1.

General Rules

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ARGUMENT-NUMBER

Specify the position of a command line argument as identifier-1 or literal-1. The values of identifier-1 and literal-1 must be in the range of 0 to 99. Execution of the DISPLAY statement positions the argument position indicator as specified as identifier-1 or literal-1.

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ENVIRONMENT-NAME

Specify the name of an environmental variable as identifier-1 or literal-1. Execution of the DISPLAY statement enables an input-output operation on the environmental variable having the name specified as identifier-1 or literal-1.

Rules Applied if Mnemonic-name-1 is Associated with the Function-name ENVIRONMENT-VALUE

1. Specify the value to be stored in an environmental variable as identifier-1 or literal-1.
2. Before execution of this DISPLAY statement, the DISPLAY statement where the mnemonic-name corresponding to the function-name ENVIRONMENT-NAME is specified must be executed. Execution of the DISPLAY statement stores the value specified as identifier-1 or literal-1 in the environmental variable positioned by the preceding DISPLAY statement.
3. If the ON EXCEPTION phrase is written, imperative-statement-1 is executed under either of the following conditions:
    a. Before execution of the ACCEPT statement, the DISPLAY statement for positioning the environmental variable is not executed.
    b. A sufficient field for storing a value in the environmental variable cannot be acquired.
4. If the NOT ON EXCEPTION phrase is written, and the exception condition in (2) does not occur, imperative-statement-2 is executed.

DIVIDE Statement (Nucleus)

The DIVIDE statement performs division and obtains the quotient and the remainder.

[Format 1] To obtain the quotient, and replace the dividend with it:

$$\begin{array}{l}
\underline{\text{DIVIDE}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \underline{\text{INTO}}\ \{\text{identifier-2}\ [\underline{\text{ROUNDED}}]\}\ \ldots\\
\quad [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-1}]\\
\quad [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\
\quad [\underline{\text{END-DIVIDE}}]
\end{array}$$

[Format 2] To obtain the quotient, and store it in a data item other than the one used for the dividend (1/2):

$$\begin{array}{l}
\underline{\text{DIVIDE}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \underline{\text{INTO}} \begin{Bmatrix}\text{identifier-2}\\ \text{literal-2}\end{Bmatrix} \underline{\text{GIVING}}\\
\qquad\qquad\qquad\qquad \{\text{identifier-3}\ [\underline{\text{ROUNDED}}]\}\ \ldots\\
\quad [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-1}]\\
\quad [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\
\quad [\underline{\text{END-DIVIDE}}]
\end{array}$$

[Format 3] To obtain the quotient, and store it in a data item other than the one used for the dividend (2/2):

$$\begin{array}{l}
\underline{\text{DIVIDE}} \begin{Bmatrix}\text{identifier-2}\\ \text{literal-2}\end{Bmatrix} \underline{\text{BY}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \underline{\text{GIVING}}\\
\qquad\qquad\qquad\qquad \{\text{identifier-3}\ [\underline{\text{ROUNDED}}]\}\ \ldots\\
\quad [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-1}]\\
\quad [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\
\quad [\underline{\text{END-DIVIDE}}]
\end{array}$$

[Format 4] To obtain the quotient and the remainder (1/2):

$$\begin{array}{l}\underline{\text{DIVIDE}}\left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\}\ \underline{\text{INTO}}\ \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\}\ \underline{\text{GIVING}}\ \ \text{identifier-3}\\ \quad[\underline{\text{ROUNDED}}]\ \ \underline{\text{REMAINDER}}\ \ \text{identifier-4}\\ \quad[\text{ON}\ \ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \ \text{imperative-statement-1}]\\ \quad[\underline{\text{NOT}}\ \ \text{ON}\ \ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\ \quad[\underline{\text{END-DIVIDE}}]\end{array}$$

[Format 5] To obtain the quotient and the remainder (2/2):

$$\begin{array}{l}\underline{\text{DIVIDE}}\ \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\}\ \underline{\text{BY}}\left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\}\ \underline{\text{GIVING}}\ \ \text{identifier-3}\\ \quad[\underline{\text{ROUNDED}}]\ \ \underline{\text{REMAINDER}}\ \ \text{identifier-4}\\ \quad[\text{ON}\ \ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \ \text{imperative-statement-1}]\\ \quad[\underline{\text{NOT}}\ \ \text{ON}\ \ \underline{\text{SIZE}}\ \ \underline{\text{ERROR}}\ \text{imperative-statement-2}]\\ \quad[\underline{\text{END-DIVIDE}}]\end{array}$$

Syntax Rules

1. identifier-1 and identifier-2 must be numeric data items.
2. identifier-3 and identifier-4 must be numeric data items or numeric edited data items.
3. literal-1 and literal-2 must be numeric literals.

General Rules

Rule for Format 1

The DIVIDE statement for format 1 divides identifier-2 by the operand before INTO, then stores the quotient in identifier-2. This division is done in the written sequence of identifier-2.

Rule for Format 2

The DIVIDE statement for format 2 divides the operand after INTO by the operand before INTO, then stores the quotient in identifier-3. The quotient is stored in the written sequence of identifier-3.

Rule for Format 3

The DIVIDE statement for format 3 divides the operand before BY with the operand after BY, then stores the quotient in identifier-3. The quotient is stored in the written sequence of identifier-3.

Rule for Format 4

The DIVIDE statement for format 4 divides the operand after INTO by the operand before INTO. Then, it stores the quotient in identifier-3, and the remainder in identifier-4.

Rule for Format 5

The DIVIDE statement for format 5 divides the operand before BY with the operand after BY. Then, it stores the quotient in identifier-3, and the remainder in identifier-4.

Rules Common to Format 4 and Format 5

1. The subscript in identifier-4, if any, is evaluated immediately before the remainder is stored in identifier-4.

2. The remainder can be obtained by the following expression:

   Remainder = dividend - quotient x divisor

   The dividend is the value of identifier-2 or literal-2. The divisor is the value of identifier-1 or literal-1. One of the following values is used as the quotient to compute the remainder:

   a. If a numeric edited data item is specified as identifier-3, the value is not edited in the format specified in the PICTURE clause.

   b. If the ROUNDED phrase is written, the value is truncated (before the quotient is rounded off). The following are the same as for identifier-3:

      The number of the digits of the quotient for computing the remainder

      Decimal position

      Whether a sign is included

   c. For other than the above, the value of identifier-3 is used.

3. The precision for identifier-4 observes the rules in (2). A value aligned by a decimal point and truncated is stored in identifier-4 as required.

4. If the ON SIZE ERROR phrase is written, the execution result is as follows:

   a. If the quotient overflows, the contents of identifier-3 and identifier-4 are not changed.

   b. If the remainder overflows, the content of identifier-4 is not changed. The user must check whether the quotient or the remainder overflows.

Rules Common to Format 1 to Format 5

1. The END-DIVIDE phrase delimits the scope of the DIVIDE statement.

2. For details of the ROUNDED and ON SIZE ERROR phrases, and the operation and copying rules, see the section titled "Common Statement Rules."

ENTRY Statement (Inter-program Communication)

The ENTRY statement specifies the secondary entry point for the called program.

[Format]

$$
\begin{array}{l}
\underline{\text{ENTRY}}\ \text{literal-1} \\
\text{[}\underline{\text{WITH}}\ \left\{\begin{array}{l}\underline{\text{C}}\\ \underline{\text{PASCAL}}\\ \underline{\text{STDCALL}}\end{array}\right\}\ \underline{\text{LINKAGE}}\text{]} \\
\quad\text{[}\underline{\text{USING}}\ \{\text{data-name-1}\}\ \ldots\ \text{]}
\end{array}
$$

Win The WITH phrase can be specified.

Syntax Rules

1. literal-1 must be a nonnumeric literal. literal-1 must observe the rules for describing program names. For details of the rules, see the section titled “Program-ID paragraph.”
2. literal-1 must not be the same as other entry names and program names in the run unit.
3. For details of the rules for the USING phrase, see the section titled “Program-ID paragraph.”
4. For details of the maximum number of data-name-1 that can be written in the USING phrase, see Appendix B, “Quantitative System Restrictions.”

Rules for the WITH Phrase

The WITH phrase specifies calling conventions. If the WITH phrase is omitted, the COBOL linkage conventions are applied.

General Rules

1. The ENTRY statement specifies the secondary entry point. Specify the name of the secondary entry point as literal-1. The name of the secondary entry point may be specified as a literal or an identifier in the CALL statement. If so, control is transferred to the secondary entry point when the CALL statement is executed. To receive parameters from the calling program, write the USING phrase.
2. Control may be transferred to the ENTRY statement by executing a statement other than the CALL statement. If so, control is transferred to the statement following the ENTRY statement without doing anything.
3. The ENTRY statement must not be written in an internal program.

Rules for the WITH Phrase

In a single separately compiled program, the Procedure Division header of the outermost program must be the same as the WITH phrase in the ENTRY statement.

EVALUATE Statement (Nucleus)

The EVALUATE statement evaluates multiple conditions, and executes the statements corresponding to the evaluation result.

[Format]

```
         ⎧ Identifier-1 ⎫ ⎡      ⎧ Identifier-2 ⎫ ⎤
         ⎪ literal-1    ⎪ ⎢      ⎪ literal-2    ⎪ ⎥
EVALUATE ⎨ expression-1 ⎬ ⎢ ALSO ⎨ expression-2 ⎬ ⎥ ...
         ⎪ TRUE         ⎪ ⎢      ⎪ TRUE         ⎪ ⎥
         ⎩ FALSE        ⎭ ⎣      ⎩ FALSE        ⎭ ⎦

{{WHEN
 ⎧ ANY                                                                   ⎫
 ⎪ condition-1                                                           ⎪
 ⎪ TRUE                                                                  ⎪
 ⎪ FALSE                                                                 ⎪
 ⎪       ⎧⎧ Identifier-3            ⎫                                    ⎪
 ⎨ [NOT] ⎨⎨ literal-3               ⎬                                    ⎬
 ⎪       ⎩⎩ arithmetic-expression-1 ⎭                                    ⎪
 ⎪                ⎡⎧ THRU    ⎫ ⎧ Identifier-4            ⎫⎤⎫             ⎪
 ⎪                ⎢⎨         ⎬ ⎨ literal-4               ⎬⎥⎬             ⎪
 ⎩                ⎣⎩ THROUGH ⎭ ⎩ arithmetic-expression-2 ⎭⎦⎭             ⎭
[ALSO
 ⎧ ANY                                                                   ⎫
 ⎪ condition-2                                                           ⎪
 ⎪ TRUE                                                                  ⎪
 ⎪ FALSE                                                                 ⎪
 ⎪       ⎧⎧ Identifier-5            ⎫                                    ⎪
 ⎨ [NOT] ⎨⎨ literal-5               ⎬                                    ⎬ ] ...} ...
 ⎪       ⎩⎩ arithmetic-expression-3 ⎭                                    ⎪
 ⎪                ⎡⎧ THRU    ⎫ ⎧ Identifier-6            ⎫⎤⎫             ⎪
 ⎪                ⎢⎨         ⎬ ⎨ literal-6               ⎬⎥⎬             ⎪
 ⎩                ⎣⎩ THROUGH ⎭ ⎩ arithmetic-expression-4 ⎭⎦⎭             ⎭
  imperative-expression-1} ...
     [WHEN OTHER imperative-expression-2]
   [END-EVALUATE]
```

Syntax Rules

1. The operands (including TRUE and FALSE) before the first WHEN phrase in the EVALUATE statement are called selection subjects.
2. The operands (including TRUE, FALSE, and ANY) in the WHEN phrase in the EVALUATE statement are called selection objects. If the THROUGH phrase is written in the WHEN phrase, two operands combined by the THROUGH phrase are considered to be one selection object.
3. THRU is synonymous with THROUGH.
4. The two operands combined by the THROUGH phrase must be identifiers, literals, or arithmetic expressions in the same class.
5. The number of the selection objects in one WHEN phrase must equal to that of the selection subjects.
6. The selection objects in one WHEN phrase must be associated with the selection subjects. For example, the first object must be associated with the first subject. The second object must be associated with the second subject. The rules for associating selection objects with selection subjects are as follows:
   a. When an identifier, literal, or arithmetic expression is written as a selection object, an operand that can be compared with the object must be written as the corresponding selection subject.
   b. When condition-1, and condition-2, TRUE, or FALSE is written as a selection object, a conditional expression, TRUE, or FALSE must be written as the corresponding selection subject.

   c. If ANY is written as a selection object, any corresponding selection subject can be written.

7. Boolean items or literal cannot be written as the two operands combined by the THROUGH phrase.

8. Boolean expressions cannot be written as expression-1 and expression-2.

General Rules

1. The EVALUATE statement evaluates the values of the individual selection subjects and those of the selection objects. Then, it selects the WHEN phrase that satisfies the condition, and executes the imperative statement for that WHEN phrase.

2. The following values are assigned to the individual selection subjects and the selection objects:

   a. If identifier-1 or identifier-2 is written as a selection subject, the value and class of the identifier are assigned to the subject. Similarly, identifier-3 or identifier-5 without the NOT and THROUGH phrases may be written as a selection object. If so, the value and class of the identifier are assigned to the object.

   b. If literal-1 or literal-2 is written as a selection subject, the value and class of the literal are assigned to the subject. Similarly, literal-3 or literal-5 without the NOT and THROUGH phrases may be written as a selection object. If so, the value and class of the literal are assigned to the object. However, the figurative constant ZERO may be specified as literal-3 or literal-5. If so, the class of a selection subject is assigned to that of the corresponding selection object.

c. An arithmetic expression may be written as expression-1 or expression-2 for a selection subject. If so, the numeric value obtained according to the rules for evaluating an arithmetic expression is assigned to the subject. Similarly, arithmetic-expression-1 or arithmetic-expression-3 without the NOT and THROUGH phrases may be written as a selection object. If so, the numeric value obtained according to the rules for evaluating an arithmetic expression is assigned to the object.

d. A conditional expression may be written as expression-1 or expression-2 for a selection subject. If so, the truth value obtained according to the rules for evaluating a conditional expression is assigned to the subject. Similarly, condition-1 or condition-2 may be written as a selection object. If so, the truth value obtained according to the rules for evaluating a conditional expression is assigned to the object.

e. If TRUE or FALSE is written as a selection subject, the truth value is assigned to the subject. If TRUE is written, true is assigned. If FALSE is written, false is assigned. Similarly, if TRUE or FALSE is written as a selection object, the truth value is assigned to the object.

f. If ANY is written as a selection object, the object is not evaluated.

g. The THROUGH phrase may be written, and the NOT phrase may be omitted as a selection object. If so, of all the values of the corresponding selection subject, those that satisfy the following conditions are assigned to the selection object:

    - Greater than or equal to the operand immediately before the THROUGH phrase
    - Less than or equal to the operand immediately after the THROUGH phrase

  h. The NOT phrase may be written as a selection object, and the THROUGH phrase may be omitted. The value not equal to the value assigned when the NOT phrase is not written is assigned to the object. Similarly, the NOT and THROUGH phrases may be written as a selection object. If so, the values not included within the scope of the values assigned when the NOT phrase is not written are assigned to the object. The values assigned when the NOT phrase is not written are those explained in items (a) to (h) above.

3. The value of each selection subject and that of the corresponding selection object assigned according to the rules in (2) are compared. Then, the WHEN phrase that satisfies the condition is determined. This comparison is done as follows:

   a. The first selection object in one WHEN phrase is compared with the first selection subject, the second selection object, the second selection subject, and so on. The comparison rules are as follows:

   - The values of selection objects and those of selection subjects may be assigned according to the rules in (a), (b), (c), (g), or (h) of (2). If so, the value of each selection object is compared with that of the corresponding selection subject. If the value assigned to the selection object is the same as that assigned to the corresponding selection subject, the comparison result is true.

- The value of a selection object may be assigned according to the rules in (d) or (e) of (2). If so, the truth value of the object and that of the corresponding selection subject are compared. If the truth values are the same, the comparison result is true.
- If ANY is written as a selection object, the comparison result is always true regardless of the value of the corresponding selection subject.

b. In one WHEN phrase, the results of comparing all selection objects with the corresponding selection subjects in the method (a) may be true. If so, this WHEN phrase is selected as the WHEN phrase that satisfies the condition.

c. The above steps (a) and (b) are repeated in the order in which the WHEN phrases are written. This is done until the WHEN phrase that satisfies the condition is selected or all the WHEN phrases are processed.

4. After the comparison in (3) is completed, execution of the EVALUATE statement continues as follows:

    a. If the WHEN phrase that satisfies the condition is selected, the first imperative-statement-1 following the selected WHEN phrase is executed. After execution of imperative-statement-1, execution of the EVALUATE statement terminates.

    b. If no WHEN phrase is selected, and the WHEN OTHER phrase is written, imperative-statement-2 is executed. After execution of imperative-statement-2, execution of the EVALUATE statement terminates.

    c. If no WHEN phrase is selected, and the WHEN OTHER phrase is omitted, execution of the EVALUATE statement terminates.

5. The END-EVALUATE phrase delimits the scope of the EVALUATE statement.

6. For details of conditional and arithmetic expressions, and the comparison rules, see the section titled “Common Statement Rules”.

Example of the EVALUATE Statement (1/2)

```
[Data division]
 01  MAIL-RECORD.
       02 KIND           PIC X.
       02 WEIGHT         PIC 9(4).
       02 CHARGE         PIC 9(5).
       02 FLAG           PIC X(5)
[Procedure division]
      EVALUATE KIND ALSO WEIGHT
         WHEN “1” ALSO 0 THRU 25
              MOVE 60 TO CHARGE
         WHEN “1” ALSO 26 THRU 50
              MOVE 70 TO CHARGE
         WHEN “2” ALSO ANY
              MOVE 40 TO CHARGE
         WHEN OTHER
              MOVE “ERROR” TO FLAG
     END-EVALUATE
```

When the data division and the EVALUATE statement may be written as shown above, the EVALUATE statement is executed based on the values of KIND and WEIGHT as shown in the following table:

Condition		**Statement to be Executed**
Value of KIND	**Value of WEIGHT**	
“1”	0 TO 25	MOVE 60 TO CHARGE
“1”	26 TO 50	MOVE 70 TO CHARGE
“2”	Any value	MOVE 40 TO CHARGE
Other than the above		MOVE “ERROR” TO FLAG

Example of the EVALUATE Statement (2/2)

```
01   EMPLOYMENT-RECORD.
       02  SEX              PIC X.
       02 AGE               PIC 9(2).
       02 MARKS             PIC 9(3).
       02 RESULT            PIC X(4).
[Procedure division]
      EVALUATE  SEX ALSO AGE TRUE
         WHEN “F” ALSO 18 THRU 22 ALSO
              MARKS > 74
         WHEN “F” ALSO 23 THRU 29 ALSO
              MARKS > 84
         WHEN “M” ALSO 18 THRU 39 ALSO
              MARKS > 79
              MOVE “PASS” TO RESULT
              WHEN OTHER MOVE “FAIL ” TO RESUT
     END-EVALUATE
```

The data division and the EVALUATE statement may be written as shown above. If so, the EVALUATE statement is executed based on the values of SEX, AGE, and MARKS as shown in the following table:

Condition			Statement to be executed
Value of SEX	Value of AGE	Value of MARKS	
“F”	18 TO 22	> 74	MOVE “PASS” TO RESULT
“F”	23 TO 29	> 84	
“M”	18 TO 39	> 79	
Other than the above			MOVE “FAIL” TO RESULT

EXIT Statement (Nucleus)

The EXIT statement specifies the exit common to a series of procedures:

[Format]

<u>EXIT</u>

Syntax Rules

One EXIT statement must be one sentence. The paragraph containing the EXIT statement must not contain any other elements.

General Rules

Write the EXIT statement only to assign a procedure-name. The EXIT statement does not affect program compilation and execution. The name of the paragraph containing the EXIT statement can be written in the THROUGH phrase of the PERFORM statement.

EXIT PERFORM Statement (Nucleus)

Specifies the exit of an in-line PERFORM statement.

[Format]

EXIT [TO TEST OF] PERFORM

General Rules

1. The EXIT PERFORM statement specifies the exit of an in-line PERFORM statement. The EXIT PERFORM statement is allowed only within an in-line PERFORM statement.
2. The EXIT PERFORM statement is associated with the innermost PERFORM statement in which it is included.
3. Executing an EXIT PERFORM statement without the TEST phrase specified passes control to the end of the corresponding PERFORM statement.
4. An EXIT PERFORM statement without the TEST phrase specified is allowed only in a PERFORM statement with a termination condition specified (one written in format 2, 3, or 4). Executing an EXIT PERFORM statement with the TEST phrase specified passes control to the inspection process of the corresponding in-line PERFORM statement. The inspection process is either of the following:
    a. A process in a PERFORM statement written in format 2 that inspects the occurrence count specified before TIMES, or

    b. A process in a PERFORM statement written in format 3 or 4 that inspects the condition specified after UNTIL.

EXIT PROGRAM Statement (Inter-program Communication)

Specifies the logical end of a called program.

[Format]

EXIT PROGRAM

Syntax Rules

1. When an EXIT PROGRAM statement is written in a list of imperative statements in a sentence, it must appear at the end of that list.
2. The EXIT PROGRAM is not allowed in the declarative procedure of a USE statement with the GLOBAL phrase specified.

General Rules

1. The EXIT PROGRAM statement specifies the logical end of a called program. Executing an EXIT PROGRAM statement passes control to right after the point of call to the program in which the EXIT PROGRAM statement appears. If a program has been called by executing a CALL statement, executing an EXIT PROGRAM statement would pass control to right after the CALL statement.

2. After the EXIT PROGRAM statement is executed, the called program is in the same status as that in effect upon execution of the CALL statement, except for the data items and file contents that are shared by the calling and called programs.
3. After the execution of the EXIT PROGRAM statement, the status of the called program (the program that has executed the EXIT PROGRAM statement) depends on whether it has an initialization attribute:
    a. If it does not have an initialization attribute, it remains in the same status in effect before it had been called, except that the scope of the PERFORM statement has been determined.
    b. If it has an initialization attribute, it enters the same status as it would upon execution of a CANCEL statement.

GENERATE Statement (Report writing)

Writes a report according to a report description entry.

[Format]

$$\underline{\text{GENERATE}} \left\{ \begin{array}{l} \text{data-name-1} \\ \text{report-name-1} \end{array} \right\}$$

Syntax Rules

1. data-name-1 must name a detail report group. Data-name-1 may be qualified by a report name.
2. report-name-1 can be written only if the report description entry meets all of the following requirements:
    a. The report description entry contains a CONTROL clause.
    b. The report description entry does not contain more than one detail report group.
    c. The report description entry contains at least one body group entry.

General Rules

1. The report writer control system performs summary report processing on GENERATE statements with report-name-1 specified. If all the GENERATE statements for a report are written in the format of a GENERATE statement with report-name-1 specified, the report that is generated and presented subsequently is called a summary report. A summary report is a report in which a detail report group is not presented.
2. The report writer control system performs detail processing on GENERATE statements with data-name-1 specified, including processing specific to the detail report group specified. Normally, executing GENERATE statements with data-name-1 specified causes the report writer control system to present the specified detail report group.

3. The report writer control system saves the value of the control data item when it executes the first GENERATE statement for the report. The system uses this value of the control data item to determine a control break until it detects one during execution of the second or subsequent GENERATE statement for the same report. When the system detects a control break, it saves the new value of the control data item, using that value thereafter to detect a control break until another control break is encountered.

4. When a need arises to perform a form feed to present the body group during report presentation, the report writer control system automatically executes processing as specified by a page heading report group and a page footing group if they are predefined.

5. The report writer control system executes processing as specified by the page heading report group and page footing group as explained in (4), processing the following report groups in the order of their appearance in the report description entry when it executes the first GENERATE statement for the report:

   a. Report heading report group

   b. Page heading report group

   c. Control heading groups at all levels, from high to low

   d. If a GENERATE statement with data-name-1 specified is executed, the detail report group specified; if a GENERATE statement with report-name-1 specified is executed, part of the detail report processing

6. The report writer control system executes processing as specified by the page heading report group and page footing group as explained in (4), performing the following operations when it executes a second or subsequent GENERATE statement for the report:

    a. Detection of a control break. The rules of relation conditions apply to determining the equality of control data items. When a control break is detected, the report writer control system:

        - Makes the value of the control data item used to detect the control break accessible for reference to the USE procedure associated with the control footing group and the SOURCE clause in the control footing group.

        - Processes the control footing groups in the ascending order of levels. All the control footing groups at any level lower than the level in effect when the control break occurred are processed.

        - Processes the control heading groups in the descending order of levels. All the control heading groups at any level lower than the level in effect when the control break occurred are processed.

    b. If a GENERATE statement with data-name-1 specified is executed, the detail report group specified; if a GENERATE statement with report-name-1 specified is executed, part of the detail report processing.

7. GENERATE statements for a report can be executed only after the execution of an INITIATE statement and before the execution of a TERMINATE statement.

8. For the report presentation rules, see the section titled “Type clause,” and the section titled “Report Group Presentation Rules.”

GO TO Statement (Nucleus)

Passes control from one point in a procedure division to another. A GO TO statement in format 1 with procedure-name-1 omitted is an obsolete element.

[Format 1] Passes control to a specific point.

GO TO [procedure-name-1]

[Format 2]Passes control according to the value of the data item.

GO TO {procedure-name-1} ... DEPENDING ON identifier-1

Syntax Rules

Rules for Format 1

1. When the destination of the GO TO statement is altered with an ALTER statement, the paragraph specified in the ALTER statement must be composed of a single sentence consisting of a GO TO statement written in format 1.
2. A GO TO statement without procedure-name-1 specified must be a single sentence. Only this statement is allowed in the paragraph in which it appears.
3. When a GO TO statement is written in a statement consisting of a number of imperative statements, it must be the last imperative statement.

Rule for Format 2

identifier-1 must be an integer item.

General Rules

Rules for Format 1

1. A GO TO statement written in format 1 passes control to procedure-name-1.
2. When procedure-name-1 is omitted, the executions of a GO TO statement must be preceded with the execution of an ALTER statement that references the GO TO statement.

Rule for Format 2

Specify the destinations to which control passes when the value of identifier-1 is 1, 2, 3, and so on in the procedure-name-1 list in sequence from left to right. A GO TO statement written in format 2 passes control the to first procedure-name-1 if the value of identifier-1 is 1, to the second procedure-name-1 if the value of identifier-1 is 2, and so on. The GO TO statement would simply pass control to the next statement without doing anything if the value of identifier-1 is none of 1, 2, 3 and so on.

IF Statement (Nucleus)

Evaluates a condition and executes the statement corresponding to the result of the evaluation.

[Format]

$$\underline{\text{IF}}\ \text{condition-1}$$

$$\text{THEN}\ \left\{\begin{array}{l}\{\text{statement-1}\}\ldots\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}$$

$$\left\{\begin{array}{l}\underline{\text{ELSE}}\ \ \{\text{statement-2}\}\ldots\ [\underline{\text{END-IF}}]\\ \underline{\text{ELSE}}\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\\ \underline{\text{END-IF}}\end{array}\right\}$$

Syntax Rules

1. The ELSE NEXT SENTENCE phrase can be omitted when it is immediately followed by a separator period.
2. The NEXT SENTENCE phrase and the END-IF phrase cannot be written at the same time.

General Rules

1. The IF statement evaluates condition-1 and executes the operation specified by the THEN phrase if it is true or the operation specified by the ELSE phrase if it is false.
2. The IF statement is terminated by either:
   a. An END-IF phrase appearing at the same level
   b. A separator period
   c. The ELSE phrase in the outermost IF statement if multiple IF statements are nested.

   For the scope of the IF statement, see the “Scope of a statement” part of the section titled “Procedure Division Configuration.”
3. If the result of evaluation of condition-1 is true, control will pass in the following ways:
   a. When statement-1 is written, the statement specified in the statement-1 list is executed. Control passes to the end of the IF statement when the execution of the statement specified in the statement-1 list is completed. If a statement is executed to pass control explicitly during statement-1, however, control would pass according to the rules of that statement.
   b. When NEXT SENTENCE is written, control passes to the execution sentence following the IF statement.

4. If the result of evaluation of condition-1 is false, control will pass to the ELSE phrase, with the THEN phrase being overridden. Then, control would pass according to the entry of the ELSE phrase in the following ways:

   a. When statement-2 is written, the statement specified in the statement-2 list is executed. Control passes to the end of the IF statement when the execution of the statement specified in the statement-2 list is completed. If a statement is executed to pass control explicitly during statement-2, however, control would pass according to the rules of that statement.

   b. When the ELSE phrase is omitted, control passes to the end of the IF statement.

   c. When NEXT SENTENCE is written, control passes to the execution sentence following the IF statement.

5. Where IF statements are nested, one IF statement nested in another is assumed to form a set of IF, ELSE, and END-IF from left to right. The IF corresponding to an ELSE and END-IF is the one, located to their left, that is the rightmost IF among the IF statements not associated with any other ELSE and END-IF.

6. The END-IF statement delimits the scope of an IF statement.

7. For more details on conditional expressions and comparison rules, see the section titled “Common Rules Concerning Statements.”

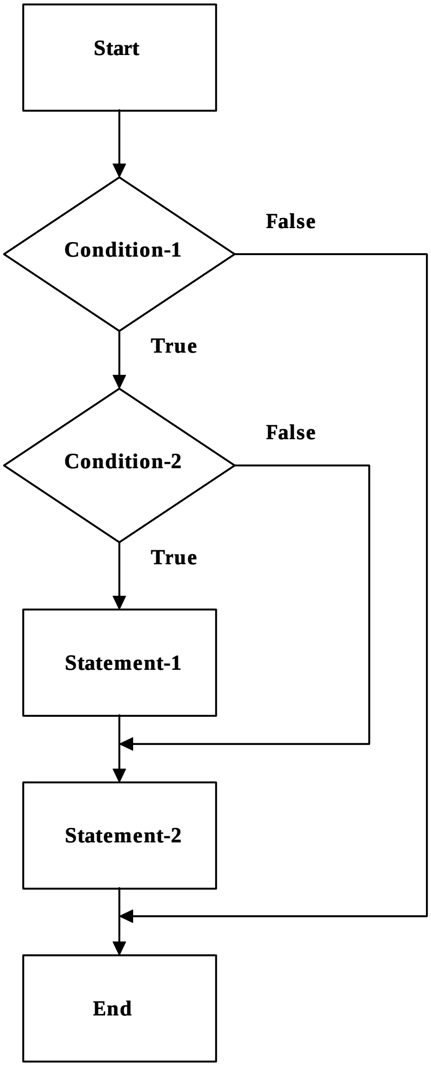
Example of Nested IF Statements

An example of nested IF statements is shown below.

[Example of nested IF statements]

```
IF condition-1 THEN
  IF condition-2 THEN
    statement-1
  END-IF
  statement-2
END-IF
```

[Logical flow of processing by nested IF statements]



INITIALIZE Statement (Nucleus)

Initializes a data item.

[Format]

```
INITIALIZE {identifier-1} ...
                   ⎧ ALPHABETIC              ⎫
                   ⎪ ALPHANUMERIC            ⎪
                   ⎪ NUMERIC                 ⎪
   [REPLACING{     ⎨ ALPHANUMERIC-EDITED     ⎬
                   ⎪ NUMERIC-EDITED          ⎪
                   ⎪ NATIONAL                ⎪
                   ⎪ NATIONAL-EDITED         ⎪
                   ⎩ BOOLEAN                 ⎭

                ⎧ identifire-2 ⎫
   DATA BY      ⎨              ⎬ } ...]
                ⎩ literal      ⎭
```

Syntax Rules

1. The combination of the category of identifier-2 or literal-1 and the category written in the REPLACING phrase must conform to the rules of move in which the former is a sending category and the latter is a receiving category.
2. A REPLACING phrase of the same category may not be duplicated in the REPLACING phrase list.
3. An OCCURS clause with the DEPENDING phrase may not be written in identifier-1 or in the data description entry of a data item subordinate to identifier-1.
4. An index data item may not be specified with identifier-1 and identifier-2.
5. A data item with the RENAME clause may not be specified in identifier-1.

General Rules

1. The INITIALIZE statement initializes the data item of identifier-1. Leave the REPLACING phrase unspecified to initialize a data item of any category with a predefined value. Write the REPLACING phrase to initialize an elementary item of a particular category with a particular value. Specify the category to be the object of initialization after REPLACING, and the initial value after BY.

2. The word that follows REPLACING specifies the category to be the object of initialization. ALPHABET represents an alphabetic data item, ALPHANUMERIC, an alphanumeric data item, NUMERIC, a numeric data item, ALPHANUMERIC-EDITED, an alphanumeric edited data item, NUMERIC-EDITED, a numeric data item, NATIONAL, a national data item, NATIONAL-EDITED, a national edited data item, and BOOLEAN, a Boolean data item.

3. If an index data item or point data item is specified in identifier-1, the execution of the INITIALIZE statement would not alter the contents of identifier-1.

4. If a group item is specified in identifier-1, among the elementary items subordinate to the group item, the following elementary items are not subject to initialization:

   a. An index data item, point data item, or FILLER, or

   b. A data item having the REDEFINES clause specified or any data item subordinate to such a data item.

5. When the REPLACING phrase is written, data item initialization takes place in the following ways:

   a. If an elementary item is specified in identifier-1, it is initialized with the value of the operand specified with BY only if that elementary item belongs to the category written in the REPLACE phrase, except in case (3) above.

   b. If a group item is specified in identifier-1, among the elementary items subordinate to the group item, only the elementary items that belong to the category written in the REPLACE phrase are initialized with the value of the operand specified with BY. If the group item includes a table, the elementary items that form table elements of that table are also subject to initialization, except in case (4) above.

   c. Initialization takes place the same way as a MOVE statement would be executed in which the operand specified with BY is a sending item and the elementary item that is subject to initialization is a receiving item.

6. When the REPLACING phrase is omitted, data item initialization takes place in the following ways:

   a. If an elementary item is specified in identifier-1, it is initialized with the value explained in (c) below.

   b. If a group item is specified in identifier-1, the elementary items subordinate to the group item are initialized with the value explained in (c) below. If the group item includes a table, the elementary items that form table elements of that table are also subject to initialization, except in case (4) above.

c. Initialization takes place the same way as a MOVE statement would be executed in which a literal having any of the values explained below is a sending item and the elementary item that is subject to initialization is a receiving item.

- If the elementary item that is subject to initialization has the category alphabetic, alphanumeric or alphanumeric edited, the value of the sending item is a alphanumeric space.
- If the elementary item that is subject to initialization has the category national, or national edited, the value of the sending item is a national space.
- If the elementary item that is subject to initialization has the category numeric or numeric edited, the value of the sending item is a numeric zero.
- If the elementary item that is subject to initialization has the category Boolean, the value of the sending item is a Boolean zero.

7. If a group item is specified in identifier-1, moves to the elementary items that are subject to initialization are initialized in the order of item within the group item.
8. If identifier-1 and identifier-2 occupy the same storage area, the validity of the result of execution of the INITIALIZATION statement would be unpredictable.

INITIATE Statement (Report writer)

Initiates a report writing process.

[Format]

INITIATE [report-name-1] ...

Syntax Rule

report-name-1 must be specified in the report description entry in the report section of the data division.

General Rules

1. The INITIATE statement executes the following initialization processes for the named report.
   a. Resetting the sum counter to 0
   b. Resetting the line-counter to 0.
   c. Setting the page counter to 1.
2. The INITIATE statement does not provide a function to keep any files relating to the named report open. To open such files, an OPEN statement with the OUTPUT or EXTEND phrase specified must be executed before the execution of the INITIATE statement.
3. Once an INITIATE statement is executed for report-name-1, it must be executed again until a corresponding TERMINATE statement is executed.

4. If two or more report names are written in a single INITIATE statement, its execution would produce the same result as the execution of separate INITIATE statements in which the report names are written in the order of their appearance in the single INITIATE statement would do.

INSPECT Statement

Counts the occurrence of a character string and replaces it.

[Format 1] Count the occurrences of a character string.

```
INSPECT identifier-1 TALLYING {identifier-2 FOR
 ⎧ CHARACTERS                                                   ⎫
 ⎪   [ ⎧BEFORE⎫ INITIAL ⎧identifier-4⎫] ...                      ⎪
 ⎪   [ ⎩AFTER ⎭         ⎩literal-2   ⎭]                          ⎪
 ⎨⎧ALL    ⎫ {⎧identifier-3⎫                                     ⎬ ...} ...
 ⎪⎩LEADING⎭ {⎩literal-1   ⎭                                     ⎪
 ⎪   [ ⎧BEFORE⎫ INITIAL ⎧identifier-4⎫] ...} ...                 ⎪
 ⎩   [ ⎩AFTER ⎭         ⎩literal-2   ⎭]                          ⎭
```

[Format 2] Replaces a character string.

```
INSPECT identifier-1 REPLACING
 ⎧ CHARACTERS BY ⎧identifier-5⎫                                      ⎫
 ⎪               ⎩literal-3   ⎭                                      ⎪
 ⎪   [ ⎧BEFORE⎫ INITIAL ⎧identifier-4⎫] ...                           ⎪
 ⎪   [ ⎩AFTER ⎭         ⎩literal-2   ⎭]                               ⎪
 ⎨ ⎧ALL    ⎫                                                         ⎬ ...
 ⎪ ⎨LEADING⎬ {⎧identifier-3⎫ BY ⎧identifier-5⎫                       ⎪
 ⎪ ⎩FIRST  ⎭ {⎩literal-1   ⎭    ⎩literal-3   ⎭                       ⎪
 ⎪   [ ⎧BEFORE⎫ INITIAL ⎧identifier-4⎫] ...} ...                      ⎪
 ⎩   [ ⎩AFTER ⎭         ⎩literal-2   ⎭]                               ⎭
```

[Format 3] Count the occurrences of a character string and replaces the string.

```
INSPECT identifier-1 TALLYING {identifier-2 FOR
⎧  CHARACTERS                                                      ⎫
⎪    ⎡⎧BEFORE⎫         ⎧identifier-4⎫⎤                             ⎪
⎪    ⎣⎩AFTER ⎭ INITIAL ⎩literal-2   ⎭⎦ ...                         ⎪
⎨⎧ALL    ⎫ ⎧⎧identifier-3⎫                                         ⎬ ...} ...
⎪⎩LEADING⎭ ⎩⎩literal-1   ⎭                                         ⎪
⎪    ⎡⎧BEFORE⎫         ⎧identifier-4⎫⎤                             ⎪
⎩    ⎣⎩AFTER ⎭ INITIAL ⎩literal-2   ⎭⎦ ...⎭ ...                    ⎭

   REPLACING
⎧                                                                      ⎫
⎪  CHARACTERS BY ⎧identifier-5⎫                                        ⎪
⎪                ⎩literal-3   ⎭                                        ⎪
⎪                                                                      ⎪
⎪    ⎡⎧BEFORE⎫         ⎧identifier-4⎫⎤                                 ⎪
⎪    ⎣⎩AFTER ⎭ INITIAL ⎩literal-2   ⎭⎦ ...                             ⎪
⎨ ⎧ALL    ⎫                                                            ⎬ ...
⎪ ⎨LEADING⎬ ⎧⎧identifier-3⎫ BY ⎧identifier-5⎫                          ⎪
⎪ ⎩FIRST  ⎭ ⎩⎩literal-1   ⎭    ⎩literal-3   ⎭                          ⎪
⎪    ⎡⎧BEFORE⎫         ⎧identifier-4⎫⎤                                 ⎪
⎩    ⎣⎩AFTER ⎭ INITIAL ⎩literal-2   ⎭⎦ ...⎭ ...                        ⎭
```

[Format 4] Specifies replacing character strings in a group.

```
INSPECT identifier-1 CONVERTING

     ⎧identifier-6⎫ TO ⎧identifier-7⎫
     ⎩literal-4   ⎭    ⎩literal-5   ⎭

     ⎡⎧BEFORE⎫         ⎧identifier-4⎫⎤
     ⎣⎩AFTER ⎭ INITIAL ⎩literal-2   ⎭⎦ ...
```

Syntax Rules

Rules Common to Format 1 to Format 4

1. identifier-1 must be a group item or an elementary item whose use is presentation.

2. identifier-2 must be a numeric data item.

3. identifier-3 to identifier-7 must each be a group item or an elementary item whose use is presentation.

4. literal-1 to literal-5 must each be a nonnumeric literal or national literal, or a figurative constant that does not begin with ALL. If a figurative constant is specified in literal-1, literal-2, or literal-4, its length is assumed one character.

5. Only one BEFORE phrase and only one AFTER phrase may follow each ALL, LEADING, FIRST, CHARACTERS, OR CONVERTING phrase.

6. Combinations of identifier-3 to identifier-7 and literal-1 to literal-5 must conform to the following rules:

   a. If identifier-1 is a national data item or national edited item, identifier-3 to identifier-7 must each be a national data item or national edited item. literal-1 to literal-5 must each be a national literal.

   b. If identifier-1 is other than a national data item or national edited item, identifier-3 to identifier-7 must each be other than a national data item or national edited item. literal-1 to literal-5 must each be other than a national literal.

Rules Common to Format 2 and Format 3

1. literal-3 and identifier-5 in the CHARACTERS phrase must be one character long.
2. literal-3 and identifier-5 in the ALL, LEADING, or FIRST phrase must be equal in their character length to literal-1 or identifier-3. If a figurative constant is specified in literal-3, it is assumed equal in its character length to literal-1 or identifier-3.

Rules for Format 4

1. literal-5 and identifier-7 must be equal in their character length to literal-4 or identifier-6. If a figurative constant is specified in literal-5, it is assumed equal in its character length to literal-4 or identifier-6.
2. The same character may not appear more than once in the character string of literal-4 or identifier-6.
3. For the maximum permissible character lengths of identifier-6 and identifier-7, see Appendix B, “System Limitations.”

General Rules

Rules for Format 1

1. The INSPECT statement in format 1 checks to see if the character string (inspected character string) specified in the FOR phrase is included in identifier-1 and counts its occurrences. The scope of character string inspection in identifier-1 is specified in the BEFORE or AFTER phrase. The CHARACTERS, ALL, or LEADING phrase specifies the way the occurrences of the inspected character string are tallied. The inspected character string is specified with identifier-3 or literal-1. If the CHARACTERS phrase is written, all the occurrences of the character string falling in the scope of character string inspection in identifier-1 are tallied.

2. If two or more inspected character strings are specified in the FOR phrase, they are inspected in the order of their entry.

3. The character string specified in identifier-1 is inspected once from left to right in sequence.

4. If two or more inspected character strings appear in a single ALL or LEADING phrase, ALL or LEADING applies to each inspected character string.

5. An initial value must be assigned to identifier-2 prior to execution of the INSPECT statement.

6. If identifier-1, identifier-3 or identifier-4 occupies the same storage area as identifier-2, the validity of the result of execution of the INSPECT statement would be unpredictable.

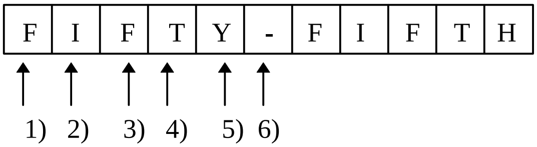
Flow of Processing of an INSPECT Statement Written in Format 1

The flow of processing is illustrated below reference to the following INSPECT statement:

[Example of an INSPECT statement in format 1]

```
INSPECT  AN-1  TALLYING  ED-2  FOR  ALL  “F”
                     BEFORE  INTIAL “-“.
```

[Contents of AN-1 before execution]



a. The inspected character string is “F.”

b. The character string of AN-1 is compared with the inspected character string from the leftmost character, one character at a time. 1) to 6) denote the leftmost character position at each instant of comparison. The comparison terminates at 6) where the scope of the character string inspection is exceeded.

c. The value of ED-2 is incremented by 1 during the comparison of the characters at 1) and 3) as they match the inspected character string. If the value of ED-2 before the execution of the INSPECT statement is 0, it is incremented to 2 after the execution of the INSPECT statement.

Rules for Format 2

1. The INSPECT statement in format 2 checks to see if the character string (inspected character string) specified in the REPLACING phrase is included in identifier-1 and replaces each occurrence of the character string matching the inspected character string with identifier-5 or literal-3. The scope of character string inspection in identifier-1 is specified in the BEFORE or AFTER phrase. The CHARACTERS, ALL, LEADING, or FIRST phrase specifies the way the occurrences of the inspected character string are replaced. The inspected character string is specified with identifier-3 or literal-1. If the CHARACTERS phrase is written, all occurrences of the character string falling in the scope of character string inspection in identifier-1 are replaced.

2. If two or more inspected character strings are specified in the REPLACING phrase, they are inspected in the order of their entry.

3. The character string specified in identifier-1 is inspected once from left to right in sequence.

4. If two or more inspected character strings appear in a single ALL, LEADING, or FIRST phrase, ALL, LEADING, or FIRST applies to each inspected character string.

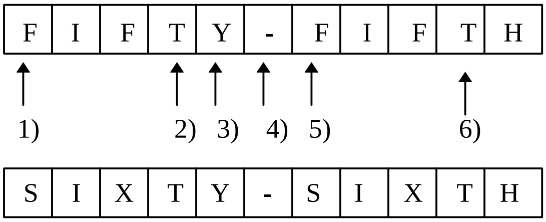
5. If identifier-3, identifier-4, or identifier-5 occupies the same storage area as identifier-1, the validity of the result of execution of the INSPECT statement would be unpredictable.

Flow of Processing of an INSPECT Statement Written in Format 2

The flow of processing is illustrated below reference to the following INSPECT statement:

[Example of an INSPECT statement in format 2]

```
INSPECT  AN-1  REPLACING  ALL  “FIF”  BY  “SIX”.
```



[Contents of AN-1 before execution]

F	I	F	T	Y	-	F	I	F	T	H

[Contents of AN-1 after execution]

S	I	X	T	Y	-	S	I	X	T	H

a. The inspected character string is “FIF.”

b. The character string of AN-1 is compared with the inspected character string from the leftmost character, three characters at a time. 1) to 6) denote the leftmost character position at each instant of comparison.

c. “FIF” is replaced with “SIX” during the comparison of the character strings starting at 1) and 5) as they match the inspected character string.

Rules for Format 3

1. The INSPECT statement in format 3 is equivalent to an INSPECT statement (format 1) with only the TALLYING phrase specified, followed by an INSPECT statement (format 2) with only the REPLACING phrase specified. For example, the INSPECT statement in (a) and that in (b) would yield the same result when executed.

```
(a) INSPECT AN-1 TALLYING ED-2  FOR  ALL   “F”
                          BEFORE INTIAL “-”.
                          REPLACING ALL “FIF” BY  “SIX”.
(b) INSPECT AN-1  TALLYING ED-2  FOR ALL  “F”
                          BEFORE INTIAL “-”.
   INSPECT  AN-1 REPLACING ALL “FIF” BY “SIX”.
```

2. The occurrence calculation process is governed by the format 1 rules, and the replacement process is governed by the format 2 rules.

Rules for Format 4

1. An INSPECT statement in format 4 is equivalent to an INSPECT statement (format 2) with the REPLACING phrase followed by two ALL phrases and nothing else. The following correspondence exists between the operand specified in format 4 and that specified in format 2:

   Identifier-1 in format 4: Identifier-1 in format 2.

   One character at a time as specified in identifier-6 or literal-4 in format 4: Identifier-3 or literal-1 in format 2 with the ALL phrase specified.

   One character at a time as specified in identifier-7 or literal-5 in format 4: Identifier-5 or literal-3 in format 2 with the ALL phrase specified.

   Identifier-4 or literal-2 in format 4: Identifier-4 or literal-2 in format 2 with the ALL phrase specified.

   For example, the execution of the INSPECT statement in (a) and that of the INSPECT statement in (b) would yield the same result.

```
(a)  INSPECT ITEM CONVERTING
       "ABCD" TO "XYZX" AFTER QUOTE BEFORE "@".

(b)  INSPECT ITEM REPLACING
       ALL "A"   BY "X" AFTER QUOTE BEFORE "@"
       ALL "B"   BY "Y" AFTER QUOTE BEFORE "@"
       ALL "C"   BY "Y" AFTER QUOTE BEFORE "@"
       ALL "D"   BY "Y" AFTER QUOTE BEFORE "@".
```

1. If identifier-4, identifier-6 or identifier-7 occupies the same storage area as identifier-1, the validity of the result of execution of the INSPECT statement would be unpredictable.

Rules Common to Format 1 to Format 4

IDENTIFIER RULES

All identifiers, except for identifier-2, are assumed to have the contents described below.

1. If an alphanumeric data item, alphabetic data item, or National data item is specified as an identifier, its contents are assumed to be a character string.

2. If an alphanumeric edited data item, numeric edited data item, unsigned numeric data item or external Boolean data item is specified as an identifier, its contents are assumed to be a character string redefined by an alphanumeric data item.

3. If a national edited item is specified as an identifier, its contents are assumed to be a character string redefined by a National data item.

4. If a signed numeric edited data item is specified as an identifier, its contents are moved to an unsigned numeric data item of the same length and are assumed to be a character string. The sign of the identifier is saved until the INSPECT statement is completed.

RULES GOVERNING THE EVALUATION OF SUBSCRIPTS AND REFERENCE MODIFIERS

If identifier-1 to identifier-7 are subscripted or reference-modified or if a function-identifier is specified for any identifier other than identifier-2, the subscript, reference identifier, and function-identifier are evaluated only once at the start of the execution of the INSPECT statement.

SCOPE OF CHARACTER STRING INSPECTION

1. Neither the BEFORE phrase nor the AFTER phrase is written, all the character string specified by identifier-1 are subject to character string inspection.

2. If the BEFORE phrase is written, those character strings specified by identifier-1 that intervene between the leftmost character position and the character string just before the first character string that matches identifier-4 or literal-2 are subject to character string inspection. The rightmost character position in the scope of character string inspection is determined before the start of the comparison with the inspected character string. If no characters are found among the character strings specified by identifier-1 that match identifier-4 or literal-2, the BEFORE phrase is assumed to have not been specified.

3. If the AFTER phrase is written, those character strings specified by identifier-1 that intervene between the character string just after the first character string that matches identifier-4 or literal-2 and the rightmost character position are subject to character string inspection. The leftmost character position in the scope of character string inspection is determined after the start of the comparison with the inspected character string. If no characters are found among the character strings specified by identifier-1 that match identifier-4 or literal-2, the comparison with the inspected character string is suppressed.

PROCEDURES FOR COMPARISON WITH THE INSPECTED CHARACTER STRING

1. A comparison with an inspected character string takes place in the sequence described below. The first term "current leftmost position" below refers to the leftmost character position in the scope of inspection for the character strings specified by identifier-1.

   a. The character string that begins at the "current leftmost position" is compared with the inspected character string the number of times equal to the number of characters in the inspected character string.

   b. If the character string matches in the comparison described in (a), all its occurrences are tallied and replaced according to the rules for "Counting occurrences and replacing character strings" explained below. Then, the "current leftmost character position" moves to the character position just to the right of the rightmost character in the character string compared.

c. When the character string does not match in the comparison described in (a), if two or more inspected character strings are specified, the character string beginning at the "current leftmost position" is compared with the inspected character string in the order of their entry. This comparison is repeated until the character string matches or there is no more inspected character string. If the character string matches, the same operation as explained in (b) is executed. If there is no more inspected character string, the "current leftmost character position" moves to the character position just to the right of the rightmost character in the character string compared.

d. The operations explained in (a) through (c) are repeated until the "current leftmost character position" exceeds the scope of inspection for the character strings specified by identifier-1.

2. If the CHARACTERS phrase is written, one implicit character is assumed as an inspected character string and all the characters are assumed to match the inspected character string at all times in the inspected character string comparison.

COUNTING OCCURRENCES AND REPLACING CHARACTER STRINGS

1. If the ALL phrase is written, all the occurrences of the character string falling in the scope of that match the inspected character string are tallied and replaced.
2. If the LEADING phrase is written, only those occurrences of the character string, beginning at the leftmost position in the scope of character string inspection, that match the inspected character string are tallied and replaced until a character string different from the inspected character string appears.

3. If the FIRST phrase is written, the occurrences of only the first character string to match the inspected character string in the scope of character string inspection are tallied and replaced.
4. If the CHARACTERS phrase is written, the occurrences of all the character strings in the scope of character string inspection are tallied and replaced.

INSPECT Statement Example (1)

```
(a)  INSPECT ITEM TALLYING
          COUNT-0 FOR ALL "AB",  ALL "D"
          COUNT-1 FOR ALL "BC"
          COUNT-2 FOR LEADING "EF"
          COUNT-3 FOR LEADING "B"
          COUNT-4 FOR CHARACTERS.
(b)  INSPECT ITEM REPLACING
          ALL "AB" BY "XY",  "D" BY "X"
          ALL "BC" BY "VW"
          LEADING "EF" BY "TU"
          LEADING "B" BY "S"
          FIRST "G" BY "R"
          FIRST "G" BY "P"
          CHARACTERS BY "Z".
```

The table below summarizes the results of execution of the INSPECT statements (a) and (b).

Initial value of ITEM	**Result of execution of (a) (*1)**					**Result of execution of (b)**
	COUNT-0	**COUNT-1**	**COUNT-2**	**COUNT-3**	**COUNT-4**	**Final value of ITEM**
EFABDBCGABCFGG	3	1	1	0	5	TUXYXVWRXYZZPZ
BABABC	2	0	0	1	1	SXYXYZ
BBBC	0	1	0	2	0	SSVW

*1 The initial values of COUNT-0 to COUNT-4 are all assumed 0.

INSPECT Statement Example (2)

```
(a)  INSPECT ITEM TALLYING
          COUNT-0 FOR CHARACTERS
          COUNT-1 FOR ALL "A".
(b)  INSPECT ITEM REPLACING
          CHARACTERS BY "Z"
          ALL "A" BY "X".
```

The following table summarizes the results of execution of the INSPECT statements (a) and (b).

Initial Value of Item	Result of Execution of (a)(*1)		Result of Execution of (b)
	COUNT-0	COUNT-1	Final value of ITEM
BBB	3	0	ZZZ
ABA	3	0	ZZZ

*1 The initial values of COUNT-0 and COUNT-2 are all assumed 0.

INSPECT Statement Example (3)

```
(a)  INSPECT ITEM TALLYING
             COUNT-0 FOR ALL "AB" BEFORE "BC"
             COUNT-1 FOR LEADING "B" AFTER "D"
             COUNT-2 FOR CHARACTERS AFTER "A" BEFORE "C".
(b)  INSPECT ITEM REPLACING
             ALL "AB" BY "XY" BEFORE "BC"
             LEADING "B" BY "W" AFTER "D"
             FIRST "E" BY "V" AFTER "D"
             CHARACTERS BY "Z" AFTER "A" BEFORE "C".
```

The table below summarizes the results of execution of the INSPECT statements (a) and (b).

Initial Value of ITEM	Result of execution of (a) (*1)			Result of execution of (b)
	COUNT-0	COUNT-1	COUNT-2	Final value of ITEM
BBEABDABABBCABEE	3	0	2	BBEXYZXYXYZCABVE
ADDDDC	0	0	4	AZZZZC
ADDDDA	0	0	5	AZZZZZ
CDDDDC	0	0	0	CDDDDC
BDBBBDB	0	3	0	BDWWWDB

*1 The initial values of COUNT-0 to COUNT-2 are all assumed 0.

INSPECT Statement Example (4)

```
(a)  INSPECT ITEM TALLYING
          COUNT-0 FOR ALL "AB" AFTER "BA" BEFORE "BC".
(b)  INSPECT ITEM REPLACING
          ALL "AB" BY "XY" AFTER "BA" BEFORE "BC".
```

The following table summarizes the results of execution of the INSPECT statements (a) and (b).

Initial value of ITEM	**Result of execution of (a) (*1)**	**Result of execution of (b)**
	COUNT-0	**Final value of ITEM**
ABABABABC	1	ABABXYABC

*1 The initial value of COUNT-0 is assumed 0.

INSPECT Statement Example (5)

```
(a)  INSPECT ITEM CONVERTING
          "ABCD" TO "XYZX" AFTER QUOTE BEFORE "@".
```

The table below summarizes the result of execution of this INSPECT statement.

Initial value of ITEM	**Final value of ITEM**
AC”AEBDFBCD@AB”D	AC”XEYXFYZX@AB”D

MERGE Statement (Sort-merge)

Merges two or more files together.

[Format]

$$\underline{\text{MERGE}}\ \text{file-name-1}$$

$$\{\text{ON} \left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DECENDING}}\end{array}\right\} \text{KEY } \{\text{data-name-1}\}\ldots\}\ldots$$

$$\begin{array}{l}[\text{COLLATING } \underline{\text{SEQUENCE}} \text{ IS alhabet-name-1}]\\ \underline{\text{USING}}\ \text{file-name-2 } \{\text{file-name-3}\}\ldots\end{array}$$

$$\left\{\begin{array}{l}\underline{\text{OUTPUT}}\ \underline{\text{PROCEDURE}}\ \underline{\text{IS}}\\ \quad\text{procedure-name-1} \left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\} \quad \text{procedure-name-2}\right]\\ \underline{\text{GIVING}}\ \{\text{file-name-4}\}\ \ldots\end{array}\right\}$$

Syntax Rules

1. The MERGE statement may appear anywhere in the procedure division, except for the declarative.
2. file-name-1 must be defined in sort merge file description entry of data division
3. File-name-2 and file-name-3 must conform to the following rules:
    a. If the file named by file-name-1 has a variable-length record format, the record length of the files named by file-name-2 and file-name-3 must be greater than or equal to the minimum record length of the file named by file-name-1 but must not exceed the maximum record length.

   b. If the file named by file-name-1 has a fixed-length record format, the record length of the files named by file-name-2 and file-name-3 must not exceed that of file-name-1.

4. data-name-1 must conform to the following rules:

   a. data-name-1 must be a data item defined in the record description entry of file-name-1.

   b. data-name-1 may be qualified.

   c. A group item having a variable occurrence data item subordinate to it may not be specified in data-name-1.

   d. If two or more record description entries are written in file-name-1, the data item written in one record description entry must be specified in the data-name-1 list. The same character position as the data item in data-name-1 in that record description entry is assumed a key in every record in that file.

   e. A data item in which an OCCURS clause is written or a data item that is subordinate to such a data item may not be specified with data-name-1.

   f. If file-name-1 has a variable-length record format, the data item of data-name-1 must be included in the range of the size of the smallest record in file-name-1.

   g. data-name-1 may not be a Boolean item.

   h. For the maximum permissible number of data items that can be specified with data-name-1, see Appendix B, “System Limitations.”

5. file-name-2, file-name-3, and file-name-4 must be defined in a file description entry other than the sort-merge file description entry in the data division.

6. Two files specified in a single MERGE statement may not exist on the same set of file reels.
7. The same file may not be named by file-name-1 to file-name-4.
8. More than one set of file-name-1 to file-name-4 may not be specified in a single SAME AREA clause or in a single SAME SORT/SORT-MERGE AREA clause, though more than one set of file-name-4 may appear in a single SAME RECORD AREA clause.
9. THROUGH and THRU are synonymous.
10. If an indexed file is named by file-name-4, it must conform to the following rules:
    a. The first KEY phrase to appear must be an ASCENDING KEY phrase.
    b. The inter-record character position of the data item specified with the first data-name-1 must be equal to that of the data item associated with the prime record key in the indexed file. The two data items must be of the same length.
11. File-name-4 must conform to the following rules:
    a. If the file named by file-name-4 has a variable-length record format, the record length of the file named by file-name-1 must be greater than or equal to the minimum record length of the file named by file-name-4 but must not exceed the maximum record length.
    b. If the file named by file-name-4 has a fixed-length record format, the record length of the file named by file-name-1 must not exceed that of file-name-4.

12. alphabet-name-1 may not be one associated with a function name in the ALPHABET clause in a special names paragraph.

13. The CHARACTER TYPE clause or PRINTING POSITION clause may not be written in the record description entries of file-name-1 to file-name-4.

14. file-name-2 to file-name-4 must name any one of the following files:

    a. A sequence file on a mass storage or floppy disk device,

    b. A sequential access method relative file, or

    c. A sequential access method indexed file.

15. For the maximum permissible number of files that can be named by file-name-2 and file-name-3, see Appendix B, “System Quantitative Restrictions.”

General Rules

Rules Governing the Coding of the MERGE Statement as a Whole

1. The records of file-name-2 and file-name-3 must be arranged in the order as specified in the ASCENDING KEY or DESCENDING key phrase. If the records are not arranged in the specified order, the validity of the result of execution of the MERGE would be unpredictable.

2. Executing a MERGE statement would cause the following operations to be executed:

a. A READ statement is executed for the files named by file-name-2 and file-name-3, with the record read by the READ statement being passed to the merge module. The files named by file-name-2 and file-name-3 must not already be open before the start of execution of this stage The files named by file-name-2 and file-name-3 open and close automatically during this stage. This process is called the first stage of a merge operation.

b. All the records in the files named by file-name-2 and file-name-3 are merged. This process is called a merge operation.

c. The following processes make the records thus merged available for use as records for the file named by file-name-1:

   This process is called the last stage of a merge operation.

- If the OUTPUT PROCEDURE phrase is written,, an output procedure is executed. An output procedure is one written in procedure-name-1 or in procedure-name-1 through procedure-name-2. If the RETURN statement is executed for the file named by file-name-1 in the output procedure, the records in the file named by file-name-1 become available for use.

- If the GIVING phrase is written, the merged records become available for use as those in file-name-1 and are written to the file named by file-name-4. The file named by file-name-4 must not already be open before the start of execution of this stage The file named by file-name-4 opens and closes automatically during this stage.

3. In the first stage of the merge operation, the following processes are executed in sequence:

    a. Initialization processing is executed. Initialization is equivalent to the execution of an OPEN statement with the INPUT phrase specified for the files named by file-name-2 and file-name-3. If the OUTPUT PROCEDURE phrase is written, initialization takes place before control passes to the output procedure.

    b. The records in the files named by file-name-2 and file-name-3 are passed to the merge operation. This is equivalent to the execution of a READ statement with the NEXT and ATTEND phrases specified for the files named by file-name-2 and file-name-3 until there are no more records to read.

    c. Termination processing is executed. This is equivalent to the execution of a CLOSE statement with only file-name-2 or file-name-3 specified. If the OUTPUT PROCEDURE phrase is written, termination is not executed until control has passed to the last statement in the output procedure. It may happen during the processes (a) through (c) above that a USE AFTER STANDARD EXCEPTION procedure relating to the files named by file-name-2 to file-name-4 is executed. Statements that manipulate any of the files named by file-name-2 to file-name-4 or that reference records in these files may not be executed during the USE procedure.

4. If the file named by file-name-1 has a fixed-length record format and file-name-2 and file-name-3 have a record format shorter than that of the file named by file-name-1, the vacant positions in the right-side of each record are padded with blanks when the record is passed to the file named by file-name-1.

5. The data item named by data-name-1 is called a key item. When two or more key items are specified, they are arranged from left to right in the order of descending key strength.

   a. If the DESCENDING phrase is written, the values of the key items of the records in the files named by file-name-2 and file-name-3 are compared according to the rules of comparison and are merged in the order of ascending key item values.

   b. If the DESCENDING phrase is written, the values of the key items of the records in the files named by file-name-2 and file-name-3 are compared according to the rules of comparison and are merged in the order of descending key item values.

6. If, as a result of comparison in (5), two or more records are found in which all the key items have the same contents, the records in the files named by file-name-2 and file-name-3 are returned in the order of their entry in the file list. If such multiple records exist in the single file named by either file-name-2 or file-name-3, the records are returned before they are returned from any other file.

7. Where the rules of nonnumeric comparison apply, the collating sequence of the key items is determined by the following rules:

   a. If the COLLATING SEQUENCE phrase is written, the collating sequence of characters associated with alphabet-name-1 governs.

b. When the COLLATING SEQUENCE phrase is omitted, if the PROGRAM COLLATING SEQUENCE clause is written in the source computer paragraph in the identification division, the collating sequence of the characters specified in that clause governs. If this clause is omitted, the computer-specific character collating sequence governs.

Rules for OUTPUT PROCEDURE Phrase

1. If the OUTPUT PROCEDURE phrase is written, control passes to the output procedure upon completion of the merge operation. Control passes further to the executable statement next to the MERGE statement after execution of the last statement in the output procedure.

2. When control passes to the output procedure, the records in the file named by file-name-1 have been completely merged and a ready for return in sequence by a RETURN statement, which is written in the output procedure in association with the file named by file-name-1. The RETURN statement returns on record from the last stage of the merge operation.

3. The statements that are executed in the output procedure are called the scope of the output procedure. The scope of the output procedure covers the following:

   a. Statements written in the output procedure

   b. All the statements that are executed by the CALL, EXIT, GO TO, and PERFORM statements written in the output procedure

   c. All the statements in the declarative procedure that are executed by any statement written in the output procedure

4. A MERGE, RELEASE, or SORT statement may not be executed in the output procedure scope.

Rules for GIVING Phrase

1. If the GIVING phrase is written, all the records in the file named by file-name-1 are moved to the file named by file-name 4 upon completion of the merge operation.
2. The file named by file-name-4 must not already be open before the start of execution of the MERGE statement.
3. The following processes are performed in sequence on the file named by file-name-4:
   a. Initialization processing is executed. Initialization is equivalent to the execution of an OPEN statement with the OUTPUT phrase specified for the file named by file-name-4.

   b. The records in the file named by file-name-1 are returned form the merge operation. This is equivalent to the execution of a WRITE statement with only a record name specified for the file named by file-name-1 until there are no more records to write.

      If a relative file is named by file-name-4, the relative record numbers of the records that are returned are assigned in sequence beginning with the first record as 1, 2, 3, and so on. If a relative key item is specified in the file named by file-name-4, the relative key item is set to a relative record number each time a record is returned. The relative key item, after the execution of the MERGE statement, points to the last record returned to the file named by file-name-4.

   c. Termination processing is executed. This is equivalent to the execution of a CLOSE statement with only file-name-4 specified.

      It may happen during the processes (a) through (c) above that a USE AFTER STANDARD EXCEPTION procedure relating to the file named by file-name-4 is executed. Statements that manipulate the file named by file-name-4 or that reference records in that file may not be executed during the USE procedure.

4. When an attempt is made for the first time to write a record beyond the range defined externally to the file named by file-name-4, either of the following processes is executed:

   a. If a USE AFTER STANDARD EXCEPTION procedure relating to the file named by file-name-4 is written, that USE procedure is executed. After control returns from the USE procedure, termination processing is executed as if a CLOSE statement with only file-name-4 specified were executed.

   b. If a USE AFTER STANDARD EXCEPTION procedure relating to the file named by file-name-4 is not written, termination processing is executed as if a CLOSE statement with only file-name-4 specified were executed.

5. If the file named by file-name-4 has a fixed-length record format and file-name-1 has a record format shorter than that of the file named by file-name-4, the vacant positions in the right-side of each record are padded with blanks when the record is returned to the file named by file-name-4.

MOVE Statement (Nucleus)

Moves data from one data item to another.

[Format 1] Move data to one or more data items.

$$\underline{\text{MOVE}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \underline{\text{TO}} \ \{\text{identifier-2}\}\ldots$$

[Format 2] Move a data item that is subordinate to two group items by maintaining its correspondence.

$$\underline{\text{MOVE}} \left\{ \begin{array}{l} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{array} \right\} \text{identifier-1} \ \underline{\text{TO}} \ \text{identifier-2}$$

Syntax Rules

1. CORRESPONDING and CORR are synonymous.
2. identifier-1 and identifier-2 in format 2 must each be a group item.
3. identifier-1 and identifier-2 in format 2 may not be an index data item.

General Rules

Rules for Format 1

1. The MOVE statement in format 1 moves the operand before specified before TO to each operand specified after TO. The operand specified before TO is called a sending operand, and each operand written after TO is called a receiving operand.

2. If identifier-1 is subscripted or reference-modified or if a function-identifier is specified identifier-1, the subscript, reference identifier, and function-identifier are evaluated only once before the sending operand is moved to the starting receiving operand.
3. If identifier-2 is subscripted or reference-modified, the subscript and reference identifier are evaluated once just before the sending operand is moved to each receiving operand.
4. The length of the data item specified by identifier-1 is evaluated once just before the sending operand is moved to the receiving operands.
5. The length of the data item specified by identifier-2 is evaluated just before it is moved to each receiving operand.

Rule for Format 2

The MOVE statement in format 2 moves data items that are subordinate to identifier-1 and identifier-2 to maintain similarity in their qualifiers; that is, it moves the data item subordinate to identifier-1 as a sending data item and the data item subordinate to identifier-2 as a receiving data item. The MOVE statement in format 2 is equivalent to the execution of a separate MOVE statement for each corresponding identifier.

Rules Common to Format 1 and Format 2

1. If a variable occurrence data item is specified as identifier-1 or identifier-2, the evaluation of the length of the variable occurrence data item is influenced by the value of the data item specified in the DEPENDING ON phrase. For more details on the evaluation of the length of a variable occurrence data item, see the section titled “OCCURS clause.”

2. For more details on the valid combination of sending and receiving operands and the operation of the MOVE statement, see the section titled “Rules of Moving.”

Example of Subscript Evaluation

```
    MOVE a (b) TO b c (b)
```

The execution of the MOVE statement would yield the same result as that of the MOVE statement shown below. temp is a temporary data item.

```
    MOVE a (b) TO temp
    MOVE temp TO b
    MOVE temp TO c (b)
```

MULTIPLY Statement (Nucleus)

Calculates the result of multiplication.

[Format 1] Move data to one or more data items.

$$\underline{\text{MULTIPLY}}\ \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\}\ \underline{\text{BY}}\ \{\text{identifier-2 [}\underline{\text{ROUNDE}}\text{]}\}\ldots$$

[ON <u>SIZE</u> <u>ERROR</u> imperative-statement-1]
[<u>NOT</u> ON <u>SIZE</u> <u>ERROR</u> imperative-statement-2]
[<u>END-MULTIPLY</u>]

[Format 2] Stores the result of multiplication in a data item different from the multiplier.

$$\underline{\text{MULTIPLY}}\ \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\}\ \underline{\text{BY}}\ \left\{\begin{matrix}\text{identifier-2}\\ \text{literal-2}\end{matrix}\right\}$$

<u>GIVING</u> {identefier-3 [<u>ROUNDED</u>]}…
[ON <u>SIZE</u> <u>ERROR</u> imperative-statement-1]
[<u>NOT</u> ON <u>SIZE</u> <u>ERROR</u> imperative-statement-2]
[<u>END-MULTIPLY</u>]

Syntax Rules

1. identifier-1 and identifier-2 must each be a numeric data item.
2. identifier-3 must be a numeric edited data item.
3. literal-1 and literal-2 must each be a numeric literal.

General Rules

Rule for Format 1

The MULTIPLY statement in format 1 multiplies the operand specified before BY with identifier-2 and stores the result in identifier-2. It executes multiplication in the order of identifiers-2 specified.

Rule for Format 2

The MULTIPLY statement in format 2 multiplies the operand specified before BY with the operand specified after BY and stores the result in identifier-3. It executes multiplication in the order of identifiers-3 specified.

Rules Common to Format 1 and Format 2

1. The END MULTIPLY phrase delimits the scope of the MULTIPLY statement.
2. For more details on the ROUNDED phrase, ON SIZE ERROR phrase, and the rules of operation and moving, see the section titled “Common Rules Concerning Statements.”

OPEN Statement (Sequential File, Relative File, Indexed File)

The OPEN statement makes a file usable.

The RESERVE phrase in the OPEN statement of the sequential file is an obsolete element.

[Format 1] Sequential file

```
       ⎧ INPUT  { file-name-1 ⎡ REVERSED                ⎤ } ...   ⎫
       ⎪                      ⎢ WITH  ⎧ LOCK      ⎫     ⎥         ⎪
       ⎪                      ⎣       ⎩ NO REWIND ⎭     ⎦         ⎪
OPEN   ⎨ OUTPUT { file-name-2 ⎡ WITH  ⎧ LOCK      ⎫ ⎤ } ...       ⎬ ...
       ⎪                      ⎣       ⎩ NO REWIND ⎭ ⎦             ⎪
       ⎪ I-O {file-name-3 [WITH LOCK]} ...                        ⎪
       ⎩ EXTEND {file-name-4 [WITH LOCK]} ...                     ⎭
```

[Format 2] Relative file, indexed file

```
       ⎧ INPUT  {file-name-1 [WITH LOCK]} ...    ⎫
OPEN   ⎨ OUTPUT { file-name-2 [WITH LOCK]} ...   ⎬ ...
       ⎪ I-O {file-name-3 [WITH LOCK]} ...       ⎪
       ⎩ EXTEND {file-name-4 [WITH LOCK]} ...    ⎭
```

Syntax Rules

Rule Common to Files

A file with different ORGANIZATION clause phrase or file specified with different ACCESS MODE clause can be specified for the rows of file-names-1 to file-name-4.

Rules for Sequential Files

1. When the REVERSED or NO REWIND phrase is written, file-name-1 must be a record sequential file.
2. The WITH NO REWIND clause is regarded as a comment.
3. When the EXTEND phrase is written, file-name-4 must be a file without the LINAGE phrase. However, the EXTEND phrase cannot be written to a multiple file reel.
4. When the INPUT phrase is written, file-name-1 cannot be a print file.
5. When the I-O phrase is written, file-name-3 cannot be a line sequential file or print file.

Rules for Relative Files and Indexed Files

When the EXTEND phrase is written, file-name-1 must be a sequential access file.

General Rules

Rules Common to Files

file-name-n in the following description denotes file-name-1 to file-name-4.

1. When the OPEN statement has been executed successfully, a file assigned to file-name-n can be used in the program and the file is open. When the OPEN statement has been executed successfully, the file is related to file-name by means of a file connector.
2. When the file assigned to file-name-n physically exists and is recognized by the I-O-control system, the file is

usable. If not, the file is regarded as not usable. Results of OPEN statement execution in each status are listed below.

OPEN statement specification	File usable	File unusable
INPUT	The file is open.	The OPEN statement fails.
INPUT (Undefined file)	The file is open.	The file is open. The first reading leads to file termination or invalid key condition.
I-O	The file is open.	The OPEN statement fails.
I-O (Undefined file)	The file is open.	The file is created by executing the OPEN statement.
OUTPUT	The file is open. The file contains no records.	The file is created by executing the OPEN statement.
EXTEND	The file is open.	The OPEN statement fails.
EXTEND (Undefined file)	The file is open.	The file is created by executing the OPEN statement.

3. When the OPEN statement has been executed successfully, the record area of file-name-n becomes usable.

4. The input-output statement for the file must not be executed before opening the file.

5. Specify the file open mode to specify the INPUT, OUTPUT, I-O, and EXTEND phrases. The input-output statement to be executed after the file being open depends on the open mode. See the section titled “input-output statement operation,” for details of the relationship between the open mode and input-output statement.

6. The first record is not written or read by the OPEN statement.

7. When one run unit contains more than one execution of the OPEN statement for one file, a CLOSE statement must be executed before each subsequent execution. This CLOSE statement must not have a REEL, UNIT (sequential file only), or LOCK phrase.

8. If the file attribute conflict condition occurs during the OPEN statement execution, the statement execution fails and the following sequence of processing is performed:

    a. The I-O status for file-name-n is set.

    b. When the procedure for the USE AFTER STANDARD EXCEPTION related to file-name-n is written, the procedure has been executed. After the execution, the OPEN statement is controlled according to the rule for the USE statement. If the FILE STATUS clause is specified as file-name-n when the USE AFTER STANDARD EXCEPTION has not been written, control transfers to the end of the OPEN statement. The execution results without writing the EXECUTION procedure or the FILE STATUS clause are not defined.

9. When the OPEN statement has been executed, the value of the I-O status in file-name-n is updated.

10. When the I-O phrase is written, the file assigned to file-name-3 must be a file which can execute both the input and the output statements. When the OPEN statement in the I-O phrase has been executed successfully, the input and output statements can be executed.

11. Specify the file lock mode for the LOCK phrase. The file lock mode is determined by the descriptions of the LOCK MODE clause and the OPEN statement in the file control entry. See the section titled “File sharing and excluding,” for details on the file lock mode.

12. When the LOCK phrase has been written, the file assigned to file-name-n is locked by executing the open STATEMENT, which is the same as having EXCLUSIVE written in the LOCK MODE clause of file-name-n.

13. When the OPEN statement in exclusive mode has been executed successfully, the file related to the file connector of file-name-n is exclusively locked. If the file assigned to file-name-n has already been opened by another file connector, the OPEN statement execution fails.
14. After the OPEN statement in shared mode has been executed successfully, the file assigned to file-name-n can be opened by one or more file connectors. If the file assigned to file-name-n has already been opened by another file connector, the OPEN statement execution fails.
15. When the OPEN statement in the INPUT phrase has been executed first for the undefined file, the file position indicator is set to indicate that no undefined file exists.
16. When the OPEN statement in the EXTEND or I-O phrase for the undefined file which has not been created has been executed successfully, a file is created. The file creation is performed as if two statements have been executed in the following sequence:

    - OPEN OUTPUT file-name-n
    - CLOSE file-name-n

    In the created file, no record exists.
17. The execution results of the OPEN statement when there are two or more file-names-n are the same as when the OPEN statements have been executed for each file in the sequence in which the files have been written.
18. The minimum and the maximum sizes of the record in the file are determined according to the user specification when the file is created. These sizes cannot be changed afterwards.

Rules for Sequential Files

1. If the OPEN statement in the INPUT or I-O phrase has been executed successfully, 1 is set to the file position indicator of file-name-n.

2. If the OPEN statement in the EXTEND has been executed successfully, the file is allocated just after the first record. The last record of the sequential file is the last record written in the file.

3. When the record sequential file in the label record phrase is specified as file-name-n, the header label is processed as follows:

    a. When the OPEN INPUT statement has been executed, the label is tested according to the standard label rules.

    b. When the OPEN OUTPUT statement has been executed, the label is written according to the standard label rules.

    If there is no label even though the label record phrase exists, or there is a label but no label record phrase exists, the result of OPEN statement execution is undefined.

4. When the OPEN statement in the EXTEND phrase has been executed for the record sequential file with the label record phrase, the label is processed as follows:

    a. The header label is processed only when the file is a sequential single reel/unit file.

    b. When the file is a sequential multiple/unit file, the header volume label of the last reel/unit is tested according to the standard label rules.

    c. The ending file label is tested according to the standard label rules. After testing, this label is deleted.

    d. Then, the label is written according to the standard label rules.

5. When the OPEN statement in the I-O phrase has been executed for the record sequential file with the label record phrase, the label is processed as follows:

    a. The label is tested according to the standard label rules.

    b. The label is written according to the standard label rules.

6. The file under the environment of multiple file tapes is regarded to be the same as the sequential file under the environment of a single file tape.

7. When assigning a file on the multiple file reel to file-name-n, the following rule is applied.

    a. No more than one file can be opened on a one tape reel.

    b. When the OPEN statement in the INPUT phrase is executed, the position number of file-name-n need not be more than the position number of the file which has been opened before.

    c. When the OPEN statement in the OUTPUT phrase is executed, the position number of file-name-n must be the maximum number among the files on the related tape reel.

    d. file-name-n must be a sequential file.

8. The OPEN statement in which the NO REWIND or REVERSED phrase is written can be executed only for files as follows:

   a. Sequential single reel / unit file

   b. Sequential file under environment of multiple file tapes completely included in one tape reel

   See the section titled “MULTIPLE FILE TAPE clause (sequential file and report creation,)” for details on the multiple file tape environment.

9. When the NO REWIND or REVERSED phrase is written in the file assigned to a file-name-n which cannot apply either function (such as a mass storage file), these functions are ignored.

10. When a rewindable unit is assigned to a file-name-n file, the following rule is applied.

    a. If the OPEN statement has been executed when none of REVERSED, EXTEND, and NO REWIND phrase has been written, the file is allocated at the beginning.

    b. When the NO REWIND phrase has been written, the beginning of the file must be specified before executing the OPEN statement. When the OPEN statement has been executed successfully, the file is allocated to the specified position.

    c. When the REVERSED phrase has been written, the file is allocated at the end if the OPEN statement has been executed.

11. When the REVERSED phrase has been written, records can be used in the reversed order from the last record if the READ statement for the file assigned to file-name-n has been executed after executing the OPEN statement.

12. No label processing is performed for the line sequential file or print file.

13. The print file must not be opened by the OPEN statement in the INPUT phrase or OPEN statement in the I-O phrase.

14. When the OPEN statement for the sequential file has been executed successfully, the volume indicator is set as follows:

    a. When the OPEN statement with the INPUT or I-O phrase for usable files has been executed, the volume indicator is set to indicate the first reel/unit or single reel/unit.

    b. When the OPEN statement in the EXTEND phrase has been executed, the volume indicator is set to indicate the reel/unit containing the last record.

    c. When the OPEN statement with the OUTPUT or I-O phrase for unusable files has been executed, the volume indicator is set to indicate a new reel/unit.

Rules for Relative Files

1. When the OPEN statement with the INPUT or I-O phrase has been executed successfully, the file position indicator of file-name-n is set to 1.

2. If the OPEN statement in the EXTEND has been executed successfully, the file is allocated just after the last record. The last record of the relative file has the highest relative record number among existing records.

Rules for Indexed Files

1. When the OPEN statement with the INPUT or I-O phrase has been executed successfully, the file position indicator of file-name-n is set to the minimum value which can be assigned to the prime record key of the file. The prime record key is set as a key of reference.

2. If the OPEN statement in the EXTEND has been executed successfully, the file is allocated just after the last record. The last record of the indexed file has the highest prime record key value among existing records. However, if the DUPLICATES phrase is written in the RECORD KEY clause, the last record of the indexed file is the record which has been created last among existing records with the maximum prime record key values.

OPEN Statement (Presentation File)

The OPEN statement makes a file usable.

[Format]

$$\underline{\text{OPEN}} \left\{ \begin{array}{l} \underline{\text{INPUT}} \ \ \{\text{file-name-1}\}\ldots \\ \underline{\text{OUTPUT}} \ \ \{\text{file-name-2}\}\ldots \\ \underline{\text{I-O}} \ \ \{\text{file-name-3}\}\ldots \end{array} \right\} \ldots$$

Syntax Rules

In addition to presentation files, sequential files, relative files, and indexed files can be specified in a list of file-name-1, file-name-2, or file-name-3. Files for which the specifications in the ORGANIZATION or ACCESS MODE clause are different can also be specified in the list.

General Rules

The following explains while files can be assigned to file-name-1, file-name-2, and file-name-3:

1. When the OPEN statement has been executed successfully, a file assigned to file-name-n can be used in the program and the file is open. When the OPEN statement has been executed successfully, the file is related to file-name by means of the file connector.
2. When the OPEN statement has been executed successfully, the record area of file-name-n becomes usable.
3. The input-output statement for the file must not be executed before opening the file.
4. Specify the file open mode to specify the INPUT, OUTPUT, I-O, and EXTEND phrases. The input-output statement to be executed after the file being opened depends on the open mode. See the section titled “input-output statement operation,” for details of the relationship between the open mode and input-output statement.
5. The open mode to be specified for file-name-n depends on the destination type which has been specified in the SYMBOLIC DESTINATION clause in file-name-n. The following table shows the relationship between the destination type and open mode.

Destination Type	**Open Statement Specification**		
	INPUT	**OUTPUT**	**I-O**
ACM	○	○	○
DSP	-	-	○
PRT	-	○	○
APL	○	○	○
TRM	-	○	○

○: Executable

-: Unexecutable

6. The first record is not written or read by the OPEN statement.
7. When the OPEN statement has been executed, the value of the I-O status of file-name-n is updated.
8. The execution results of the OPEN statement when two ore more file-name-n are written are same as that when the OPEN statements have been executed for each file in the sequence in which the files have been written.

OPEN Statement (Report Writer)

The OPEN statement makes a file usable.

[Format]

$$\underline{\text{OPEN}} \left\{ \begin{array}{l} \underline{\text{OUTPUT}}\ \{\text{file-name-1}\ \ [\text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}}\ ]\}\ldots \\ \underline{\text{EXTEND}}\ \ \{\text{file-name-2}\}\ldots \end{array} \right\} \ldots$$

Syntax Rule

Only the OUTPUT or EXTEND phrase can be specified by the OPEN statement for the report writer.

General Rules

1. The OPEN statement for the report writer must be executed for the report related to the file before executing the INITIALIZE statement.
2. The rule for each clause of the OPEN statement in the report file is the same as that of the OPEN statement in the sequential file. See the section titled “OPEN statement (sequential file, relative file, indexed file).”

PERFORM Statement (Nucleus)

The PERFORM statement executes a series of statements repeatedly.

[Format 1] A group of statements is executed only once.

```
         { procedure-name-1 [ { THRU    } procedure-name-2 ] }
PERFORM  {                  [ { THROUGH }                  ] }
         { imperative-statement-1 END-PERFORM                }
```

[Format 2] A group of statements is executed as many times as specified.

```
         { procedure-name-1 [ { THRU    }  procedure-name-2 ]             }
         {                  [ { THROUGH }                   ]             }
PERFORM  {                    { identifier-1 }  TIMES                     }
         {                    { integer-1    }                            }
         { { identifier-1 }                                               }
         { { integer-1    } TIMES imperative-statement-1 END-PERFORM      }
```

[Format 3] A group of statements is executed repeatedly until conditions are satisfied.

```
PERFORM [ procedure-name-1 [ { THRU    }  procedure-name-2 ] ]
        [                  [ { THROUGH }                   ] ]
        [ WITH TEST { BEFORE } ] UNTIL condition-1
        [           { AFTER  } ]
        [imperative-statement-1 END-PERFORM]
```

[Format 4] The value of data item or index-name is changed according to the number of executions while a group of statements is executed repeatedly until conditions are satisfied.

$$\underline{\text{PERFORM}} \left[\text{procedure-name-1} \left[\left\{ \begin{matrix} \underline{\text{THRU}} \\ \underline{\text{THROUGH}} \end{matrix} \right\} \text{procedure-name-2} \right] \right]$$

$$\left[\text{WITH } \underline{\text{TEST}} \left\{ \begin{matrix} \underline{\text{BEFORE}} \\ \underline{\text{AFTER}} \end{matrix} \right\} \right]$$

$$\underline{\text{VARYING}} \left\{ \begin{matrix} \text{identifier-2} \\ \text{index-name-1} \end{matrix} \right\}$$

$$\underline{\text{FROM}} \left\{ \begin{matrix} \text{identifier-3} \\ \text{index-name-2} \\ \text{integer-1} \end{matrix} \right\} \underline{\text{BY}} \left\{ \begin{matrix} \text{identifier-4} \\ \text{integer-2} \end{matrix} \right\} \underline{\text{UNTIL}} \text{ condition-1}$$

$$\left[\underline{\text{AFTER}} \left\{ \begin{matrix} \text{identifier-5} \\ \text{index-name-3} \end{matrix} \right\} \right.$$

$$\left. \underline{\text{FROM}} \left\{ \begin{matrix} \text{identifier-6} \\ \text{index-name-4} \\ \text{integer-3} \end{matrix} \right\} \underline{\text{BY}} \left\{ \begin{matrix} \text{identifier-7} \\ \text{integer-4} \end{matrix} \right\} \underline{\text{UNTIL}} \text{ condition-2} \right] \ldots$$

$$[\text{imperative-statement-1 } \underline{\text{END-PERFORM}}]$$

[Format 5] A group of statements is executed repeatedly without any condition.

$$\underline{\text{PERFORM}} \text{ WITH } \underline{\text{NO}} \ \underline{\text{LIMIT}}$$

$$\text{imperative-statement-1 } \underline{\text{END-PERFORM}}$$

Syntax Rules

Rules Common to Format 1 to Format 5

1. Either procedure-name-1 or imperative-statement-1 must be written in formats 1 to 4. Both names cannot be written at the same time.
2. THROUGH is synonymous with THRU.
3. If one procedure-name is a procedure-name in the declarative part in the procedure section when writing both procedure-name-1 and procedure-name-2, another procedure-name must be in the same declarative section.
4. When neither TEST BEFORE nor TEST AFTER phrase is written in formats 3 and 4, TEST BEFORE is assumed to have been written.
5. The conditional expression explained in the section titled “Conditional expression,” should be written to condition-1 and condition-2 in formats 3 and 4.

Rules for Format 2

1. identifier-1 must be an integer item.
2. See Appendix B, “System Literal Restrictions,” for the maximum value to be specified for integer-1 and identifier-1.

Rules for Format 4

1. When imperative-statement-1 is written, the AFTER phrase must not written.
2. identifier-2 to identifiers-7 must be integer items.
3. literal-1 to literal-4 must be numeric literals.

4. literal-2 and literal-4 must not be zero.

5. When index-name-1 is written in the VARYING phrase, operands for the FROM phrase and the BY phrase must be written according to the following rules:

   a. identifier-3 and identifier-4 must be integer items.

   b. literal-1 must be a positive integer.

   c. literal-2 must be a non-zero integer.

6. When index-name-3 is written in the AFTER phrase, operands for the FROM phrase and the BY phrase must be written according to the following rules:

   a. identifier-6 and identifier-7 must be integer items.

   b. literal-3 must be a positive integer.

   c. literal-4 must be a non-zero integer.

7. When index-name-2 is written in the FROM phrase in the VARYING phrase, operands for the VARYING phrase and the BY phrase must be written according to the following rules:

   a. identifier-2 and identifier-4 must be integers.

   b. literal-2 must be a non-zero integer.

8. When index-name-4 is written in the FROM phrase in the AFTER phrase, operands for the VARYING phrase and the BY phrase must be written according to the following rules:

   a. identifier-5 and identifier-7 must be integers.

   b. literal-4 must be a non-zero integer.

9. Up to six AFTER phrases can be written.

General Rules

All Rules for PERFORM statements

1. A PERFORM statement containing procedure-name-1 is called an “out-of-line PERFORM statement.” A PERFORM statement containing procedure-name-2 is called an “in-line PERFORM statement.” A series of statements to be executed by the PERFORM statement is called a “group of statements.”

2. The out-of-line PERFORM statement repeatedly executes the group of statements written in another section or paragraph for the specified times or according to the condition.

3. The in-line PERFORM statement repeatedly executes the group of statements (imperative-statement-1) written in the in-line PERFORM statement for the specified times or according to the condition.

4. The same rule applies to the out-of-line PERFORM statement and the in-line PERFORM statement, excluding the difference whether the group of statements to be executed is written out of the statement or in the statement.

Rules for a Group of Statements to be Executed in the PERFORM Statement

1. The PERFORM statement logically consists of statements to be executed until the control returns at the end of the PERFORM statement. When the CALL statement, the EXIT statement, the GO TO statement, or the PERFORM statement is written in the group of statements, all statements to be executed by these statements are included in the range of the PERFORM statement. Statements included in the range need not be written serially in the program.
2. When the EXIT PROGRAM statement written in the same program as the PERFORM statement in the range of the PERFORM statement is executed, the statement to be executed as a result of control transferred by executing the EXIT PROGRAM statement is not regarded to be within the range of the PERFORM statement.

Rules for a Group of Statements to be Executed by the In-line PERFORM Statement

1. The in-line PERFORM statement repeatedly executes a group of statements specified in imperative-statement-1.
2. The END-PERFORM phrase partitions the range of the in-line PERFORM statement.
3. A group of statements to be executed by the in-line PERFORM statement contains statements to be executed from the first statement of the imperative statement to the statement just before the END-PERFORM statement.

Rules for a Group of Statements to be Executed by the Out-of-line PERFORM Statement

1. The out-of-line PERFORM statement repeatedly executes a group of statements written in another section or paragraph. Statements from the first statement of the procedure-name-1 to the following statements are executed repeatedly.

   a. When procedure-name-2 has been omitted and the paragraph name has been specified as a procedure-name-1, the out-of-line PERFORM statement executes statements up to the last statement of the paragraph.

   b. When procedure-name-2 has been omitted and the section-name has been specified as a procedure-name-1, the out-of-line PERFORM statement executes statements up to the last statement of the section.

   c. When the paragraph name is specified as a procedure-name-2, the out-of-line PERFORM statement executes statements up to the last statement of the paragraph.

   d. When the section-name is specified as a procedure-name-2, the out-of-line PERFORM statement executes statements up to the last statement of the section.

2. The relationship between procedure-name-1 and procedure-name-2 specifies the flow of control, starting from procedure-name-1 and ending at procedure-name-2. The GO TO statement or PERFORM statement can be written during the period from procedure-name-1 to procedure-name-2. To return control from the PERFORM statement which is not the last statement in the group of statements, a paragraph containing only the EXIT statement must be defined to return control from the paragraph containing only the EXIT statement to the PERFORM statement. procedure-name-2 must be a paragraph name of a paragraph containing only the EXIT statement or a section-name of a section containing a paragraph consisted of only the EXIT statement.

3. A group of statements can be executed by a statement other than the PERFORM statement. When the control transfers from a statement other than the PERFORM statement to a group of statements, control never returns to the PERFORM statement from the last statement of the group of statements. Instead, control transfers to the next execution statement of the last statement in the group of statements.

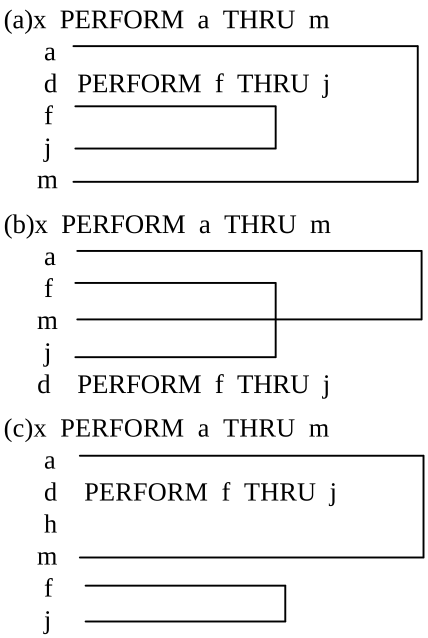
4. procedure-name-1 and procedure-name-2 must be section-names or paragraph names in the program in which the PERFORM statement has been written. A procedure-name in the program containing the program in which the PERFORM statement has been written or a procedure-name in the program contained in the program in which the PERFORM statement has been written cannot be specified as procedure-name-1 or procedure-name-2.

5. The following rules apply when writing another PERFORM statement in the range of the PERFORM statement. The former PERFORM statement is called “containing PERFORM statement” and the latter is called “contained PERFORM statement.”

   a. The range of the contained PERFORM statements must be either completely within or completely outside the range of logical execution of the containing PERFORM statement. That is, the exit of the containing PERFORM statement cannot be passed during execution of the contained PERFORM statement.

   b. PERFORM statements within the range of containing statements cannot have a common exit. In this compiler, however, one or more PERFORM statements within the range of containing statements can have the common exit.

6. The following example shows correct usage of the out-of-line PERFORM statement.



```
(a)x  PERFORM  a  THRU  m
    a
    d  PERFORM  f  THRU  j
    f
    j
    m
(b)x  PERFORM  a  THRU  m
    a
    f
    m
    j
    d  PERFORM  f  THRU  j
(c)x  PERFORM  a  THRU  m
    a
    d  PERFORM  f  THRU  j
    h
    m
    f
    j
```

Rule for Format 1

The PERFORM statement in format 1 executes a group of statements only once. Then, the control returns to the end of the PERFORM statement.

Rules for Format 2

1. The PERFORM statement in format 2 repeatedly executes a group of statements as many times as specified in the initial value of integer-1 or identifier-1. Then, the control returns to the end of the PERFORM statement.
2. When the value of identifier-1 before the PERFORM statement execution is zero or negative, control transfers to the end of the PERFORM statement without executing a group of statements.
3. Even if the value of identifier-1 has been changed during execution of a group of statements, the execution number never changes.

Rules for Format 3

1. The PERFORM statement in format 3 repeatedly executes a group of statements until condition-1 is satisfied. Then, the control returns to the end of the PERFORM statement.
2. When testing condition-1 before executing a group of statements, write the TEST BEFORE phrase or omit the TEST phrase. When testing condition-1 after executing a group of statements, write the TEST AFTER phrase.
3. When a subscript and a reference modifier are added to operands in condition-1, they are evaluated every time condition-1 is tested.

Rules for Format 4

1. The PERFORM statement in format 4 repeatedly executes a group of statements until the specific condition is satisfied and changes the value of the data item according to the repetitive execution. Then, the control returns to the end of the PERFORM statement.

   In the out-of-line PERFORM statement, one or more data items to be changed according to the repetitive execution can be specified in the VARYING phrase and the AFTER phrase. In the in-line PERFORM statement, only one data item to be changed according to the repetitive execution can be specified in the VARYING phrase.

   Requirements for ending repetitive execution are written after the UNTIL. When testing the at end condition before executing a group of statements, write the TEST BEFORE phrase or omit the TEST phrase. When testing the at end condition after executing a group of statements, write the TEST AFTER phrase.

2. identifier-4 and identifier-7 must not be zero.

3. When identifier-1 has been written in the VARYING phrase, the value of operand in the FROM phrase must be a value corresponding to the occurrence number of the table containing index-name-1. The value of operand in the BY phrase must be an increment of the value corresponding to the occurrence number of the table containing index-name-1.

4. A value outside the range of the table containing index-name-1 cannot be set to index-name-1 until condition-1 is satisfied. In the same way, a value outside the range of the table containing index-name-3 cannot be set to index-name-3 until condition-2 is satisfied. However, the value of index-name-1 after execution of the PERFORM statement may be as much as one increment or decrement value outside the range of the table.

5. When a subscript and a reference modifier are added to the identifier or operand in the condition of the UNTIL phrase, they are evaluated at the condition as follows:

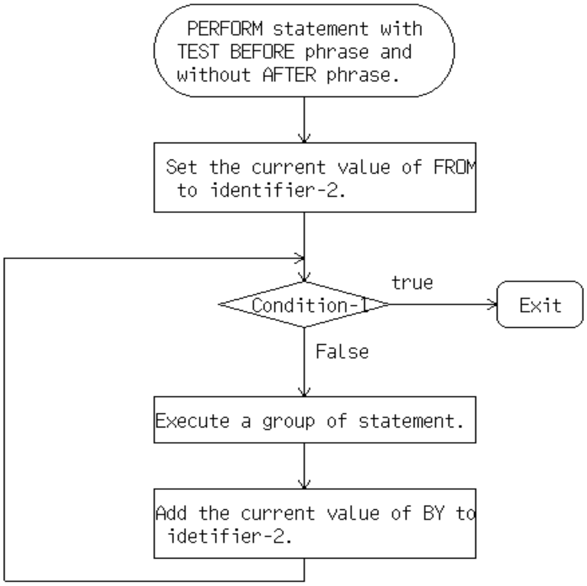
   a. Subscripts of identifier-2 and identifier-5 are evaluated every time initial values are set to these identifiers or increment values are added.

   b. Subscripts of identifier-3 and identifier-6 are evaluated when initial values are set to these identifiers.

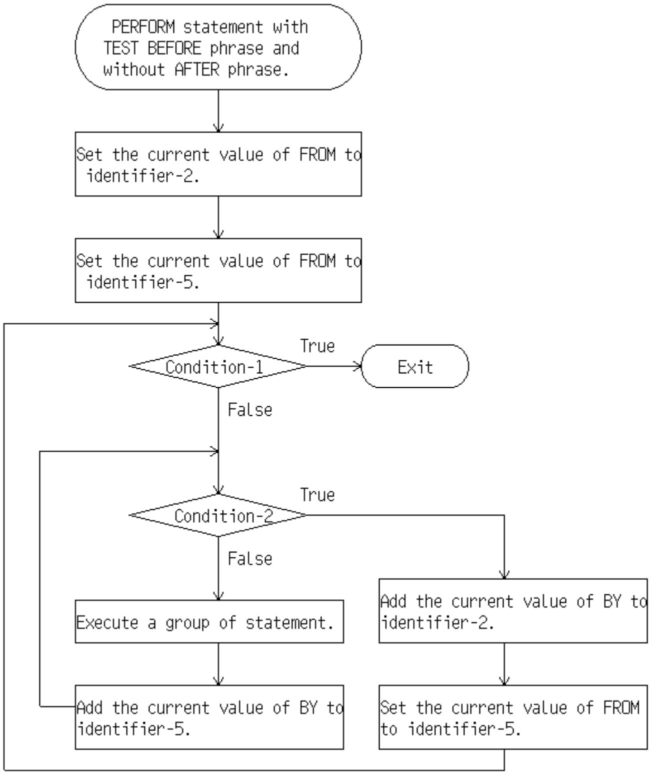
   c. Subscripts of identifier-4 and identifier-7 are evaluated when increment values are added.

   d. Reference modifiers added to condition-1 and condition-2 are evaluated every time condition-1 or condition-2 is tested.

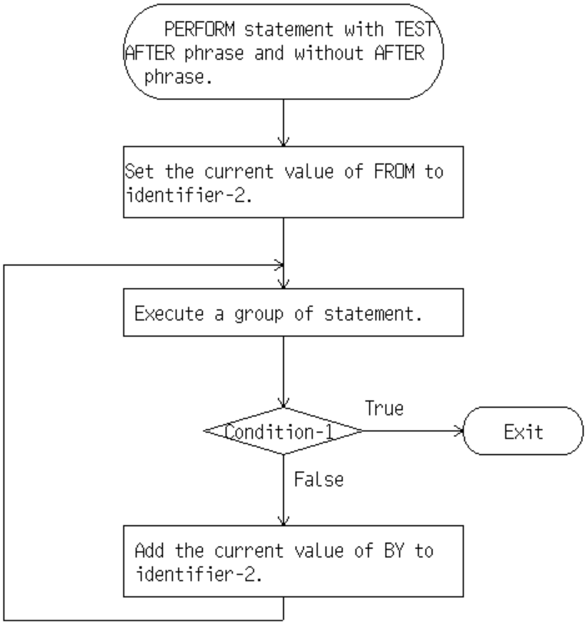
6. When the TEST BEFORE phrase is specified implicitly or explicitly, and the AFTER phrase is omitted, the processing will be as shown in the following figure.


PERFORM statement with TEST BEFORE phrase and without AFTER phrase.
Set the current value of FROM to identifier-2.
true
Condition-1
Exit
False
Execute a group of statement.
Add the current value of BY to idetifier-2.

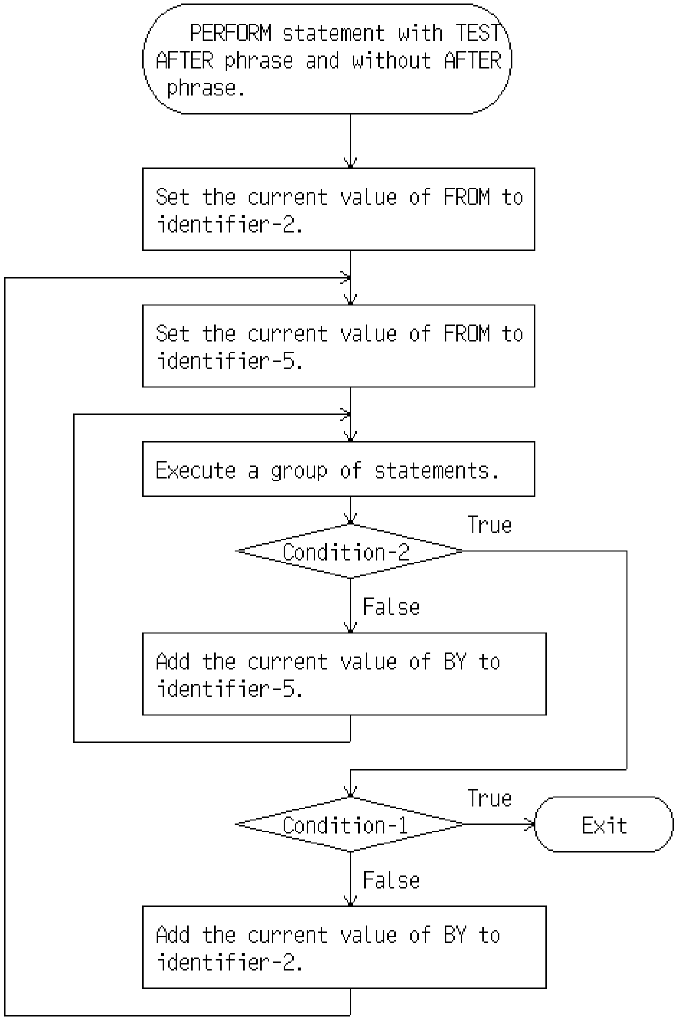
7. When the TEST BEFORE phrase is specified implicitly or explicitly, and the AFTER phrase is written, the processing will be as shown in the following figure.


PERFORM statement with TEST BEFORE phrase and without AFTER phrase.
Set the current value of FROM to identifier-2.
Set the current value of FROM to identifier-5.
Condition-1
True
Exit
False
Condition-2
True
False
Execute a group of statement.
Add the current value of BY to identifier-5.
Add the current value of BY to identifier-2.
Set the current value of FROM to identifier-5.

8. When the TEST AFTER phrase is written and the AFTER phrase is omitted, the processing will be as shown in the following figure.


PERFORM statement with TEST AFTER phrase and without AFTER phrase.
Set the current value of FROM to identifier-2.
Execute a group of statement.
Condition-1
True
Exit
False
Add the current value of BY to identifier-2.

9. When the TEST AFTER phrase is written, and one AFTER phrase is written, the processing will be as shown in the following figure.


PERFORM statement with TEST AFTER phrase and without AFTER phrase.
Set the current value of FROM to identifier-2.
Set the current value of FROM to identifier-5.
Execute a group of statements.
Condition-2
True
False
Add the current value of BY to identifier-5.
Condition-1
True
Exit
False
Add the current value of BY to identifier-2.

10. When two or more AFTER phrases have been written, the test feature of the AFTER phrase is the same as when only one AFTER phrase has been written. However, the processing for an operand in each AFTER phrase repeats every time the value of the operand in the preceding AFTER phrase is changed.

Rules for Format 5

The PERFORM statement in format 5 repeatedly executes a group of statements until the branch statement indicating explicit control transfer or the EXIT PERFORM statement is executed.

READ Statement (Sequential File, Relative File, Indexed File)

The READ statement reads a record from the file.

[Format 1] Records are read in order. (Sequential file, relative file, indexed file)

READ file-name-1 [NEXT] RECORD
[INTO identifier-1]
[WITH [NO] LOCK]
[AT END imperative-statement-1]
[NOT AT END imperative-statement-2]
[END-READ]

[Format 2] Records are read at random. (Relative file)

READ file-name-1 RECORD
[INTO identifier-1]
[WITH [NO] LOCK]
[INVALID KEY imperative-statement-3]
[NOT INVALID KEY imperative-statement-4]
[END-READ]

[Format 3] Records are read at random. (Indexed file)

```
READ file-name-1 RECORD
 [INTO identifier-1]
 [WITH [NO] LOCK]
 [KEY IS {data-name-1} ... ]
 [INVALID KEY imperative-statement-3]
 [NOT INVALID KEY imperative-statement-4]
 [END-READ]
```

Syntax Rules

Rules Common to Files

1. The storage area of identifier-1 must be different from the record area of file-name-1.
2. When the USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not written, the AT END or INVALID KEY must be written. In this compiler, however, the USE AFTER STANDARD EXCEPTION procedure and the AT END phrase or INVALID KEY phrase, or both phrases, can be omitted.
3. The data item defined in the literal section must not be specified as identifier-1.

Rules for Relative Files

1. Use format 1 to read records from a file in sequential access method.
2. Use format 1 to read records from a file sequentially in dynamic access method. The NEXT phrase must be written.
3. Use format 2 to read records from a file in random access method or at random in dynamic access method.

Rules for Indexed Files

1. Use format 1 to read records from a file in sequential access method.
2. Use format 1 to read records from a file sequentially in dynamic access method. The NEXT phrase must be written.
3. Use format 3 to read records from a file in random access method or at random in dynamic access method.
4. data-name-1 must be the name specified in the RECORD KEY or ALTERNATE RECORD KEY phrase in file-name-1. If two or more data-names have been specified in the RECORD KEY phrase when data-name of the RECORD KEY phrase is specified, the row of data-name-1 must be the same as the row of data-name specified in the RECORD KEY phrase, both in the specified sequence and number. If two or more data-names have been specified in the ALTERNATE RECORD KEY phrase when data-name in the ALTERNATE RECORD KEY phrase is specified, the row of data-name-1 must be the same as the row of data-name specified in the ALTERNATE RECORD KEY phrase, both in the specified sequence and number.
5. data-name-1 can be qualified.

General Rules

Rules Common to Format 1 to Format 3

RULES COMMON TO FILES

1. The file assigned to file-name-1 must be opened in input mode or I-O mode before executing the READ statement.
2. When the READ statement has been executed, the records to be read from the file are selected and the value of the

file position indicator of file-name-1 is set. The value of I-O status of file-name-1 is also updated. The next operation is determined depending on the value of the file position indicator.

3. If the READ statement execution failed, contents of the record area in file-name-1 are not defined. The file position indicator is set to indicate that there is no remaining valid record.
4. When the value of the file position indicator indicates that there is a valid record, the record becomes usable in the record area. When the INTO phrase has been written, a record is transferred from the record area to identifier-1.
5. When the number of record character positions of read record is less than the minimum record size specified in the record description entry of file-name-1, the contents of the part exceeding the number of record character positions will not be defined. When the number of record character positions of read record is more than the minimum record size specified in the record description entry of file-name-1, the read record is truncated on the right in accordance with the maximum record size and set in the record area of file-name-1.
6. The END-READ phrase partitions the range of the READ statement.

RULES FOR INTO PHRASE

1. The INTO phrase can be written in the following cases:
   a. Only one record entry of file-name-1 is written.
   b. All record-names and identifier-1 related to file-name-1 are group items or alphanumeric data items.
2. When the INTO phrase is written, the processing will be as follows:

a. The same READ statement without the INTO phrase is executed.

b. When the READ statement has been executed successfully in the processing (a), the current record is transferred from the record area of file-name-1 to identifier-1 according to the MOVE statement rule without the CORRESPONDING phrase. The read record becomes usable both in the record area of file-name-1 and the data item of identifier-1.

The current record size is determined in accordance with the rules of the RECORD phrase in file-name-1. When the RECORD phrase is written in the file description entry of file-name-1 in a format other than not format 3, the group items are transferred in the processing (b). The subscript added to identifier-1 is evaluated before processing (b), which is after processing (a). If processing (a) fails, the implicit MOVE statement is not executed.

RULES FOR RECORD LOCKS

1. When either of the following READ statements is executed for the file specified with the LOCK MODE IS AUTOMATIC phrase, the read record is locked.

   a. READ statement with LOCK phrase

   b. READ statement without LOCK phrase or NO LOCK phrase

2. When the READ statement with LOCK phrase has been executed for the file specified with the LOCK MODE IS MANUAL phrase (relative file or indexed file), the read record is locked.

3. When the READ statement has been executed for the file opened in input mode, the record is not locked, regardless of whether the LOCK phrase has been written.

4. If the record to be read on the READ statement by locking is locked by another file connector when the READ statement execution starts while the record is being locked implicitly or explicitly, the READ statement execution fails. The value indicating that the record is locked is set. At that time, the value of the file position indicator is not changed.

Rules for Format 1

RULES COMMON TO FILES

1. When sequential file, relative file in sequential access method, or indexed file in sequential access method is specified as file-name-1, the NEXT phrase can be omitted. The NEXT phrase will not affect the READ statement execution at all.
2. The READ statement in which sequential file, relative file in sequential access method, or indexed file in sequential access method is specified to file-name-1 calls the next record from the file assigned to file-name-1.
3. The READ statement in which the relative file in dynamic access method or the indexed file in dynamic access method is specified to file-name-1 and in which the NEXT phrase is specified calls the next record from the file assigned to file-name-1.
4. When the READ statement has been executed, the value of the file position indicator is determined in accordance with the rules for determining a usable record. The next operation is determined depending on the value of the file position indicator.

RULES FOR DETERMINING A USABLE RECORD (SEQUENTIAL FILE)

1. The value of the file position indicator to start the READ statement execution is used to determine the usable record in accordance with the following rules. Records in the sequential file are compared by record numbers.

    a. When the file position indicator indicates that there is no remaining valid record, the READ statement execution fails.

    b. When the file position indicator indicates that there is no optional file, the file at end condition occurs.

    c. When the file position indicator is set by an OPEN statement which has been previously executed, the first file record having a record number greater than or equal to the file position indicator is selected.

    d. When the file position indicator is set by a READ statement which has been previously executed, the first file record having a record number greater than the file position indicator is selected.

2. If no record satisfies conditions (c) and (d) above, the file at end condition occurs. The value to indicate that there is no more record is set at the file position indicator.

3. If a record satisfies the conditions (c) and (d) above is found, the record becomes usable in the record area related to file-name-1. Then, the file position indicator is set at the record number of the usable record.

RULES FOR DETERMINING A USABLE RECORD (RELATIVE FILE)

1. The value of the file position indicator to start the READ statement execution is used to determine the usable record in accordance with the following rules. Records in the sequential file are compared by relative key numbers.

   a. When the file position indicator indicates that there is no remaining valid record, the READ statement execution fails.

   b. When the file position indicator indicates that there is no optional file, the file at end condition occurs.

   c. When the file position indicator is set by an OPEN statement which has been previously executed, the first file record having a record number greater than or equal to the file position indicator is selected.

   d. When the file position indicator is set by a START statement which has been previously executed, the file record having a relative record number equal to the file position indicator is selected.

   e. When the file position indicator is set by a READ statement which has been previously executed, the first file record having a relative record number greater than the file position indicator is selected.

2. If no record satisfies the conditions (c) and (d) above, the file at end condition occurs. The value to indicate that there is no more record is set at the file position indicator.

3. If a record satisfies conditions (c) and (d) above is found (selected record), the processing will be as follows:

   a. When the RELATIVE KEY phrase is written in the ACCESS MODE phrase of file-name-1, the number of valid digits of the relative record number in the selected record and the size of the relative key item are compared.

   - If the number of valid digits of the relative record number is greater than the size of the relative key item, the file at end condition occurs.

   - At the same time, the value indicating that the number of valid digits of the relative record number exceeded the size of the relative key item is set.

   - If the number of valid digits of the relative record number is less than the size of the relative key item, the selected record becomes usable in the record area in file-name-1. At the same time, the relative record number of the usable record is set in the file position indicator. The relative record number of the usable record is transferred to the relative key item in accordance with the rule for moving.

   b. When the RELATIVE KEY phrase has not been written in the ACCESS MODE phrase of file-name-1, the selected record becomes usable in the record area of file-name-1. At the same time, the relative record number of the usable record is set in the file position indicator.

RULES FOR DETERMINING A USABLE RECORD (INDEXED FILE)

1. The value of the file position indicator to start the READ statement execution is used to determine the usable record in accordance with the following rules. Records in the sequential file are compared by the order of keys of reference at present, depending on the values.

   a. When the file position indicator indicates that there is no remaining valid record, the READ statement execution fails.

   b. When the file position indicator indicates that there is no optional file, the file at end condition occurs.

   c. When the file position indicator is set by an OPEN statement which has been previously executed, the first record in the file with the current key of reference is selected.

   d. When the file position indicator is set by a START statement which has been previously executed, the file record having the key value as the file position indicator is selected.

   e. When the file position indicator is set by a READ statement which has been previously executed, a record that satisfies either of the following conditions is selected:

  - Record which is logically allocated just after the record which became usable by execution of the previous READ statement.
  - If the direction for searching is reversed by executing the START statement in the REVERSED ORDER phrase, the record which is logically after the record which became usable by the READ statement previously executed is selected. That is, the record which is physically following the record to the direction from the record at the end of the file to the preceding record is selected.

2. If no record to satisfy the conditions (c), (d), and (e) above is found, the file at end condition occurs. A value indicating no remaining record is set at the file position indicator.

3. If the record to satisfy the conditions (c), (d), and (e) above is found, the record becomes usable in the record area of file-name-1. At the same time, the current key of reference of the usable record is set in the file position indicator.

4. If the READ statement execution failed, the value of the key of reference is not defined.

5. When the READ statement has been executed with the prime record key or alternate record key as a key of reference, there may be multiple records with key reference values. In that case, records are read in the following sequence:

   a. When the file position indicator is set by the START statement in the REVERSED ORDER previously executed, records are read in the reversed order of the order in which multiple records have been created with the keys of reference by the WRITE or REWRITE statement.

b. Other cases: Records are read in the order which multiple records have been created with the keys of reference by the WRITE or REWRITE statement.

6. If the READ statement in format 1 has been executed for the file in dynamic access method, following the execution of the READ statement in format 3, the following key of reference is used in the READ statement in format 1 and records are read in the direction for searching as follows:

   a. The key of reference set by the execution of the READ statement in format 3 is used until the key of reference is changed.

   b. Records are searched in the normal ascending order until the START statement in the REVERSED ORDER phrase is executed.

CONTROL TRANSFER DEPENDING ON OCCURRENCE OF EXCEPTIONAL CONDITION

1. When the file at end condition occurs during the READ statement execution, the READ statement execution fails. After the value indicating the file at end condition is set in the I-O status of file-name-1, the control transfers according to the rules shown in the section titled “AT END phrase.”

2. When an exceptional condition other than the file at end condition occurs during the READ statement execution, the control transfers according to the rules shown in the section titled “AT END phrase,” after the file position indicator and the I-O status of file-name-1 are set.

3. If no file at end condition or other exceptional condition occurs during the READ statement execution, the processing will be as follows:

   a. The file position indicator and the I-O status of file-name-1 are set.

   b. The record which has been read becomes usable in the record area.

   c. The control transfers according to the rules shown in the section titled "AT END phrase."

RULES RELATED TO REEL/UNIT FILE IN SEQUENTIAL FILES

When the end of reel/unit during the READ statement execution for the sequential file has been detected, or when no record exists in the reel/unit, the following processing sequence will be performed if it is not the logical end of the file.

1. The standard procedure for end reel/unit labels is performed.

2. The reel/unit is replaced. The volume indicator is updated to indicate the next reel/unit.

3. The standard procedure for start reel/unit labels is performed.

RULES FOR LINE SEQUENTIAL FILES

1. When the READ statement has been executed for a line sequential file, some record data may be transferred to the internal format. Refer to "COBOL85 User's Guide" for the transfer rules for record data.

2. When the READ statement has been executed for the line sequential file, record data will be set as follows:

   a. When the length up to the record delimiter is longer than the maximum record length, the section up to the record delimiter is set in the record area.

   b. When the length up to the record delimiter is shorter than the maximum record length, the section up to the record delimiter is set in the record area and the remaining section of the record area is set to blanks.

   c. When the length up to the record delimiter is longer than the maximum record length, the whole record data is read by executing two or more READ statements repeatedly. At the execution of the first READ statement, the data as long as the maximum record length is read from the beginning of the record. At the execution of the second READ statement or later, the data starting from the next character position of the section read by the preceding READ statement is read. Rules (a) to (c) are applied in repetitive execution of READ statements.

Rules for Format 2

1. When the file position indicator to be used to start the READ statement execution indicates that there is no optional file for input, the invalid key condition occurs. The READ statement execution fails.

2. The relative record number of the record to be read must be set in the relative key item of file-name-1 before the READ statement execution. When the READ statement has been executed, the value of the relative key item is set in the file position indicator. If a record having the relative record number equal to the value of the file position indicator is found, the record becomes usable in the record area related to file-name-1. If no record having the relative record number equal to the value of the file position indicator is found, the file at end condition occurs and the READ statement execution fails.

Rules for Format 3

1. When the file position indicator to be used to start the READ statement execution indicates that there is no optional file for input, the invalid key condition occurs. The READ statement execution fails.
2. When the KEY phrase has been written, data-name-1 becomes the key of reference for the READ statement.
3. When the KEY phrase has been omitted, the prime record key becomes the key of reference for the READ statement.
4. The value of the key of reference is set in the file position indicator by the READ statement execution. To select the record to be read from the file, the section corresponding to the record key of reference is compared with the value of the file position indicator. The comparison operation repeats until the first record in which the section corresponding to the record key of reference matches with the value of the file position indicator is found.

When the record to satisfy the condition is found as a result of the comparison, the record becomes usable in the record area related to file-name-1. If no record having a relative record number equal to the value of the file position indicator is found, the file at end condition occurs and the READ statement execution fails.

5. If the READ statement execution failed, the value of the key of reference is not defined.

Rules Common to Format 2 and Format 3

1. When the invalid key condition occurs during the READ statement execution, the control transfers according to the rule shown in the section titled "INVALID KEY phrase," after the value indicating the invalid key condition is set in the I-O status of file-name-1.
2. If an exceptional condition other than the invalid key condition occurs during the READ statement execution, the control transfers according to the rules shown in the section titled "INVALID KEY phrase," after the file position indicator and the I-O status of file-name-1 are set.
3. When no invalid key condition or other exceptional condition occurs during the READ statement execution, the following processing sequence will be performed:

    a. The file position indicator and the I-O status of file-name-1 are set.

    b. The record which has been read becomes usable in the record area. The implicit transfer is performed if the INTO phrase has been written.

    c. The control transfers according to the rule shown in the section titled "INVALID KEY phrase."

READ Statement (Presentation File)

The READ statement reads a record from a file.

[Format]

```
 READ file-name-1 [NEXT] RECORD
 [INTO identifier-1]
 [AT END imperative-statement-1]
 [NOT AT END imperative-statement-2]
[END-READ]
```

Syntax Rules

1. The storage area of identifier-1 must be different from the record area of file-name-1.
2. When the USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not written, the AT END or INVALID KEY must be written. In this compiler, however, both the USE AFTER STANDARD EXCEPTION procedure and the AT END phrase can be omitted.
3. The data item defined in the CONSTANT SECTION must not be specified as identifier-1.

General Rules

1. The file assigned to file-name-1 must be opened in input mode or I-O mode before the READ statement is executed.
2. When the READ statement is executed, the value of I-O status of file-name-1 is updated.
3. The NEXT phrase can be omitted. The NEXT phrase will in no way affect the READ statement execution.

4. When the READ statement execution starts, the file is checked for existence of the next valid record. If one exists, the record becomes usable in the record area related to file-name-1. If there is no valid record, the file at end condition occurs.

5. The INTO phrase is written in conditions as follows:

   a. Only one record description entry of file-name-1 is written.

   b. All record-names and identifier-1 related to file-name-1 are group items or alphanumeric data items.

6. When the INTO phrase is written, the processing will be as follows:

   a. The same READ statement without the INTO phrase is executed.

   b. When the READ statement has been executed successfully in processing (a), the current record is transferred from the record area of file-name-1 to identifier-1 according to the MOVE statement rule without the CORRESPONDING phrase. The read record becomes usable both in the record area of file-name-1 and the data item of identifier-1.

   The current record size is determined in accordance with the rules of the RECORD phrase in file-name-1. When the RECORD VARYING phrase is written in the file description entry of file-name-1, the group items are transferred in the processing (b). The subscript added to identifier-1 is evaluated before processing (b), which is after processing (a). If the processing (a) failed, the implicit MOVE statement is not executed.

7. When the file at end condition occurs during the READ statement execution, the READ statement execution fails. After the value indicating the file at end condition is set in the I-O status of file-name-1, the control transfers according to the rules shown in the section titled "AT END phrase."

8. When an exceptional condition other than a file at end condition occurs during the READ statement execution, the control transfers according to the rules shown in the section titled "AT END phrase," after the file position indicator and the I-O status of file-name-1 are set.

9. If no file at end condition or other exceptional condition occurs during the READ statement execution, the processing will be as follows:

    a. The file position indicator and the I-O status of file-name-1 are set.

    b. The record which has been read becomes usable in the record area.

    c. The control transfers according to the rules shown in the section titled "AT END phrase."

10. If the READ statement execution failed, contents of the record area of file-name-1 are not defined.

11. When the number of record character positions of a read record is less than the minimum record size specified in the record description entry of file-name-1, file-name-1 record area contents that exceed the number of record character positions will not be defined. The number of record character positions of a read record may be more than the minimum record size specified in the record description entry of file-name-1. In this case, the read record is truncated on the right in accordance with the maximum record size and set in the record area of file-name-1. The READ statement can be executed successfully in either case. The value indicating the record length conflict is set in the I-O status of file-name-1.

12. The END-READ phrase partitions the range of the READ statement.

RELEASE Statement (Sort-merge)

The RELEASE statement delivers a record to the first step of a sorting operation.

[Format]

<u>RELEASE</u> record-name-1 [<u>FROM</u> identifier-1]

Syntax Rules

1. The RELEASE statement can be written only in the input procedure specified by the SORT statement for the file assigned to record-name-1.

2. record-name-1 must be defined in the record description entry related to the file description entry for sort-merge.

3. record-name-1 can be qualified by file-name.

4. When the data item is specified to identifier-1, the storage area of record-name-1 must be different from that of identifier-1. When the function-identifier-1 is specified to identifier-1, identifier-1 must be an alphanumeric function.

General Rules

1. The RELEASE statement delivers a record assigned to record-name-1 to the first step of sorting operation.
2. When the RELEASE statement has been executed, the record assigned to record-name-1 becomes usable in the record area. However, when the file related to record-name-1 is specified in the SAME RECORD AREA clause in the I-O-CONTROL paragraph, the record assigned to record-name-1 will not be usable in the record area. The record is usable as an output record of another file specified in the same SAME RECORD AREA clause.
3. Executing the RELEASE statement with FROM phrase produces the same results as executing two statements in the following sequence:

a. MOVE identifier-1 TO record-name-1

b. Same RELEASE statement without FROM phrase

RETURN Statement (Sort-merge)

The RETURN statement receives the sorted or merged record from the last step of a sort or merge operation.

[Format]

```
    RETURN file-name-1 RECORD [INTO identifier-1]
 AT END imperative-statement-1
 [NOT AT END imperative-statement-2]
 [END-RETURN]
```

Syntax Rules

1. The RETURN statement can be written only in the output procedure specified by the SORT or MERGE statement for the file assigned to file-name-1.
2. file-name-1 must be defined in the file description entry for sort-merge.
3. The storage area of identifier-1 must be different from that of file-name-1.

General Rules

1. When the RETURN statement has been executed, the next record determined by the row of keys specified in the SORT or MERGE statement is given from the file assigned to file-name-1. If there is no following record, the file at end condition occurs.
2. Records defined in the record description entry of file-name-1 share the same storage area in accordance with redefinition rules.

3. The INTO phrase is written in the following conditions:
   a. Only one record description entry of file-name-1 is written.
   b. All record-names and identifier-1 related to file-name-1 are group items or alphanumeric data items.
4. When the INTO phrase is written, the processing will be as follows:
   a. The same RETURN statement without the INTO phrase is executed.
   b. When the RETURN statement has been executed successfully in processing (a), the current record is transferred from the record area of file-name-1 to identifier-1 according to the MOVE statement rule without the CORRESPONDING phrase. When the RECORD VARYING phrase has been written in the file description entry of file-name-1, group items are posted in processing (b). The subscript added to identifier-1 is evaluated before processing (b), which after processing (a). If processing (a) failed, the implicit MOVE statement is not executed.
5. When the file at end condition occurs, RETURN statement execution fails. After control has been transferred to imperative-statement-1 and that statement has been executed, the control transfers to the end of the RETURN statement. But, when a procedure branching statement to cause control explicit transfer in the imperative statement or the conditional statement has been executed, control transfers according to the statement rule. After execution of imperative-statement-1, the RETURN statement cannot be executed as a part of the output procedure during execution.

6. If no file at end condition occurs during the RETURN statement execution, the processing will be as follows:

   a. The record which has been given becomes usable in the record area of file-name-1. When the INTO phrase has been written, the record is transferred from the record area of file-name-1 to the record area of identifier-1.

   b. If the NOT AT END phrase has been written, the control transfers to imperative-statement-2. After the execution of imperative-statement-2, the control transfers to the end of the RETURN statement. However, when the procedure branching statement to cause control explicit transfer in the imperative statement or the conditional statement has been executed in imperative-statement-2, the control transfers according to the statement rules. When the NOT AT END phrase has been omitted, the control transfers to the end of the RETURN statement.

7. The END-RETURN phrase partitions the range of the RETURN statement.

REWRITE Statement (Sequential File, Relative File, Indexed File)

The REWRITE statement logically rewrites a record in a mass storage file.

[Format 1] Records are rewritten in the order that they are read. (Sequential file, relative file)

REWRITE record-name-1 [FROM identifier-1]
[END-REWRITE]

[Format 2] The specified record is rewritten. (Relative file, indexed file)

REWRITE record-name-1
[FROM identifier-1]
[INVALID KEY imperative-statement-1]
[NOT INVALID KEY imperative-statement-2]
[END-REWRITE]

Syntax Rules

Rules Common to Files

1. The record-name-1 must be defined in the record description entry related to the file description entry for sort-merge. record-name-1 can be qualified by file-name.
2. When the data item is specified to identifier-1, the storage area of record-name-1 must be different from that of identifier-1. When the function identifier-1 is specified to identifier-1, identifier-1 must be an alphanumeric function.

Rules for Relative Files

1. To specify the file in sequential access in record-name-1, the REWRITE statement must be written in format 1.
2. To specify the file in random or dynamic access method in record-name-1, the REWRITE statement must be written in format 2. When the USE AFTER STANDARD EXCEPTION procedure related to record-name-1 is not written, the INVALID KEY must be written. In this compiler, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

Rules for Indexed Files

1. The file related to record-name-1 must not be a file in random access method, in which the RECORD KEY phrase with the DUPLICATES phrase has been specified.
2. When the USE AFTER STANDARD EXCEPTION procedure related to record-name-1 is not written, the INVALID KEY must be written. In this compiler, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

General Rules

Rules Common to Files

1. The file related to record-name-1 must be a mass storage file.
2. The file related to record-name-1 must be opened in I-O mode before executing the REWRITE statement.

3. When the REWRITE statement has been executed successfully, the record assigned to record-name-1 becomes usable in the record area. However, when the file related to record-name-1 is specified in the SAME RECORD AREA phrase in the I-O-CONTROL paragraph, the record assigned to record-name-1 will not be usable by the REWRITE statement. The record can be used in the program.

4. The result of executing the REWRITE statement with FROM phrase is the same as the result of executing two statements in following sequence:

   a. MOVE identifier-1 TO record-name-1

   b. Same REWRITE statement without FROM phrase

5. After the REWRITE statement has been executed, the information of record-name-1 area cannot be used except in cases in which the file related to record-name-1 has been specified in the SAME RECORD AREA clause. However, information in identifier-1 area is usable.

6. Even if the REWRITE statement has been executed, the file position indicator is not changed.

7. When the REWRITE statement is executed, the value of I-O status of the file related to record-name-1 is updated.

8. The record is delivered to the I-O-CONTROL system by executing the REWRITE statement.

9. When an attempt to rewrite the record locked by another file connector is made, the REWRITE statement execution fails. Then, the value indicating the record lock is set to the I-O status related to record-name-1.

10. When AUTOMATIC has been specified in the LOCK MODE phrase in the file related to record-name-1, the existing record is released from being locked if the REWRITE statement has been executed successfully.

11. The END-REWRITE phrase partitions the range of the REWRITE statement.

Rules for Sequential Files

1. The READ statement must be executed for the file related to record-name-1 before executing the REWRITE statement. If the REWRITE statement is executed after the READ statement has been executed successfully, the record which has been read by the READ statement is rewritten logically.
2. If the number of character positions of the record assigned to record-name-1 is different from the length of the record to be rewritten, the REWRITE statement execution fails and no record will be rewritten. The contents of the record area of record-name-1 are not changed. In that case, the following processing sequence will be performed:
    a. The value indicating the cause of status is set in the I-O status of the file related to record-name-1.
    b. When the USE AFTER STANDARD EXCEPTION procedure of the file related to record-name-1 has been written, the procedure is synthesized. After the execution of the procedure, the control is delivered according to the rule for the USE statement. If the FILE STATUS phrase is specified in the file related to record-name-1 when the STANDARD EXCEPTION procedure has not been written, the control transfers to the end of the REWRITE statement. Processing without the USE AFTER STANDARD EXCEPTION procedure and FILE STATUS clause is undefined.

Rules for Relative Files

1. For the REWRITE statement (in format 1) for the file in sequential access method, the READ statement must be executed for the file related to record-name-1 before executing the REWRITE statement. If the READ statement has been executed successfully, the record which has been read by the READ statement is rewritten by executing the REWRITE statement.

2. If the REWRITE statement (in format 2) has been executed for the file by random or dynamic access method, the record containing the same relative record number as the value of the relative key item is rewritten. If no record in the file has the same relative record number as the value of the relative key item, the invalid key condition occurs.

Rules for Indexed Files

1. The last I-O statement to be executed before executing the REWRITE statement in the following cases must be the READ statement for the file related to record-name-1. If the REWRITE statement is executed when the READ statement has been executed successfully, the record which has been read by the READ statement is rewritten.

   a. REWRITE statement for the file in sequential access method

   b. REWRITE statement for the file in which the RECORD KEY clause with the DUPLICATES phrase has been specified

   c. REWRITE statement for the file in which the file position indicator has been determined by the START statement with WITH REVERSED phrase

2. In the REWRITE statement for the file in sequential access method, the record to be rewritten is specified by the value of the prime record key. To execute the REWRITE statement, the value of the prime record key must be equal to the value of the prime record key of the record which has been read last from this file.

3. In the REWRITE statement for a file in which the RECORD KEY clause with the DUPLICATES phrase has been specified or for a file in which the file position indicator has been determined by the START statement in the WITH REVERSED phrase, the record to be rewritten is specified by the value of the prime record key. To execute the REWRITE statement, the prime record key must be equal to the prime record key of the last record read from this file.

4. For the file in random access or dynamic access method, the record to be rewritten is specified by the value of the prime record key.

5. When the ALTERNATE RECORD KEY clause has been specified in the file related to record-name-1, the REWRITE statement is executed depending on the value of the alternate record key as follows:

   a. If the value of the alternate key does not change, the order of searching the record does not change even though the key is a key of reference.

   b. If the value of the alternate key changes, the order of searching the record may change if the key is a key of reference. When multiple key values are allowed, the record is logically located at the end of the group of records containing the same alternate record key as the specified alternate record key.

6. The invalid key condition occurs in the following cases:

    a. In the REWRITE statement to satisfy the following condition, when the value of the prime record key in the record to be rewritten is not equal to the value of the prime record key in the record which has been read last from the same file, the invalid key condition occurs.

       REWRITE statement for the file in sequential access method

       REWRITE statement for the file in which the RECORD KEY clause with the DUPLICATES phrase has been specified

       REWRITE statement for the file in which the file position indicator has been determined by the START statement in the REVERSED ORDER phrase

    b. In the REWRITE statement for the file in dynamic or random access method, when no record in the file has the same prime record key as the record to be rewritten, the invalid key condition occurs.

    c. In the REWRITE statement for the file in which the ALTERNATE RECORD KEY clause has been specified without the DUPLICATES, when no record in the file has the same alternate record key as the record to be rewritten, the invalid key condition occurs.

Rule Common to Relative Files and Indexed Files

1. The number of character positions of the record assigned to record-1 need not be the same as the length of the record to be rewritten.

2. The number of character positions of the record assigned to record-1 must not be more than the maximum record length of the file related to record-name-1. In that case, the REWRITE statement execution fails and the record is not rewritten. The contents of the record area in the file related to record-name-1 are not changed. The value indicating the cause of status is set in the I-O status of the file related to record-name-1.

3. When the invalid key condition occurs during the REWRITE statement execution, the REWRITE statement execution fails and the record is not rewritten. The control transfers according to the rules shown in the section titled “INVALID KEY phrase,” after the value indicating the invalid key condition is set in the I-O status of record-name-1.

4. If an exceptional condition other than an invalid key condition occurs during the REWRITE statement execution, the control transfers according to the rules shown in the section titled “INVALID KEY phrase,” after the file position indicator and the I-O status of record-name-1 are set.

5. When no invalid key condition or other exceptional condition occurs during the READ statement execution, the following processing sequence will be performed:

    a. The I-O status of the file related to record-name-1 is set.

    b. The record which has been read by the READ statement is rewritten.

    c. The control transfers according to the rules shown in the section titled “INVALID KEY phrase.”

SEARCH Statement (Nucleus)

The SEARCH statement searches table elements.

[Format 1] Table elements are searched in sequence.

$$\underline{\text{SEARCH}}\ \text{identifier-1}\left[\underline{\text{VARYING}}\left\{\begin{array}{l}\text{identifier-2}\\ \text{index-name-1}\end{array}\right\}\right]$$

$$[\text{AT}\ \underline{\text{END}}\ \text{imperative-statement-1}]$$

$$\left\{\underline{\text{WHEN}}\ \text{condition-1}\left\{\begin{array}{l}\text{imperative-statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}\right\}\ \ldots$$

$$[\underline{\text{END-SEARCH}}]$$

[Format 2] Table elements aligned in ascending or descending order are searched.

$$\underline{\text{SEARCH}}\ \underline{\text{ALL}}\ \text{identifier-1}\ [\text{AT}\ \underline{\text{END}}\ \text{imperative-statement-1}]$$

$$\underline{\text{WHEN}}\left\{\begin{array}{l}\text{data-name-1}\left\{\begin{array}{l}\text{IS}\ \underline{\text{EQUAL}}\ \text{TO}\\ \text{IS} =\end{array}\right\}\left\{\begin{array}{l}\text{identifier-3}\\ \text{literal-1}\\ \text{arithmetic-expression-1}\end{array}\right\}\\ \text{condition-name-1}\end{array}\right\}$$

$$\left[\underline{\text{AND}}\left\{\begin{array}{l}\text{data-name-2}\left\{\begin{array}{l}\text{IS}\ \underline{\text{EQUAL}}\ \text{TO}\\ \text{IS} =\end{array}\right\}\left\{\begin{array}{l}\text{identifier-4}\\ \text{literal-2}\\ \text{arithmetic-expression-2}\end{array}\right\}\\ \text{condition-name-2}\end{array}\right\}\right]\ \ldots$$

$$\left\{\begin{array}{l}\text{imperative-statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{array}\right\}$$

$$[\underline{\text{END-SEARCH}}]$$

Note: The symbol = is required, but not underlined not to avoid confusion with other symbols.

Syntax Rules

Rules for Format 1

1. No subscript or reference modifier can be added to identifier-1. identifier-1 must be a data item in which the OCCURS clause with INDEXED BY phrase has been specified.
2. The identifier-2 must be an index data item or integer item. The first or only index-name among index-names written in items in which the OCCURS clause with INDEXED BY phrase has been specified in identifier-1 must not be used for adding the subscript to identifier-2.

Rules for Format 2

1. The identifier-1 must be a data item in which the OCCURS clause with INDEXED BY phrase and the KEY IS phrase are specified. No subscript or reference modifier can be added to identifier-1.
2. The data-name-1 and data-name-2 must be data-names written in the KEY IS phrase of the OCCURS clause in the identifier-1.
3. Conditional variables for condition-name-1 and condition-name-2 must be data-names written in the KEY IS phrase of the OCCURS clause in identifier-1. The THRU phrase must not be written in the VALUE clause of the data description entry in condition-name-1 and condition-name-2.
4. To add a subscript to data-name-1, data-name-2, condition-name-1, or condition-name-2, at least one subscript must be the first index-name in the INDEXED BY phrase in the OCCURS clause of identifier-1.

5. The data-name written in the KEY IS phrase in the OCCURS clause of identifier-1 cannot be specified as identifier-3, identifier-4, the identifier in arithmetic expression-1, or the identifier in arithmetic expression-2. To add a subscript to either of these identifiers, the subscript must not be the first index-name in the INDEXED BY phrase in the OCCURS clause of identifier-1.

6. When two or more data-names have been written in the KEY IS phrase of the OCCURS clause in identifier-1, all data-names in the KEY IS phrase beginning from the first data-name to the nth data-name (not necessarily) or all conditions related to data-names must be written in data-name or condition name in the WHEN phrase. data-name and condition-name can be written in one WHEN phrase.

7. data-name-1, data-name-2, identifier-3, identifier-4, literal-1 and literal-2 cannot be Boolean items.

Rules Common to Format 1 and Format 2

The NEXT SENTENCE phrase and the END-SEARCH phrase are mutually exclusive.

Example of SEARCH Statement in Format 2

When the following table is defined in the data division, SEARCH statements (a) and (b) can be written in the procedure division.

```
01 TBL.
   02  Y  OCCURS  50  TIMES
          ASCENDING  KEY  IS  K1, K2, K3, K4
          INDEXED BY  IX1.
      03  K1  PIC  9.
        88  T1  VALUE  3.
      03  K2  PIC  9.
        88  T2  VALUE  2.
      03  K3  PIC  9.
        88  T3  VALUE  1.
      03  K4  PIC  9.
77 X  PIC  9.
```

```
(a)  SEARCH  ALL  Y  WHEN  K1(IX1) = 3
                       AND  K2(IX1) = 2
                       AND  K3(IX1) = 1.
(a)  SEARCH  ALL  Y  WHEN  T1(IX1)
                       AND  T2(IX1)
                       AND  T3(IX1)
                       AND  K4(IX1) = X.
```

General Rules

Rules for Format 1

1. The SEARCH statement in format 1 searches table elements in identifier-1 table in sequence. These elements are indicated by index-names for table lookup. When a table element satisfying condition-1 is found, the processing for imperative-statement-2 corresponding to condition-1 is performed.

2. To search table elements, the following index-names are used. These index-names are called “index-names for table lookup.”

   a. index-name which has been written first in the INDEXED BY phrase in the OCCURS clause of identifier-1 is used in the following cases:

   - The VARYING phrase has been omitted.

   - The index-name in the table which is not identifier-1 table has been written in the VARYING phrase.

   - The identifier-2 has been written in the VARYING phrase.

   b. When index-name which has been written first in the INDEXED BY phrase in the OCCURS clause of identifier-1 has been written, index-name is used.

3. Table elements are searched according to the following procedure:

a. When the value of index-name for table lookup is greater than the value corresponding to the highest occurrence number of identifier-1, table elements are not searched. In that case, the control transfers to imperative-statement-1 if the AT END phrase has been written. If the AT END phrase has been omitted, the control transfers to the end of the SEARCH statement.

b. When the value of index-name for table lookup is less than the value corresponding to the highest occurrence number of identifier-1, the processing proceeds as follows:

    i The table element indicated by the value of index-name for table lookup is tested to see whether it satisfies condition-1. The test is performed in the order of elements to which the WHEN phrase has been written.

    ii If no condition has been satisfied, the value of index-name for lookup is increased. Then, the processing 1) repeats.

    iii When the new value of index-name for lookup is not within the range of occurrence numbers in the table, the SEARCH statement processing terminates in the same way as the processing described in (a).

    iv When one of the conditions is satisfied, the table element test terminates. When imperative-statement-2 has been written, the control transfers to imperative-statement-2. When the NEXT SENTENCE has been written, the control transfers to the next execution sentence. At that time, the value of index-name for lookup corresponds to the occurrence number when the condition has been satisfied.

4. Values of index-name for lookup, identifier-2 and index-name-1, are increased according to the following rules:

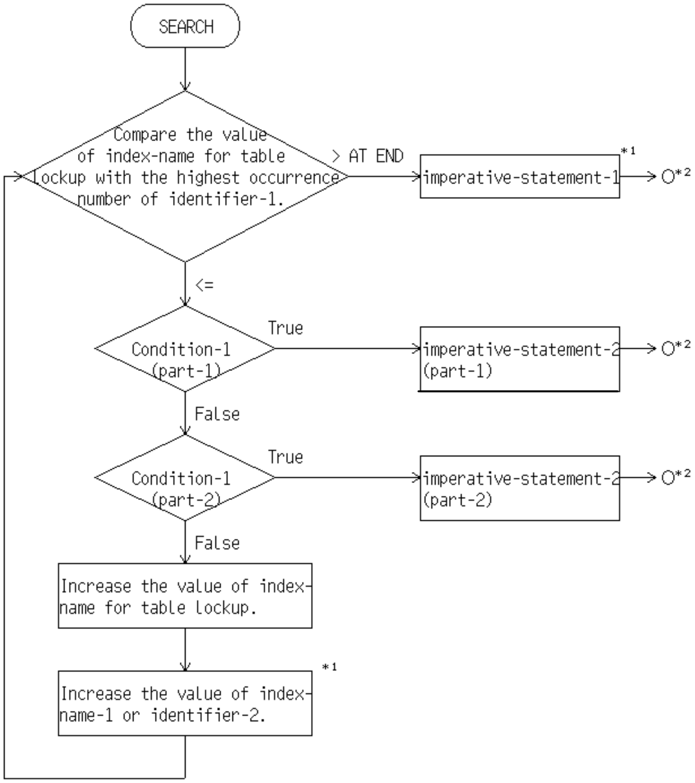
    a. When the VARYING phrase has been omitted, or index-name written in the INDEXED BY phrase in the OCCURS clause of identifier-1 has been written in the VARYING phrase, only index-name for table lookup is increased.

    b. When a index-name in a table other than identifier-1 table has been written in index-name-1 of the VARYING phrase, the value of index-name-1 increases as much as the value corresponding to the occurrence number indicated by the increment each time index-name for table lookup increases.

    c. When the index data item has been written in the VARYING phrase, the value of the index data item increases as much as the increment each time index-name for table lookup increases.

    d. When the literal item has been written in the VARYING phrase, the value of the literal item increases by one.

5. When two WHEN phrases have been written, the table element test is performed according to the following flow chart:


SEARCH
Compare the value of index-name for table lockup with the highest occurrence number of identifier-1.
> AT END
imperative-statement-1
*1
O*2
<=
Condition-1 (part-1)
True
imperative-statement-2 (part-1)
O*2
False
Condition-1 (part-2)
True
imperative-statement-2 (part-2)
O*2
False
Increase the value of index-name for table lockup.
*1
Increase the value of index-name-1 or identifier-2.

*1 This operation is performed only when the SEARCH statement has been written.

*2 The control transfers at the end of the SEARCH statement after the execution of the imperative statement. However, when the procedure branching statement to cause control explicit transfer has been written in the imperative statement, the control transfers according to the statement rules.

Rules for Format 2

1. The SEARCH statement non-sequentially searches table elements to satisfy the WHEN phrase condition in identifier-1 table. When the table element to satisfy the condition is found, imperative-statement-2 is executed.

2. The index-name which has been written first in the INDEXED BY phrase in the OCCURS clause of identifier-1 is used for searching the table element. index-name is called an “index-name for table lookup.” The value of only the index name for lookup is changed.

3. The execution result of the SEARCH statement in format 2 is defined only when all the following condition is satisfied.

    a. Data in identifier-1 table is aligned in ascending or descending order according to the description of the KEY IS phrase in the OCCURS clause of identifier-1.

    b. Values of data-name-1 and data-name-2, or values of condition-name-1 and condition-name-2 are values which enable table elements to be identified at once.

4. Regardless of values of index-names for table lookup, table elements are searched according to the following procedure:

    a. Table elements to satisfy all the condition written in the WHEN phrase are searched.

    b. If no table element to satisfy all the conditions written in the WHEN phrase is found, the control transfers to imperative-statement-1 when the AT END phrase has been written. When the AT END phrase has been omitted, the control transfers at the end of the SEARCH statement. The value of index-name for table lookup is not defined.

c. If an table element to satisfy all the condition written in the WHEN phrase is found, the control transfers to imperative-statement-2 when the imperative statement has been written. When the NEXT SENTENCE has been written, the control transfers to the next execution sentence. The value that corresponding to the occurrence number of the table element satisfying the condition is set to index-name for table lookup.

Rules Common to Format 1 and Format 2

1. The end of the SEARCH statement is specified by any of the following methods:

   a. END-SEARCH phrase

   b. Separator period

   c. ELSE phrase corresponding to IF statement or END-IF phrase when the SEARCH statement has been written in the imperative statement of the IF statement

   See “Range of statement” in the section titled “Configuration of Procedure Division,” for the range of the IF statement.

2. After execution of imperative-statement-1 or imperative-statement-2, the control transfers to the end of the SEARCH statement. However, when the procedure branching statement to cause control explicit transfer has been written in the imperative statement, the control transfers according to the statement rules.

3. The END-SEARCH phrase partitions the range of the SEARCH statement.

SET Statement (Nucleus)

The SET statement sets the table element index, the external switch status, and sets the condition variable value.

[Format 1] Table element indexes are set.

$$\underline{\text{SET}} \left\{ \begin{array}{l} \text{index-name-1} \\ \text{identifier-1} \end{array} \right\} \ldots \ \underline{\text{TO}} \left\{ \begin{array}{l} \text{index-name-2} \\ \text{identifier-2} \\ \text{integer-1} \end{array} \right\}$$

[Format 2] The value of index-name is increased or decreased.

$$\underline{\text{SET}} \quad \{\text{index-name-3}\}\ldots \left\{ \begin{array}{l} \underline{\text{UP}}\ \underline{\text{BY}} \\ \underline{\text{DOWN}}\ \underline{\text{BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{integer-2} \end{array} \right\}$$

[Format 3] The external switch status is set.

$$\underline{\text{SET}} \left\{ \{\text{mnemonic -name-1}\}\ldots \ \underline{\text{TO}} \left\{ \begin{array}{l} \underline{\text{ON}} \\ \underline{\text{OFF}} \end{array} \right\} \right\} \ldots$$

[Format 4] The value of the condition variable is set.

$$\underline{\text{SET}} \quad \{\text{condition -1}\}\ldots \ \underline{\text{TO}}\ \underline{\text{TRUE}}$$

Syntax Rules

1. The identifier-1 and identifier-2 must be index data items or integer items.
2. The identifier-3 must be an integer item.

3. A non-zero positive integer can be specified to integer-1. A non-zero positive integer or negative integer can be specified to integer-2.

4. The mnemonic-name-1 must be related to SWITCH-0 to SWITCH-7 or SWITCH-1 to SWITCH-8 in the special-name paragraph of the environment division.

General Rules

Rules for Format 1

1. The SET statement in format 1 sets values corresponding to the occurrence numbers indicated by operands after the TO execution to each operand before the TO execution. The operand after the TO execution is called “sending operand” and the operand before the TO execution is called “receiving operand.”

2. Combinations of sending operands and receiving operands in the SET statement are listed in the table below.

Sending Operand		**Receiving Operand**		
		index-name-1	**identifier-1**	
			Index Data Item	**Integer Item**
index-name-2		o (5)(a)	o (5)(b)	o (5)(c)
integer-2	index data item	o (5)(a)	o (5)(b)	-
	integer item	o (5)(a)	-	-
integer-1		o (5)(a)	-	-

o: Combined -: Not combined (n): General rule number

3. When index-name-1 is written, the value of the operand after the TO execution must correspond to the occurrence number of the table to which index-name has been related.

4. When index-name-2 is written, the value of index-name-2 must correspond to the occurrence number of the table to which index-name has been related.

5. Index values are set according to the following rules:

   a. When index-name-1 has been written, the value of occurrence number corresponding to index-name-2, identifier-2 or integer-1 is set in index-name-1. When the index data item has been specified in identifier-2, or when the index-name related to the same table as index-name-1 has been specified, the value of the occurrence number is not converted.

   b. When the index data item has been specified in identifier-1, the same value as the contents of index-name-2 or identifier-2 is set without being converted to identifier-1.

   c. When the integer item is specified to identifier-1, the value of the occurrence number corresponding to the value of index-name-2 is set in identifier-1.

6. When two or more receiving operands have been written, index setup operation repeats. In the repetitive operation, the value before the SET statement execution is used as the value for index-name-2 and identifier-2. When subscripts have been added to identifier-1, the subscripts are evaluated in sequence one by one at every repetitive operation.

Rules for Format 2

1. The SET statement in format 2 does either of the following:

   a. Increases the value of each operand before the UP BY execution by as much as the value corresponding to the occurrence number indicated by the operand after the UP BY execution.

   b. Decreases the value of each operand before the DOWN BY execution by as much as the value corresponding to the occurrence number indicated by the operand after the DOWN BY execution.

2. The value of index-name-3 before and after the SET statement execution must correspond to the occurrence number of the table to which index-name-3 is related.
3. When more than one index-name-3 has been written, values of index-names are repeatedly increased or decreased. In the repetitive operation, the value before the SET statement execution is used as the value for identifier-3.

Rules for Format 3

1. The SET statement in format 3 sets the external switch related to mnemonic-name-1 to ON or OFF. This operation repeats in the written sequence of mnemonic-name-1.
2. The status of the external switch related to mnemonic-name-1 is set as follows:
   a. When the ON phrase has been written, the external switch status is set so that the result of the truth value when the condition name related the external switch is evaluated should be ON.

   b. When the OFF phrase has been written, the external switch status is set so that the result of the truth value when the condition name related the external switch is evaluated should be OFF.

Rules for Format 4

1. The SET statement in format 4 sets the value of the VALUE clause written in the condition name description entry of condition-name-1 to the condition variable related to condition-name-1. This operation repeats in the written sequence of condition-name-1.
2. When two or more literal have been written in the VALUE clause of condition-name-1, the first literal value written in the VALUE clause is set as the condition variable.

SORT Statement (Sort-merge)

The SORT statement sorts records in the order determined by the key item values.

[Format]

```
SORT  file-name-1
    { ON { ASCENDING  } KEY {data-name-1} ... } ...
         { DESCENDING }
    [ WITH DUPLICATES IN ORDER ]
    [ COLLATING SEQUENCE IS alphabetic-name-1 ]
    { INPUT PROCEDURE IS                                                 }
    {                       [ { THRU    }                              ] }
    {     procedure-name-1  [ { THROUGH }  procedure-name-2            ] }
    { USING {file-name-2} ...                                            }
    { OUTPUT PROCEDURE IS                                                }
    {                       [ { THRU    }                              ] }
    {     procedure-name-3  [ { THROUGH }  procedure-name-4            ] }
    { GIVING {file-name-3}                                               }
```

Syntax Rules

1. The SORT statement can be written in any part in the procedure division besides the declarative part.
2. The file-name-1 must be defined in the file description entry for sort-merge in the data division.
3. The following rules apply to file-name-2:
   a. When the record format of file-name-1 is variable, the record length of file-name-2 must be longer than the minimum record length of file-name-1 and shorter than the maximum record length.

   b. When the record format of file-name-1 is fixed, the record length of file-name-2 must be shorter than the length of file-name-1.
4. The following rules apply to data-name-1:
   a. The data-name-1 must be a data item defined in the record description entry of file-name-1.

   b. The data-name-1 can be qualified.

   c. The group item followed by the variable occurrence data item must not be specified as data-name-1.

   d. When two or more record description entries have been written in file-name-1, the data item written in one record description entry must be specified in data-name-1 row. The character position which is the same as the data item of data-name-1 in the record description entry is regarded as the key of all records in the file.

   e. A data item which specifies the OCCURS clause or a data item following the data item which specifies the OCCURS clause cannot be specified as data-name-1.

   f. When the record format of file-name-1 is variable, the data item of data-name-1 must be included in the range of the minimum record size of file-name-1.

   g. The data-name-1 cannot be a Boolean item.

   h. See Appendix B, "System determination limitations," for the maximum number of times data-name-1 can be specified.

5. The file-name-2 and file-name-3 must be defined in the file description entry which is not the file for sort-merge in the data division.

6. Files on the same multiple file reels can be specified to file-name-2 and file-name-3.

7. Some groups of file-name-1, file-name-2, and file-name-3 cannot be specified in one SAME AREA clause or one SAME SORT/SORT-MERGE AREA clause. Some groups of file-name-3 cannot be specified in one SAME clause, either.

8. THROUGH is synonymous with THRU.

9. The following rules apply when specifying an indexed file to file-name-3:

   a. The first KEY phrase must be the ASCENDING KEY phrase.

   b. The character position in the record of the data item specified to the first data-name-1 must be the same as that in the record of the data record related to the prime record key in the indexed file. Both data items must have the same length.

10. The following rules apply to file-name-3:

   a. When the record format of file-name-3 is variable, the record length of file-name-1 must be longer than the minimum record length of file-name-3 and shorter than the maximum record length.

b. When the record format of file-name-3 is fixed, the record length of file-name-1 must be shorter than the length of file-name-3.

11. The alphabet-name-1 must not be an alphabet-name corresponding to the function name in the ALPHABET clause in the special-name paragraph.

12. The CHARACTER TYPE clause or PRINTING POSITION clause must not be written in the record description entry of file-name-1 to file-name-3.

13. file-name-2 and file-name-3 must be files as follows:

    a. Sequential file in mass storage unit or floppy disk drive

    b. Relative file in sequential access method

    c. Indexed file in sequential access method

14. See Appendix B, “System determination limitations,” for the maximum number of times file-name-2 can be specified.

General Rules

All Rules for SORT Statement

1. When the SORT statement has been executed, the processing is performed in the following sequence:

    a. Records are delivered to the sorting operation. This processing is called “first step of sorting operation.”

    The input procedure is executed when the INPUT PROCEDURE has been written. “INPUT PROCEDURE” indicates the procedure written in procedure-name-1 or the procedure written in the procedures from procedure-name-1 to procedure-name-2. When the RELEASE statement has been executed to the record of file-name-1 in the input procedure, the record is delivered to the sorting operation.

When the USING phrase has been written, the READ statement is executed to file-name-2 and the record read by the READ statement is delivered to the sorting operation. Before starting this step, the file assigned to file-name-2 should not be open. During operation of this step, the file assigned to file-name-2 is automatically opened and closed.

b. Records in file-name-1 are sorted in the order specified in the KEY phrase. This processing is called “sorting operation.” file-name-2 and file-name-3 are not processed in this step.

c. Records sorted in the following processing become usable as records of file-name-1. This operation is called “last step of sorting operation.”

The output procedure is executed when the OUTPUT PROCEDURE has been written. The “OUTPUT PROCEDURE” indicates the procedure written in procedure-name-3 or the procedure written in the procedures from procedure-name-3 to procedure-name-4. When the RETURN statement has been executed to the record of file-name-1 in the input procedure, the record becomes usable.

When the GIVING phrase has been written, sorted records become usable as records of file-name-1 in the output procedure and are written in the file assigned to file-name-3. Before starting this step, the file assigned to file-name-3 should not be open. During operation of this step, the file assigned to file-name-3 is automatically opened and closed.

2. The data item in data-name-1 is called a “key item.” To specify two or more key items, align key items from left to right in the order determined by the key intensity.

   a. When the ASCENDING phrase has been written, key items of the record delivered to the sorting operation are compared according to the comparison rules and aligned in the order starting from smallest record of the key item value to the largest one.

   b. When the DESCENDING phrase has been written, key items of the record delivered to the sorting operation are compared according to the comparison rule and aligned in the order starting from largest record of the key item value to the smallest one.

3. When comparison described in (2) finds two or more records containing the same contents of all key items, sorted records are given as follows:

   a. When the DUPLICATES phrase has been written:

      When the USING phrase has been written, records in each file are given in the order which the row of file-name-2 has been written. If two or more such records exist in one file assigned to file-name-2, records are given in the order that they have been read. When the INPUT PROCEDURE phrase has been written, records are given in the order that they have been delivered in the input procedure.

   b. When the DUPLICATES phrase has been omitted:

      The order to receive records is not defined.

4. When the rule for comparing characters is applied, the order of the key item values is determined according to the following rules:

    a. When the COLLATING SEQUENCE phrase has been written, the order is determined by the collating sequence characters corresponding to alphabetic-name-1.

    b. If the PROGRAM COLLATING SEQUENCE clause has been written in the computer paragraph for compiling in the identification division when the COLLATING SEQUENCE phrase has been omitted, the order is determined by the collating sequence characters specified in the clause. When the PROGRAM COLLATING SEQUENCE clause has been omitted, the order is determined by the collating sequence of native computer characters.

Rules for INPUT PROCEDURE Phrase

1. When the INPUT PROCEDURE phrase has been written, the control transfers to the input procedure before the sorting operation starts. After the last statement in the input procedure has been executed, the sorting operation starts.

2. The RELEASE statement for the record in file-name-1 is written in the input procedure. The RELEASE statement delivers one record to the first step of the sorting operation.

3. The statement to be executed in the input procedure is called a “range of input procedure.” The range of the input procedure is as follows:

   a. Statement written in the input procedure

   b. All statements to be executed by the CALL statement, EXIT statement, GO TO statement, and PERFORM statement written in the input procedure

   c. All statements in the declarative procedure to be executed by the statements written in the input procedure

4. The MERGE statement, RETURN statement, and SORT statement must not be executed in the range of the input procedure.

Rules for USING Phrase

1. When the USING phrase has been written, all records in the file assigned to file-name-2 are transferred to the file assigned to file-name-1.

2. For each file in file-name-2, the following processing sequence is performed:

   a. The initial processing is performed. This processing is the same as the execution of the OPEN statement specified with the INPUT phrase for the file assigned to file-name-2.

   b. Records in file-name-2 are delivered to the sorting operation. This processing is the same as though a READ statement with the NEXT phrase and the AT END phrase has been executed up to the end of records. When the relative file has been specified as file-name-2, the contents of relative key items after the SORT statement execution are not defined.

c. The termination processing is performed. This processing is the same as the execution of the CLOSE statement which specified only file-name-2. This processing is performed before the sorting operation starts.

During processing from (a) to (c), the USE AFTER STANDARD EXCEPTION procedure may be executed. During USE procedure, any statement to operate the file assigned to file-name-2 or statement to reference records of file-name-2 must not be executed.

3. If the record length of file-name-2 is shorter than the record length of file-name-1 when the record format of file-name-1 is fixed, the record is padded with blanks on the right when it is delivered to the file assigned to file-name-1.

Rules for OUTPUT PROCEDURE Phrase

1. When the OUTPUT PROCEDURE phrase has been written, the control transfers to the output procedure after the sorting operation terminates. After the last statement in the output procedure has been executed, the control transfers to the next SORT statement to be executed.
2. When the control transfers to the output procedure, records in file-name-1 are all sorted and ready to be given in order by the RETURN statement. The RETURN statement for the file assigned to file-name-1 is written in the output procedure. The RETURN statement receives one record from the last step of the sorting operation.

3. The statement to be executed in the output procedure is called a “range of output procedure.” The range of the output procedure is as follows:
    a. Statement written in the output procedure
    b. All statements to be executed by the CALL statement, EXIT statement, GO TO statement, and PERFORM statement written in the output procedure
    c. All statements in the declarative procedure to be executed by the statements written in the output procedure
4. The MERGE statement, RETURN statement, and SORT statement must not be executed in the range of the output procedure.

Rules for GIVING Phrase

1. When the GIVING phrase has been written, all records in the file assigned to file-name-1 are transferred to the file assigned to file-name-3 after termination of the sorting operation.
2. For each file in file-name-3, the following processing sequence is performed:
    a. The initial processing is performed. This processing is the same as the execution of the OPEN statement specified with the OUTPUT phrase for the file assigned to file-name-3.
    b. Records in file-name-1 are given from the sorting operation. This processing is the same as though a WRITE statement that specifies only record-name phrase has been executed up to the end of records.

When the relative file has been specified as file-name-3, the relative record numbers to be given are set in order starting from 1, 2, 3, etc. When the relative key item has been specified in the file assigned to file-name-3, the relative record number is set to the relative key item each time the record is given. The contents of relative key items after the SORT statement execution indicate the last record given to the file assigned to file-name-3.

c. The termination processing is performed. This processing is the same as the execution of a CLOSE statement which specifies only file-name-3.

During processing from (a) to (c), the USE AFTER STANDARD EXCEPTION procedure may be executed. During USE procedure, any statement to operate the file assigned to file-name-3 or statement to refer records of file-name-3 must not be executed.

3. When an attempt to write the record first over the area defined out of the file assigned to file-name-3 is made, the processing will be as follows:

   a. When the USE AFTER STANDARD EXCEPTION procedure related to file-name-3 has been written, the USE procedure is performed. After the control returns from the USE procedure, the termination processing is performed as if a CLOSE statement that specifies only file-name-3 had been executed.

   b. When the USE AFTER STANDARD EXCEPTION procedure related to file-name-3 has not been written, the termination processing is performed as if a CLOSE statement that specifies only file-name-3 had been executed.

4. If the record length of file-name-1 is shorter than the record length of file-name-3 when the record format of file-name-3 is fixed, the record is padded with blanks on the right when it is given to the file assigned to file-name-3.

START Statement (Relative File)

The START statement logically allocates a file for calling records sequentially.

[Format]

```
START file-name-1
 ┌          ┌                        ┐                ┐
 │          │ IS EQUAL TO            │                │
 │          │ IS =                   │                │
 │          │ IS GREATER THAN        │                │
 │          │ IS >                   │                │
 │   KEY    │ IS NOT LESS THAN       │  data-name-1   │
 │          │ IS NOT <               │                │
 │          │ IS GREATER THAN        │                │
 │          │      OR EQUAL TO       │                │
 │          │ IS >=                  │                │
 └          └                        ┘                ┘
 [INVALID KEY imperative-statement-1]
 [NOT INVALID KEY imperative-statement-2]
 [END-START]
```

Note: **Symbols =, >, <, and >= are required, but not underlined to avoid confusion with other symbols.**

Syntax Rules

1. The file-name-1 must be a file in serial access or dynamic access method.
2. The data-name-1 must be the data item (relative key item) written in the RELATIVE KEY phrase of the ACCESS MODE clause in the file control entry related to file-name-1. data-name-1 can be qualified.
3. When the USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not written, the INVALID KEY phrase must be written. In this compiler, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

General Rules

1. The file assigned to file-name-1 must be opened in input mode or I-O mode before executing the START statement.
2. When the KEY phrase has been omitted, the relation IS EQUAL TO is assumed to be specified.
3. Even if the START statement has been executed, the contents in the record area of file-name-1 are not changed.
4. The key related to the record in the file assigned to file-name-1 and the relative key item in file-name-1 are compared in accordance with the description in the KEY phrase. The rules for numeric comparison is applied. As a result of comparison, the following processing will be performed:

a. When a record to satisfy the comparison condition exists in the file assigned to file-name-1, the file position indicator is set in the first record satisfying the comparison condition.

b. When no record satisfying the comparison condition exists in the file assigned to file-name-1, the invalid key condition occurs and the START statement execution fails.

5. When the START statement has been executed, the value of I-O status of the file assigned to file-name-1 is updated.
6. When the file position indicator indicates that no optional file for input exists before executing the START statement, the invalid key condition occurs and the START statement execution fails.
7. When the START statement execution has failed, the value indicating that no valid record remains is set in the file position indicator.
8. If the invalid key condition occurs during the START statement execution, the control transfers according to the rule shown in the section titled “INVALID KEY phrase,” after the value indicating the invalid key condition is set in the I-O status of file-name-1.
9. If no other exceptional condition occurs during the START statement execution, the control transfers according to the rule shown in the section titled “INVALID KEY phrase,” after the value indicating the invalid key condition is set in the I-O status of file-name-1.
10. The END-START phrase partitions the range of the START statement.

11. When AUTOMATIC has been specified in the LOCK MODE phrase in the file related to file-name-1, the existing record is released from being locked if the START statement has been executed.

START Statement (Indexed File)

The START statement logically allocates the file for calling records sequentially.

[Format 1] Files are allocates logically to be called in the normal ascending sequence.

$$
\begin{array}{l}
\underline{\text{START}}\ \ \text{file-name-1}\\
\left[\ \text{KEY}\left\{\begin{array}{l}
\text{IS }\ \underline{\text{EQUAL}}\ \text{ TO}\\
\text{IS }\ =\\
\text{IS }\ \underline{\text{GREATER}}\ \text{ THAN}\\
\text{IS }\ >\\
\text{IS }\ \underline{\text{NOT}}\ \ \underline{\text{LESS}}\ \text{ THAN}\\
\text{IS }\ \underline{\text{NOT}}\ <\\
\text{IS }\ \underline{\text{GREATER}}\ \text{ THAN}\\
\qquad\ \underline{\text{OR}}\ \ \underline{\text{EQUAL}}\ \text{ TO}\\
\text{IS }\ >=
\end{array}\right\}\ \{\text{data-name-1}\}\ \ldots\right]\\
[\underline{\text{INVALID}}\ \text{KEY imperative-statement-1}]\\
[\underline{\text{NOT}}\ \underline{\text{INVALID}}\ \text{KEY imperative-statement-2}]\\
[\underline{\text{END-START}}]
\end{array}
$$

[Format 2] Files are allocates logically to be called in the descending sequence.

```
START  file-name-1
 ┌        ⎧ IS EQUAL TO                 ⎫                   ┐
 │        ⎪ IS =                        ⎪                   │
 │        ⎪ IS LESS THAN                ⎪                   │
 │        ⎪ IS <                        ⎪                   │
 │  KEY   ⎨ IS NOT GREATER THAN         ⎬ {data-name-1} ... │
 │        ⎪ IS NOT <                    ⎪                   │
 │        ⎪ IS LESS THAN                ⎪                   │
 │        ⎪       OR EQUAL TO           ⎪                   │
 └        ⎩ IS <=                       ⎭                   ┘
  WITH REVERSED  ORDER
 [INVALID KEY imperative-statement-1]
 [NOT INVALID KEY imperative-statement-2]
 [END-START]
```

[Format 3] Files are allocates logically to the record which is the beginning or and of the key of reference .

```
START  file-name-1
    FIRST  RECPRD
   [KEY  IS  {data-name-1} ...]
   [WITH REVERSED  ORDER]
   [INVALID KEY imperative-statement-1]
   [NOT INVALID KEY imperative-statement-2]
   [END-START]
```

Note: The symbols =, >, <, and >= are required, but not underlined to avoid confusion with other symbols.

Syntax Rules

1. file-name-1 must be a file in serial access or dynamic access method .
2. data-name-1 can be qualified.
3. When the USE AFTER STANDARD EXCEPTION procedure related to file-name-1 is not written, the INVALID KEY phrase must be written. In this compiler, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.
4. To write the KEY phrase, data-name-1 must be one of the following. However, it must be the item described in (a) if the KEY phrase is written is format 3.

   a. data-name-1 must be a data item (prime record key item) written in the RECORD KEY clause of the file control entry related to file-name-1 or data item (alternate record key item) written in the ALTERNATE RECORD KEY clause. When two or more data-names have been written in the RECORD KEY clause, however, the same number of data-names must be written in the ALTERNATE RECORD KEY clause in the order that they have been written. In the same way, when two or more data-names have been written in the ALTERNATE RECORD KEY clause, the same number of data-names must be written in the RECORD KEY clause in the order that they have been written.

a. data-name-1 must be a part of a row of data-names written in the RECORD KEY clause related to file-name-1. That means that data-names from the first to the n-1 or earlier among n data-names written in the RECORD KEY clause are specified with the same names and in the same order as they have been written in the RECORD KEY clause.

b. data-name-1 must be a part of a row of data-names written in the ALTERNATE RECORD KEY clause related to file-name-1. That means that data-names from the first to the n-1 or earlier among n data-names written in the ALTERNATE RECORD KEY clause are specified with the same names and in the same order as they have been written in the ALTERNATE RECORD KEY clause.

c. When only one data-name has been written in the RECORD KEY clause related to file-name-1, a data item which satisfies all the following conditions:

   data-name exists in the record related to file-name-1 and the character position at the left end equals to that of the prime record key item.

   The size of data-name is less than or equal to the size of the prime record key item.

   data-name is an alphanumeric data item, national data item, or group item.

   Only one data-name can be written in the row of data-name-1.

d. When only one data-name has been written in the ALTERNATE RECORD KEY clause related to file-name-1, a data item which satisfies all the following conditions:

   data-name exists in the record related to file-name-1 and the character position at the left end equals to that of the alternate record key item.

   The size of data-name is less than or equal to the size of the alternate record key item.

   data-name is an alphanumeric data item, national data item, or group item.

   Only one data-name can be written in the row of data-name-1.

General Rules

Rules Common to Format 1 to Format 3

1. The file assigned to file-name-1 must be opened in input mode or I-O mode before executing the START statement.
2. Even if the START statement has been executed, the contents in the record area of file-name-1 are not changed. Also, the contents of data item specified in DEPENDING ON phrase of RECORD clause related to file-name-1 are not changed.
3. When the START statement has been executed, the value of I-O status of the file of file-name-1 is updated.
4. When the file position indicator indicates that no optional file for input exists before executing the START statement, the invalid key condition occurs and the START statement execution fails.

5. When the START statement execution has failed, the value indicating that no more valid record exists is set in the file position indicator. The key of reference is not defined.

6. The key of reference is determined according to the following rules which are applied in the order described:

   a. When the KEY phrase has been omitted, the prime record key of file-name-1 is a key of reference.

   b. When all names in data-name-1 row of the KEY phrase are the same and the number of names is the same as in the RECORD KEY clause, the prime record key of file-name-1 is a key of reference.

   c. When all names in data-name-1 row of the KEY phrase are the same and the number of names is the same as in the ALTERNATE RECORD KEY clause the alternate record key of file-name-1 is a key of reference.

   d. When a part of row of data-names written in the RECORD KEY clause has been written in data-name-1 row in the KEY phrase, that is, data-names from the first to the n-1 or before among n data-names are written in the same order as in RECORD KEY clause, the prime record key written in the row of data-names-1 is a key of reference.

e. When a part of row of data-names written in the ALTERNATE RECORD KEY clause has been written in data-name-1 row in the KEY phrase, that is, data-names from the first to the n-1 or before among n data-names written in the ALTERNATE RECORD KEY clause are written in the same order as in the ALTERNATE RECORD KEY clause, the alternate record key to satisfy the following condition is a key of reference.

   When data-name-1 matches with a part of the alternate record key specified in one ALTERNATE RECORD KEY clause, it is a key of reference.

   When data-name-1 matches with a part of the alternate record key specified in two or more ALTERNATE RECORD KEY clauses, the alternate record key consisted of the minimum number of data-names is a key of reference. If two or more alternate record keys that consist of the minimum number of data-names, the alternate record key in which the ALTERNATE RECORD KEY clause has been written in the file control entry of file-name-1 first is a key of reference.

f. When data-name which is not data-name written in the RECORD KEY clause or ALTERNATE RECORD KEY clause has been specified as data-name-1 in the KEY phrase, the record key to satisfy all the following condition is a key of reference.

   The character position at the left end equals to that of the data item in data-name-1.

   The size of data-name is bigger or equals to that of the data item in data-name-1

   .

The record key consists of only one data item.

data-name is an alphanumeric data item, national data item, or group item.

g. When two or more record keys to satisfy all the conditions mentioned in (f) exist, the following record key will be a key of reference.

   When the prime record key satisfies the condition (f), the prime record key is a key of reference.

   When the prime record key does not satisfy the condition (f), the smallest alternate record key among alternate record keys to satisfy the condition will be a key of reference. If two or more alternate records key which are minimum, the alternate record key in which the ALTERNATE RECORD KEY clause has been written in the file control entry of file-name-1 first is a key of reference.

7. The key of reference determined in accordance with rule (6) is used to determine the order of record keys in the START statement. When the START statement has been executed successfully, the key of reference specified in the START statement will be used for the following READ statement in sequential access method.

8. When the START statement has been executed successfully, the searching method for the READ statement in sequential access method for the file assigned to file-name-1 is determined as follows:

   a. For the START statement without the REVERSED ORDER phrase, the searching operation will be performed in the normal ascending order.

   b. For the START statement with the REVERSED ORDER phrase, the searching operation will be performed in the reversed order.

9. If the invalid key condition occurs during the START statement execution, the control transfers according to the rules shown in the section titled “INVALID KEY phrase,” after the value indicating the invalid key condition is set in the I-O status of file-name-1.

10. If no other exceptional condition occurs during the START statement execution, the control transfers according to the rules shown in the section titled “INVALID KEY phrase,” after the value indicating the invalid key condition is set in the I-O status of file-name-1.

11. The END-START phrase partitions the range of the START statement.

12. When AUTOMATIC has been specified in the LOCK MODE phrase in the file related to file-name-1, the existing record is released from being locked if the START statement has been executed.

Rules Common to Format 1 and Format 2

1. According to the description in the KEY phrase, the record keys existing in the file assigned to file-name-1 are compared with data items as follows:

   a. When the KEY phrase has been written, the data item of data-name-1 is compared with the record key.

   b. When the KEY phrase has been omitted, the data item written in the RECORD KEY clause of file-name-1 is compared with the record key. The relation “IS EQUAL TO” is assumed to have been specified in the KEY phrase.

2. Keys of reference in the file, which have been sorted in ascending order according to the size of characters, are targets for comparing in the method described in (1). If sizes of operands differ, the right end of the longer operand is truncated on the right to be the same length as the shorter one, then characters are compared. As a result of comparison, the following processing will be performed:

   a. In format 1, the file position indicator is set to the value indicated in the first record to satisfy the comparison condition.

   b. In format 2, the file position indicator is set to the value indicated in the last record to satisfy the comparison condition.

   c. If no record in the file can satisfy the condition above, the invalid key condition occurs and the START statement execution fails.

Rules for Format 3

1. When the START statement in format 3 has been executed, the file position indicator of file-name-1 is allocated as follows:

   a. When the REVERSED ORDER phrase has been omitted, the file assigned to file-name-1 is allocated at the first record of the key of reference. The file position indicator is set to the value of the key of reference of the allocated record.

   b. When the REVERSED ORDER phrase has been written, the file assigned to file-name-1 is allocated at the last record of the key of reference. The file position indicator is set to the value of the key of reference of the allocated record.

c. If no valid record exists in the file, the invalid key condition occurs and the START statement execution fails.

2. data-name-1 is specified only to determine the key of reference. The value of data-name-1 is ignored.

STOP Statement (Nucleus)

The STOP statement terminates execution in a run unit. The STOP statement which specified literal-1 is an obsolete element.

[Format]

$$\underline{\text{STOP}} \left\{ \begin{array}{l} \underline{\text{RUN}} \\ \text{literal-1} \end{array} \right\}$$

Syntax Rules

1. literal-1 must not be a figurative constant starting with ALL or national nonnumeric literal.
2. When a numeric literal is specified to literal-1, it must be an unsigned integer. In this compiler, however, literal-1 or a literal with a symbol or a decimal-point can be specified.
3. To write the STOP RUN statement in the series of imperative statements in one sentence, the STOP RUN statement must be the last statement in the series of imperative statements.

General Rules

1. When the STOP RUN statement has been executed, one run unit of execution terminates and the control is transferred to the operating system.
2. If files remain open during execution of the STOP RUN statement, a CLOSE statement which does not specify selection for these files is executed. The USE procedure related to these files is not performed.
3. When the STOP RUN statement in which literal-1 has been written has been executed, execution of the run unit is interrupted and the value of literal-1 is reported to the operator. When the operator instructs to continue the execution, execution restarts from the next execution of the STOP statement.

STRING Statement (Nucleus)

The STRING statement links character-strings.

[Format]

$$\underline{\text{STRING}} \left\{ \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \ldots\ \underline{\text{DELIMITED}}\ \text{BY} \left\{ \begin{array}{l} \text{identifier-2} \\ \text{literal-2} \\ \underline{\text{SIZE}} \end{array} \right\} \right\} \ldots$$

$$\begin{array}{l} \underline{\text{INTO}}\ \text{identifier-3 [WITH}\ \underline{\text{POINTER}}\ \text{identifier-4]} \\ \text{[ON}\ \underline{\text{OVERFLOW}}\ \text{imperative-statement-1]} \\ \text{[}\underline{\text{NOT}}\ \text{ON}\ \underline{\text{OVERFLOW}}\ \text{imperative-statement-2]} \\ \text{[}\underline{\text{END-STRING}}\text{]} \end{array}$$

Syntax Rules

1. literal-1 and literal-2 must be nonnumeric literals, national nonnumeric literals, or figurative constants not starting with ALL.
2. identifier-1, identifier-2, and identifier-3 must be used for presentation.
3. To specify numeric data items to identifier-1 and identifier-2, they must be integer items that do not contain PICTURE symbol P.
4. identifier-3 must not be an alphanumeric edited data item, National edited data item, or numeric edited data item. identifier-3 must not be a data item in which the JUSTIFIED clause has been specified, either.
5. Partial reference of identifier-3 is not possible.
6. identifier-4 must be an integer item not containing PICTURE character P. identifier-4 must be large enough to hold the value of identifier-3 plus one character.
7. When the category of data in either of identifier-1, identifier-2, identifier-3, literal-1, or literal-2 is a National or national edited data item, all categories must be National or national edited data items.

General Rules

1. The STRING statement links some character-strings of identifier-1 or literal-1 and transfers the result to identifier-3. identifier-1 and literal-1 are called “sending items.” identifier-3 is called a “receiving item.” When the STRING statement has been executed, all or part of the character-string of each sending item is transferred to the high order end character position of the receiving item or

to the character position following the position specified in identifier-4.

To transfer all character-strings of the sending item, write SIZE in the DELIMITED BY phrase. To transfer a character-string from the high order end character position of the sending item to a certain character, specify identifier-2 or literal-2 in the DELIMITED BY phrase. identifier-2 and literal-2 are called “delimiters.”

To transfer sending items to the position following the specific character position specified in the receiving item, set the character position at identifier-4 in the POINTER phrase. The value of identifier-4 is increased according to the length of the transferred character each time it is transferred. To transfer the sending item at the position following the high order end character position of the receiving item, omit the POINTER phrase.

2. In accordance with the description in the DELIMITED BY phrase, the following character-strings in sending items are transferred:

    a. When identifier-2 or literal-2 has been written in the DELIMITED BY phrase, the character-string from the high order end of the sending item to where the same character as the delimiter appears is transferred. However, the delimiter itself is not transferred.

    b. When the SIZE has been written in the DELIMITED BY phrase, the whole character-string of the sending items is transferred.

3. The initial value of identifier-4 must be set before executing the STRING statement. The initial value must be more than 1. The initial value of identifier-4 is used as an initial value of the “current transfer position.” The “current transfer position” changes by 1 each time transfer

is performed and it becomes the value of identifier-4 when the STRING statement execution terminates.

4. When the POINTER phrase has been omitted, the initial value of the “current transfer position” is 1.

5. One character each is transferred from each sending item to the receiving item according to the transfer rules. The character-string is transferred from each sending item to the receiving item by repeating the following processing:

   a. When the “current transfer position” does not exceed the number of characters of the receiving item, one character in the sending item is transferred to the current transfer position.

   b. When the “current transfer position” exceeds the number of characters of the receiving item, the overflow condition occurs and the transfer operation terminates. Then, the control transfers in accordance with the description in the ON OVERFLOW phrase.

   c. The processing (a) or (b) repeats until the delimiter written in the DELIMITED BY phrase is detected (when identifier-2 or literal-2 has been written in the DELIMITED BY phrase), until all data in the sending item has been transferred, or until data is transferred up to the end of the receiving items.

6. Only the transferred part of the receiving item changes by the STRING statement execution. Other parts of the receiving item does not change by executing the STRING statement.

7. When the figurative constant has been specified as a delimiter, the figurative constant is regarded as one-nonnumeric literal or national nonnumeric literal.

8. After the transfer operation by executing the STRING statement, the control transfer depends on the condition

whether the ON OVERFLOW or NOT ON OVERFLOW is specified. The control transfer after transfer operation is shown in the table below.

ON OVERFLOW phase	**NOT ON OVERFLOW phase**	**Control transfer**	
		Overflow condition occurred in the transfer operation. (*3)	**No overflow condition occurred in the transfer operation.**
Yes	Yes	1.The transfer operation terminates and the control transfers to imperative-statement-1. 2.After imperative-statement-1 has been executed, the control transfers to the end of the STRING statement. (*1)	1.After all the character-string has been transferred, the control transfers to imperative-statement-2. 2.After imperative-statement-2 has been executed, the control transfers to the end of the STRING statement. (*2)
Yes	No	3.The transfer operation terminates and the control transfers to imperative-statement-1. 4.After imperative-statement-1 has been executed, the control transfers to the end of the STRING statement. (*1)	After all character-strings have been transferred, the control transfers to imperative-statement-2.
No	Yes	The transfer operation terminates and the control transfers to the end of the STRING statement.	1.After all the character-string has been transferred, the control transfers to imperative-statement-2. 2.After imperative-statement-2 has been executed, the control transfers to the end of the STRING statement. (*2)
No	No	The transfer operation terminates and the control transfers to the end of the STRING statement.	After all character-strings have been transferred, the control transfers to imperative-statement-2.

*1 When the procedure branching or conditional statement to cause control explicit transfer in the imperative statement has been written in imperative-statement-1, the control transfers according to the statement rules.

*2 When the procedure branching or conditional statement to cause control explicit transfer in the imperative statement has been written in imperative-statement-2, the control transfers according to the statement rules.

*3 The overflow condition occurs if:

The initial value of identifier-4 is less than 1 or more than the number of characters in the character-string of identifier-3.

The “current transfer position” is larger than the number of characters in the character-string of identifier-3. That is, some characters have been left in the sending item without being transferred even though character-strings have been transferred up to the low order end of identifier-3.

The “current transfer position” is larger than the number of characters in the character-string of identifier-3. That is, some characters have been left in the sending item without being transferred even though character-strings have been transferred up to the low order end of identifier-3.

9. The END-STRING phrase partitions the range of the STRING statement.

10. The result of the STRING statement execution is not defined in the following cases:

    a. identifier-1 or identifier-2 possesses the same storage area as identifier-3 or identifier-4.

    b. identifier-3 possesses the same storage area as identifier-4.

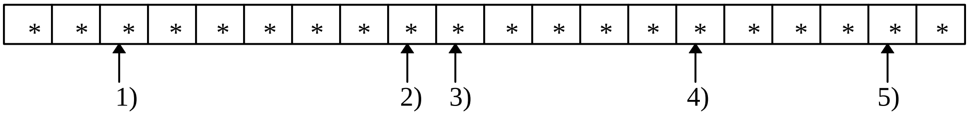
Processing Flow of STRING Statement

An example of processing flow of the STRING statement is given below.

```
 [Data division]
77  AN-1  PIC  X (6)    VALUE "STRING".
77  AN-2  PIC  X (5)    VALUE "STATE".
77  AN-4  PIC  X (6)    VALUE "MENT#0".
77  AN-5  PIC  X (4)    VALUE "#123".
77  AN-7  PIC  X (20)   VALUE "********************".
77  ED-8  PIC  99       VALUE  3.
 [Procedure division]
    STRING  AN-1  SPACE  AN-2  DELIMITED  BY  SIZE
                AN-4  AN-5  DELIMITED  BY  "#"
                       INTO AN-7  WITH  POINTER  ED-8.
```

When this STRING statement has been executed, sending items (AN-1, SPACE, AN-2, AN-4 and AN-5) are linked sequentially and transferred to the receiving item (AN-7).

[Contents of AN-7 before execution]



*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

[Contents of AN-7 after execution]

*	*	S	T	R	I	N	G		S	T	A	T	E	M	E	N	T	*	*

a. The initial value of ED-8 is 3 so that the third (position 1) and subsequent characters of AN-7 will be the target for transferring. <<START>>

b. The entire character-string of AN-1 is transferred first. As a result of transfer, the value of ED-8 is updated to 9 (position 2).

c. Then, the figurative constant SPACE is transferred as one nonnumeric literal. As a result of transfer, the value of ED-8 is updated to 10 (position 3).

d. The entire character-string of AN-2 is transferred next. As a result of transfer, the value of ED-8 is updated to 15 (position 4).

e. The character-string up to # in AN-4 is transferred. As a result of transfer, the value of ED-8 is updated to 19 (position 5).

f. Since there is no character string containing # in AN-5, no character-string is transferred. The value of ED-8 is not updated.

g. The value of ED-8 is 19 when the STRING statement execution terminates.

SUBTRACT Statement (Nucleus)

The SUBTRACT statement obtains the result of subtraction.

[Format 1] The target of the subtraction is replaced with the minuend.

```
SUBTRACT { identifier-1 } ...
         { literal-1    }

    FROM { identifier-2  [ ROUNDED ] } ...
   [ ON SIZE ERROR imperative-statement-1 ]
   [ NOT ON SIZE ERROR imperative-statement-2 ]
   [ END-SUBTRACT ]
```

[Format 2] The result of the subtraction is stored in a specified data item.

```
SUBTRACT { identifier-1 } ...
         { literal-1    }

   FROM { identifier-2 } GIVING { identifier-3 [ ROUNDED ] } ...
        { literal-2    }
  [ ON SIZE ERROR imperative-statement-1 ]
  [ NOT ON SIZE ERROR imperative-statement-2 ]
  [ END-SUBTRACT ]
```

[Format 3] Subtraction is performed by corresponding the data items following two group items.

```
SUBTRACT { CORRESPONDING }
         { CORR          }

    identifier-1  FROM   identifier-2   [ ROUNDED ]
   [ ON  SIZE  ERROR  imperative-statement-1 ]
   [ NOT  ON  SIZE  ERROR  imperative-statement-2 ]
   [ END-SUBTRACT  ]
```

Syntax Rules

1. In formats 1 and 2, identifier-1 and identifier-2 must be numeric data items. In format 3, identifier-1 and identifier-2 must be group items.
2. identifier-3 must be a numeric data item or numeric edited data item.
3. literal-1 and literal-2 must be numeric literals.
4. CORRESPONDING is synonymous with CORR.

General Rules

Rules for Format 1

The SUBTRACT statement in format 1 subtracts the sum of operands before FROM from identifier-2 and stores the result in identifier-2. The subtraction is performed in the order of the row in which identifier-2 has been written.

Rules for Format 2

The SUBTRACT statement in format 2 subtracts the sum of operands before FROM from the operand after FROM and stores the result in identifier-3. The subtraction is performed in the order of the row in which identifier-3 has been written.

Rules for Format 3

The SUBTRACT statement in format 3 works out for the difference of items with the same qualifier on names between data items dependent on identifier-1 and data items dependent on identifier-2. The data item dependent on identifier-1 is treated as a subtrahend and the data item dependent on identifier-2 is treated as a minuend in order to obtain the difference, which is to be stored to the data item dependent on identifier-2. This result is the same as the result when the SUBTRACT statements have been written in each corresponding identifier.

Rules Common to Format 1 to Format 3

1. The END-SUBTRACT phrase partitions the range of the SUBTRACT statement.
2. See the section titled “Common rule related to statement,” for rules for the ROUNDED phrase, ON SIZE ERROR phrase, CORRESPONDING phrase, the operation, and the transfer.

SUPPRESS Statement (Report Writer)

The SUPPRESS statement suppresses the report group presentation.

[Format]

```
    SUPPRESS PRINTING
    ‾‾‾‾‾‾‾‾
```

Syntax Rule

The SUPPRESS statement can be written only during the USE BEFORE REPORTING procedure.

General Rules

1. The report groups to which the SUPPRESS statement suppresses presentation are only the report groups specified in the USE procedure in which the SUPPRESS statement has been written.
2. To suppress the report group presentation, the SUPPRESS statement must be executed to each group.
3. When the SUPPRESS statement has been executed, the report writer control system performs the following processing:
   a. Presentation of printing lines in the report group
   b. Processing for all LINE NUMBER phrases in the report group
   c. Processing for the NEXT GROUP phrase in the report group
   d. Line-counter updating

TERMINATE Statement (Report Writer)

The TERMINATE statement terminates the report processing.

[Format]

<u>TERMINATE</u> {report-name-1} ...

Syntax Rules

report-name-1 must be specified in the report description entry of report section in the data division.

General Rules

1. When the TERMINATE statement has been executed, the report writer control system writes and displays all control footing groups in the ascending order from the lowest level. Then, the report writer control system writes and displays the report footing report groups. The report writer control system enables the old values of control data items which have been stored previously to be used in the SOURCE phrase or USE procedure of the control footing group and the report footing report group, just as the system detected the control break of the control data-name in the highest level.
2. If the TERMINATE statement is executed when the GENERATE statement has not been executed even once after the INITIATE statement execution, the report writer control system does not write and display the report group or perform any processing related to the operation.

3. When the page feed is required for displaying the body group during the report presentation, the report writer control system automatically performs the processing if the page heading report group and the page footing group have been defined.

4. When the INITIATE statement has not been executed for a certain report or when the TERMINATE statement has already been executed, the TERMINATE statement must not be executed.

5. The result of the TERMINATE statement execution when two or more report-names-1 have been written is the same as when TERMINATE statements have been executed for each report in the order in which report names have been written.

6. The TERMINATE statement has no function to close files related to this report. To close a file after the TERMINATE statement execution, the CLOSE statement must be executed separately. If the reports related to the file have been started, they must be all terminated before executing the CLOSE statement.

UNLOCK Statement (Sequential File, Relative File, Indexed File)

The UNLOCK statement cancels all locks obtained for file connectors.

[Format]

$$\underline{\text{UNLOCK}}\ \text{file-name-1} \left\{ \begin{array}{l} \text{RECORD} \\ \text{RECORDS} \end{array} \right\}$$

General Rules

1. file-name-1 must be a file created by executing the OPEN statement successfully, regardless of whether it is locked or not.
2. When the UNLOCK statement has been executed, the lock obtained for the file connector of file-name-1 is canceled.
3. Even if no lock exists in the file assigned to file-name-1, the UNLOCK statement is executed successfully.

UNSTRING Statement (Nucleus)

The UNSTRING statement unlinks character-strings.

[Format]

```
UNSTRING identifier-1
  [ DELIMITED BY [ALL] { identifier-2 } [ OR [ALL] { identifier-3 } ] ... ]
                       { 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. literal-1 and the literal-2 must be nonnumeric literals, national nonnumeric literals, or figurative constants not starting with ALL.
2. identifier-1, identifier-2, identifier-3, and identifier-5 must be alphanumeric data items, national data items, or national edited data items.
3. identifier-4 must be an alphabetic item, alphanumeric data item, national data item or numeric data item. The numeric data item is used for presentation and must be a character-string not containing PICTURE character “P.”
4. identifier-6 and identifier-8 must be integers not containing the PICTURE character “P.”

5. identifier-7 must be an integer not containing the PICTURE character “P.” identifier-7 must be large enough to hold the data item of identifier-1 plus one character.

6. No reference modifier can be added to identifier-1.

7. When the category of data in any of identifier-1, to identifier-5, literal-1, or literal-2 is a national or national edited data item, all categories must be national or national edited data items.

General Rules

1. The UNSTRING statement unlinks some character-strings of identifier-1 according to the condition of the DELIMITED BY phrase and transfers the strings to identifier-4. identifier-1 is called a “sending item.” identifier-4 is called a “receiving item.” identifier-2, identifier-3, literal-1, and literal-2 are called “delimiters.”

   The UNSTRING statement with the description in the DELIMITED BY phrase delimits the high order end character position of the sending item or the character-string following the specific character position with a delimiter and transferred the delimited items to each receiving item in the order that they have been delimited. The UNSTRING statement without the description in the DELIMITED BY phrase delimits the high order end character position of the sending item or the character-string following the specific character position at the length of each receiving item and transferred the delimited items to each receiving item in the order that they have been delimited.

When the parts following the high order end character position of the sending item are targets for transfer, the POINTER phrase is omitted. When the parts following the specific character position of the sending item are targets for transfer, the POINTER phrase is written and the character position is set to identifier-7 in the POINTER phrase. The value of identifier-7 is incremented each time it is compared with the delimiter.

The delimiter or the number of characters up to the delimiter can be obtained by transferring to each receiving item. To obtain the delimiter, write the DELIMITER IN phrase. To obtain the number of characters up to the delimiter, write the COUNT IN phrase.

The number of receiving items of which character-strings have been actually transferred can be counted. To count the number of receiving items of which character-strings have been actually transferred, write the TALLYING IN phrase.

2. The initial value of identifier-7 must be set before executing the UNSTRING statement. The initial value must be more than 1. The initial value of identifier-7 is used as an initial value of the “current test position.” The value of the “current test position” increments by 1 each time the character in the sending item is tested. The value when the UNSTRING statement execution terminates becomes the value of identifier-7.

3. When the POINTER phrase has been omitted, the initial value of the “current test position” is 1.

4. When the DELIMITED BY phrase has been written, the sending items are delimited in the following procedure and transferred to each receiving item. The first “current receiving item” is the first identifier-4 written in the INTO phrase in the following description.

   a. When the “current test position” does not exceed the number of characters of the sending item, one character in the sending item following the “current test position” is compared with the delimiter. When two or more delimiters have been written, the character-strings starting from the same position are compared repeatedly in the order that delimiters have been written until they match the delimiters. When the delimiter contains two or more characters, the character-string containing the same number of characters as that of the delimiter is compared with the delimiter.

      When the “current test position” exceeds the number of characters of the sending item, the delimiter test terminates.

   b. If the character matching to the delimiter is found at the position after the “current test position” in the processing (a), the character-string from the “current test position” to the character matching to the delimiter is transferred to the “current receiving item.” At that time, the character matching the delimiter is not transferred. When the DELIMITER IN phrase has been written, the delimiter is transferred to identifier-5.

c. If no character matching the delimiter is found in processing (a), the character-string in the sending item is transferred to the “current receiving item.” A space is transferred to identifier-5 written in the DELIMITER IN phrase.

d. If the character at the “current test position” matches the delimiter in processing (a), the following values are transferred to the “current receiving items.”

   When the “current receiving item” is an alphabetic item, alphanumeric data item or national data item, a space is transferred.

   When the “current receiving item” is a numeric data item, zero is transferred.

e. When the COUNT IN phrase has been written, the number of characters transferred to the “current receiving item” is set to identifier-6. In the case of (d), zero is set to identifier-6.

f. The “current test position” and the “current receiving item” are changed as follows and the processing (a) to (f) repeats until all characters in the sending items are transferred or all receiving items are used.

   The character position to the right of the character matching the delimiter is the “current test position.”

   The next receiving item is the “current receiving item.”

5. When the DELIMITED BY phrase has been omitted, the sending items are divided in the following procedure and transferred to the receiving items. The initial value of the “current receiving item” in the following processing is the first identifier-4 in the INTO phrase.

a. When the “current test position” does not exceed the number of characters in the sending items, the number of characters in the sending items after the “current test position” is compared with the number of characters in the “current receiving item.” When the “current receiving item” is a numeric data item, the following value is used as the number of characters.

   When the SEPARATE is specified in the SIGN clause, the number of characters excluding the number of operational signs is used.

   When the decimal point is specified in the PICTURE clause, the number of characters excluding the number of decimal points is used.

b. When the “current test position” not exceeds the number of characters in the sending items, the transfer operation terminates.

c. The character-string containing only the number of characters of the “current receiving item” starting from the “current test position” is transferred to the “current receiving item.” Then, the “current test position” and the “current receiving item” are changed as follows and the processing (a) to (f) repeats until all characters in the sending items are transferred or all receiving items are used.

   The character position to the right of the last character transferred is the “current test position.”

   The next receiving item is the “current receiving item.”

6. When the TALLYING phrase has been written, the number of receiving items of which character-strings have actually been transferred among receiving items is added to identifier-8. The initial value must be set to identifier-8 in advance. The value of identifier-8 is incremented by one in each identifier-4.

7. When the figurative constant has been specified to the delimiter, the figurative constant is regarded as a one-character nonnumeric literal.

8. When the ALL has been specified before the delimiter, characters in the character-string of the sending items, from the same character as the delimiter to the character which is not the delimiter, are skipped. When the DELIMITER IN phrase has been specified, the delimiter in one of repetitive operation is set to identifier-5 regardless of the number of skipped characters.

9. Any character in the computer character set can be set as a delimiter.

10. After the transfer operation by executing the UNSTRING statement, the control transfer depends on whether the ON OVERFLOW or NOT ON OVERFLOW is specified. The control transfer after transfer operation is shown in the table below.

ON OVERFLOW phrase	**NOT ON OVERFLOW phrase**	**Control transfer**	
		Overflow condition occurred in the transfer operation. (*3)	No overflow condition occurred in the transfer operation
Yes	Yes	1. The transfer operation terminates and the control transfers to imperative-statement-1. 2. After imperative-statement-1 has been executed, the control transfers to the end of the UNSTRING statement. (*1)	1. After all the character-string has been transferred, the control transfers to imperative-statement-2. 2. After imperative-statement-2 has been executed, the control transfers to the end of the UNSTRING statement. (*2)
Yes	No	1.The transfer operation terminates and the control transfers to imperative-statement-1. 2.After imperative-statement-1 has been executed, the control transfers to the end of the UNSTRING statement. (*1)	After all character-strings have been transferred, the control transfers to the end of the UNSTRING statement.
No	Yes	The transfer operation terminates and the control transfers to the end of the UNSTRING statement.	1. After all the character-string has been transferred, the control transfers to imperative-statement-2. 2. After imperative-statement-2 has been executed, the control transfers to the end of the UNSTRING statement. (*2)
No	No	The transfer operation terminates and the control transfers to the end of the UNSTRING statement.	After all character-strings have been transferred, the control transfers to the end of the UNSTRING statement.

*1 When the procedure branching or conditional statement to cause control explicit transfer in the imperative statement has been written in imperative-statement-1, the control transfers according to the statement rules.

*2 When the procedure branching or conditional statement to cause control explicit transfer in the imperative statement has been written in imperative-statement-2, the control transfers according to the statement rules.

*3 The overflow condition occurs in the following cases:
The initial value of identifier-4 is less than 1 or more than the number of characters in the character-string of identifier-3.
The “current test position” is larger than the number of characters in the character-string of identifier-1. That is, some characters have been left in the receiving item without being tested even though the character-strings have been transferred to all receiving items.

11. The END-UNSTRING phrase partitions the range of the UNSTRING statement.

12. The result of the UNSTRING statement execution is not defined in the following cases:

    a. identifier-1, identifier-2, or identifier-3 possess the same storage area as identifier-4 to identifier-8.

    b. identifier-4, identifier-5, or identifier-6 possess the same storage area as identifier-7 or identifier-8.

    c. identifier-7 possesses the same storage area as identifier-8.

Processing Flow of UNSTRING Statement

An example of processing flow of the UNSTRING statement is given below.

```
 [Data division]
77  AN-SND  PIC  X (17)  VALUE "AA0#BBB##CCC**DDD".
77  AN-DEL  PIC  X       VALUE "*".
77  AN-R1   PIC  X (2)   VALUE "II".
77  AN-R2   PIC  X (3)   VALUE "JJJ".
77  AN-R3   PIC  X (4)   VALUE "KKKK".
77  AN-R4   PIC  X (3)   VALUE "LLL".
77  AN-D2   PIC  X (2)   VALUE "MM".
77  AN-D4   PIC  X       VALUE "N".
77  ED-C3   PIC  99      VALUE  11.
77  ED-C4   PIC  99      VALUE  22.
77  AN-PTR  PIC  99      VALUE  1.
77  AN-TLY  PIC  99      VALUE  0.
 [Procedure division]
   UNSTRING  AN-SND
         DELIMITED  BY  ALL  "#"  OR  AN-DEL
         INTO  AN-R1
                 AN-R2  DELIMITER  IN  AN-D2
                 AN-R3  COUNT  IN  ED-C3
                 AN-R4  DELIMITER  IN  AN-D4
                          COUNT  IN  ED-C4
         WITH  POINTER  ED-PTR
         TALLYING  IN  ED-TLY
         ON  OVERFLOW  GO  TO  OVERFLOW-PROC.
```

When this UNSTRING statement has been executed, sending items (AN-SND) are delimited and transferred to the receiving item (AN-R1, AN-R2, AN-R3, and AN-R4).

[Contents of AN-SND before execution]

A	A	0	#	B	B	B	#	#	C	C	C	*	*	D	D	D
↑				↑					↑				↑	↑		
1)				2)					3)				4)	5)		

[Contents of AN-SND after execution]

AN-R1

A	A

AN-R2

B	B	B

AN-R3

C	C	C	

AN-R4

[Contents of receiving items of delimiters after execution]

AN-D2

#	

AN-D4

*

a. The initial value of ED-PTR is 1 so that the first (position 1) and subsequent characters of AN-SND will be targets for transferring. The target character-strings are delimited by delimiters and transferred to the sending items. The delimiters are repetitive # and one-character *.

b. The character-string starting from position 1) to the "#" is transferred to the AN-R1 first. As a result of transfer, the value of ED-PTR is changed to 5 (position 2).

c. Then, the character-string starting from position 2) to the "#" is transferred AN-R2. As a result of transfer, the value of ED-PTR is changed to 10 (position 3).

d. The character-string starting from position 3) to the "*" is transferred to AN-R3 next. As a result of transfer, the value of ED-PTR is updated to 14 (position 4). Since the number of characters up to the delimiter is 3, the value of the ED-C3 is changed to 3.

e. Since the character position of 4) is at “*”, a blank is transferred to AN-R4. As a result of transfer, the value of ED-PTR is changed to 15 (position 5). Since the number of characters up to the delimiter is 0, the value of the ED-C4 is changed to 0.

f. The value of the ED-PTR is 15 when the UNSTRING statement execution terminates. Since there are characters not transferred left in AN-SND, the control transfers to the OVERFLOW-PROC.

g. Since character-strings have been transferred to four receiving items (AN-R1, AN-R2, AN-R3 and AN-R4), ED-TLY is incremented by 4 and the value of ED-TLY is changed to 4.

Example of UNSTRING Statement

Suppose that the data to be used in the UNSTRING statement is defined as follows:

```
77 SEND-A   PICTURE  X (30).
77 RCV-1    PICTURE  X (10).
77 RCV-2    PICTURE  X (10).
77 RCV-3    PICTURE  X (10).
77 DEL-1    PICTURE  XX.
77 DEL-2    PICTURE  XX.
77 DEL-3    PICTURE  XX.
77 CTR-1    PICTURE  99.
77 CTR-2    PICTURE  99.
77 CTR-3    PICTURE  99.
77 PTR-A    PICTURE  99.
77 TALL-A   PICTURE  99.
```

Suppose that the following values are set in each data item before executing the UNSTRING statement. In the following description, ! indicates a one-character space.

SEND-A:

```
MUNICH/NEW ! YORK//TOKYO********
```

RCV-1, RCV-2, RCV-3, DEL-1, DEL-2 and DEL-3: Blank
CTR-1, CTR-2, CTR-3 and TALL-A: 00
PTR-A: 01

Examples of UNSTRING statement description and the execution result are given as follows:

Example 1

```
UNSTRING  SEND-A
        INTO  RCV-1  RCV-2  RCV-3
   WITH  POINTER  PTR - A.
```

Identifier	**Identifier value after execution**
RCV-1	MUNICH / NEW
RCV-2	!YORK / / TOK
RCV-3	YO********
PTR-A	31

Example 2

```
UNSTRING  SEND-A
     DELIMITED  BY  "/"
     INTO  RCV-1  COUNT  IN  CTR-1
           RCV-2  COUNT  IN  CTR-2
           RCV-3  COUNT  IN  CTR-3
     WITH  POINTER  PTR -A  TALLYING  IN  TALL-A.
```

Identifier	**Identifier value after execution**
RCV-1	MUNICH!!!!
RCV-2	NEW!YORK!!
RCV-3	!!!!!!!!!!
CTR-1	06
CTR-2	08
CTR-3	00
PTR-A	18
TALL-A	03

In this example, there are untested characters left in the SEND-A even though all receiving items have been used. When the ON OVERFLOW phrase has been written in the UNSTRING statement in this example, the imperative statement written in the ON OVERFLOW phrase is executed.

Example 3

```
UNSTRING  SEND-A
      DELIMITED  BY ALL  "/"  OR  ALL "*"
      INTO   RCV-1  DELIMITER  IN  DEL-1
                    COUNT  IN  CTR-1
             RCV-2  DELIMITER  IN  DEL-2
                    COUNT  IN  CTR-2
             RCV-3  DELIMITER  IN  DEL-3
                    COUNT  IN  CTR-3
      WITH  POINTER  PTR -A  TALLYING  IN  TALL-A.
```

Identifier	**Identifier value after execution**
RCV-1	MUNICH!!!!
RCV-2	NEW!YORK!!
RCV-3	TOKYO!!!!!
DEL-1	/!
DEL-2	/!
DEL-3	*!
CTR-1	06
CTR-2	08
CTR-3	05
PTR-A	31
TALL-A	03

USE Statement (Creation of Sequential Files, Relative Files, Indexed Files, Presentation Files, and Reports)

This statement specifies start of an I-O error handling procedure.

[Format 1] For sequential, relative, and indexed files

```
USE [GLOBAL] AFTER STANDARD

          { EXCEPTION }
          { ERROR     }  PROCEDURE  ON

          { {file-name-1} ... }
          { INPUT             }
          { OUTPUT            }  .
          { I-O               }
          { EXTEND            }
```

[Format 2] For presentation file

```
USE [GLOBAL] AFTER STANDARD

          { EXCEPTION }
          { ERROR     }  PROCEDURE  ON

          { {file-name-1} ... }
          { INPUT             }
          { OUTPUT            }  .
          { I-O               }
```

[Format 3] For report file

```
USE [GLOBAL] AFTER STANDARD

          { EXCEPTION }
          { ERROR     }  PROCEDURE  ON

          { {file-name-1} ... }
          { OUTPUT            }  .
          { EXTEND            }
```

Syntax Rules

1. The USE statement must follow the section header in the declarative in the procedure division. Only one USE statement can be defined. It is followed by several paragraphs that define USE procedures. These USE procedure paragraphs may be omitted. The USE AFTER STANDARD EXCEPTION procedure is abbreviated to “USE procedure” in this explanation.

2. Each USE statement must have a unique file-name-1. Duplicate file-name-1 descriptions would cause multiple USE procedures to run at the same time.

3. EXCEPTION and ERROR are synonymous.

4. Files having different ORGANIZATION or ACCESS MODE phrase values can be specified in a list of file-name-1 descriptions. Such files can also be implicitly referenced using the USE statement.

5. Only one USE statement can be written in the declarative in one procedure division if it contains the INPUT, OUTPUT, I-O, or EXTEND phrase.

General Rules

1. The USE statement specifies the conditions used to execute the USE procedure. The USE statement itself is not executed.

2. The USE procedure is executed if an input-output statement (including input-output statements executed implicitly) has failed, except when the AT END phase (sequential, relative, indexed, or presentation file) or INVALID KEY phase (relative or indexed file) takes precedence. Specify the conditions for executing the USE procedure as follows:

   a. The file-name-1 phrase executes the USE procedure if the input-output statement related to file-name-1 failed.

   b. The INPUT phrase specifies to execute the USE procedure during execution of the input-output statement related to the file opened in input mode or if the file failed to be opened in input mode.

   c. The INPUT phrase specifies to execute the USE procedure during execution of the input-output statement related to the file opened in output mode or if the file failed to be opened in output mode.

   d. The I-O phrase specifies to execute the USE procedure during execution of the input-output statement related to the file opened in input-output mode or if the file failed to be opened in input-output mode.

   e. The EXTEND phrase specifies to execute the USE procedure during execution of the input-output statement related to the file opened in extended mode or if the file failed to be opened in extended mode.

3. The declarative section can be written in any program from the innermost program to the outermost program.
4. A declarative section is selectively executed based on the priorities listed below if an input-output statement failed in a nested program. When an USE statement that satisfies specified conditions is found, only the USE procedure following the USE statement is executed.
    a. USE statement that is written in a program containing the failed input-output statement and that satisfies conditions.
    b. USE statement that has the GLOBAL phrase in a program directly or indirectly containing the failed input-output statement and that satisfies conditions. If more than one USE statement is found, the USE statement in the highest nesting level program is selected.
5. Conditions specified in the USE statement are checked during execution of the separately compiled program containing the USE statement. Conditions in the USE statement in another compiled program.
6. The procedure shall not be referenced from the USE procedure.
7. The PERFORM statement must be used when the section-name in the declarative section and the paragraph name in the USE procedure is referenced from the procedure in another declarative section or procedure division.

8. When file-name-1 is defined in the USE statement, conditions written in other USE statements do not apply to file-name-1.
9. When no serious error occurred in the I-O status after execution of the USE procedure, control is transferred to the executable statement following the input-output statement that executed the USE procedure. For details on control transfer when a serious error occurs in the I-O status, refer to “COBOL85 User's Guide.”
10. Before the USE procedure is completed, no statement shall be executed which may run the procedure again.

USE BEFORE REPORTING Statement (Creation of Reports)

This statement specifies start of the procedure that is executed immediately before a report group is created and displayed.

[Format]

USE BEFORE REPORTING identifier-1

Syntax Rules

1. The USE BEFORE REPORTING statement must follow the section header in the declarative in the procedure division. Only one USE statement can be defined. It is followed by several paragraphs that define USE BEFORE REPORTING procedures. These USE BEFORE REPORTING procedure paragraphs may be omitted.
2. The identifier-1 must be the report group in the report section of the data division. The same identifier-1 must not be specified in more than one USE BEFORE REPORTING statement.

3. The GENERATE, INITIATE, or TERMINATE statement must not be written in the USE BEFORE REPORTING procedure. These statements must not be written in the range of execution of the PERFORM statement in the USE BEFORE REPORTING procedure.
4. The value of control data items must not change in the USE BEFORE REPORTING procedure.

General Rules

1. The USE BEFORE REPORTING specifies conditions used to execute the USE BEFORE REPORTING procedure. The USE BEFORE REPORTING statement itself is not executed.
2. The declarative section can be written in any program from the innermost program to the outermost program.
3. The USE BEFORE REPORTING procedure is executed by the Report Writer Control System immediately before the report group specified in identifier-1 is created and displayed.
4. The procedure must not be referenced from the USE BEFORE REPORTING procedure.
5. The PERFORM statement must be used when the section-name in the declarative section and the paragraph name in the USE BEFORE REPORTING procedure is referenced from the procedure in another declarative section or procedure division.
6. Before the USE BEFORE REPORTING procedure is completed, no statement shall be executed which may run the procedure again.

USE FOR DEAD-LOCK Statement (Presentation File)

The USE FOR DEAD-LOCK statement specifies the procedure to be executed when a deadlock occurs.

[Format]

```
USE FOR DEAD-LOCK
```

Syntax Rules

1. The USE FOR DEAD-LOCK statement must be written following the Declaratives section header in the Procedure Division. The USE FOR DEAD-LOCK statement, as such, must be a sentence. After the USE FOR DEAD-LOCK statement, write several procedures that define procedures. Procedure paragraphs do not need to be written.
2. The USE FOR DEAD-LOCK statement itself is not executed, and only specifies the execution condition for the USE FOR DEAD-LOCK procedures.
3. The USE FOR DEAD-LOCK statement can only be written once in a program.
4. The USE FOR DEAD-LOCK statement cannot be written in a program that is contained in another program.

General Rules

1. The specified procedure is executed by the transaction control system.
2. Any procedure in the Procedure Division must not be referenced from within a Declaratives procedure.
3. A procedure-name associated with the USE FOR DEAD-LOCK statement can be referenced in another declarative section or in a procedure in the Procedure Division only by using the PERFORM statement.
4. During its execution, a USE FOR DEAD-LOCK procedure must reference a procedure that is not a Declaratives procedure in a statement such as GO TO. A USE FOR DEAD-LOCK procedure must not be executed until its end.
5. If a USE FOR DEAD-LOCK procedure is executed until its end, the COBOL standard processing procedure is executed.
6. In a USE FOR DEAD-LOCK procedure, any statement that might cause this same USE FOR DEAD-LOCK statement to be executed again must not be executed.
7. If a run unit contains two or more programs in which USE FOR DEAD-LOCK statements are written, control passes to the program that was executed most recently (of those programs having USE FOR DEAD-LOCK statements).
8. Refer to the “COBOL85 User's Guide” for the action taken when a USE FOR DEAD-LOCK statement is written in a run unit.

WRITE Statement (Sequential File)

This statement writes records to a file or scrolls lines on a logical page.

[Format]

```
WRITE record-name-1
   [FROM identifier-1]
   [ { BEFORE }  ADVANCING
     { AFTER  }
              { { identifier-2 } [ LINE  ] }
              { { literal-1    } [ LINES ] }
              { { mnemonic-name-1 }        }
              { { PAGE            }        }
   [ AT { END-OF-PAGE }    imperative-statement-1 ]
   [    { EOP         }                           ]
   [ NOT AT { END-OF-PAGE }    imperative-statement-2 ]
   [        { EOP         }                           ]
   [END-WRITE]
```

Syntax Rules

1. The record-name-1 must be the name of the record defined in the file section of the data division. It can be qualified by the file name.
2. When a data item is specified in identifier-1, the same area must not be specified for record-name-1 and identifier-1. A function-identifier specified in identifier-1 must be an alphanumeric function.

3. identifier-2 must be an integer item.

4. integer-1 can be zero.

5. When the LINAGE clause is written in a file description entry related to record-name-1, mnemonic-name-1 must not be written.

6. mnemonic-name-1 must be assigned to the function name (CHANNEL-01 to CHANNEL-12, SLC, CTL, STACKER-01 or STACKER-2) in the special-names paragraph in the environment division.

7. A WRITE statement must not contain both the ADVANCING PAGE phrase and END-OF-PAGE phrase.

8. When the END-OF-PAGE or NOT END-OF-PAGE phrase is specified, the LINAGE clause must be written in a file description entry related to record-name-1.

9. END-OF-PAGE and EOP are synonymous.

10. The END-OF-PAGE phrase can be written in the WRITE statement for the print file only. It cannot be written in the WRITE statement for the print file having the FORMAT clause.

11. The ADVANCING phrase can only be written in a WRITE statement for a print file.

    HP The ADVANCING phrase can also be written in a WRITE statement for a line sequential file.

12. A mnemonic name assigned to the function name CTL must not be specified in mnemonic-name-1 when the ADVANCING phrase is written in the WRITE statement for the print file having the FORMAT clause.

13. No ADVANCING phrase must be written when the record specified in the CONTROL RECORDS clause is specified in record-name-1.

14. Reference modification must not be made using identifier-1 when identifier-1 explicitly or implicitly contains the data item specified in the CHARACTER TYPE or PRINTING POSITION clause.

General Rules

1. The file related to record-name-1 must have been opened in output or extended mode before execution of the WRITE statement.

2. After the WRITE statement has been executed successfully, the record specified in record-name-1 cannot be used in the record area any more. However, execution of the WRITE statement does not make the record unusable if the file related to record-name-1 is specified in the SAME RECORD AREA clause of the I-O control paragraph. The program can use record-name-1.

3. The WRITE statement having the FROM phrase produces the same result as the following statements executed in the provided order:

   a. Statements according to the MOVE statement rules

      MOVE identifier-1 TO record-name-1

   b. Same WRITE statement as above, but without FROM phrase

      The result is based on the CHARACTER TYPE or PRINTING POSITION phrase in the data item contained in identifier-1 when identifier-1 explicitly or implicitly contains the national data item or national edited data item with the CHARACTER TYPE or PRINTING POSITION phrase.

4. After the WRITE statement has been executed successfully, data in the area of record-name-1 may not be accessible any more, except when the file related to record-name-1 is specified in the SAME RECORD AREA clause. Data in the area of identifier-1 is accessible.

5. The file position indicator remains unchanged after the WRITE statement has been executed.

6. The value of the I-O status of the file related to record-name-1 is updated after execution of the WRITE statement.

7. The record is passed to the i-o-control system after execution of the WRITE statement.

8. The size of the record in record-name-1 must not exceed the maximum record length of the file related to record-name-1. The size of the record in record-name-1 must not be less than the minimum record length of the file related to record-name-1. If these conditions are not satisfied, the WRITE statement will fail and no record is written. At this time, the record area of the file related to record-name-1 remains unchanged. The I-O status of the file related to record-name-1 is set with a value indicating the cause of the state.

   Note that the compiler executes the WRITE successfully when a record smaller than the minimum record size is written.

9. The END-WRITE phrase delimits the range of the WRITE statement.

10. The sequence of records in the sequential file is determined by the execution sequence of the WRITE statements which create the file. It is changed only when records are appended to the file.

11. When records are added to a file opened in extended and shared mode with the WRITE statement for the file connector of two or more files, the order of record addition may be undefined.

12. When the file related to record-name-1 is opened in extended mode, the WRITE statement appends records as if the file has been opened in output mode. If the file contains records, the WRITE statement adds records immediately after the last record in the file when WRITE is executed first after the OPEN statement having the EXTENDED phrase has been executed.

13. If the WRITE statement attempts to write records beyond the file size specified explicitly or implicitly from an external program, an exception condition occurs to execute processing below. Data in the record area remains unchanged.

    a. The I-O status of the file related to record-name-1 is set with a value indicating write over an area.

    b. When the USE AFTER STANDARD EXCEPTION procedure corresponding to the file related to record-name-1, the procedure is executed. After execution of the procedure, control is passed according to the rules of the USE statement. Otherwise, control is passed to the end of the WRITE statement when the FILE STATUS clause is specified in the file related to record-name-1. If neither USE AFTER EXCEPTION procedure nor FILE STATUS phrase is written, the result may not be defined.

14. The program performs the following processing when the reel or unit file reaches its end and it is within an area defined externally:

a. Executes the standard at-end reel or unit label procedure.

b. Exchanges the reel or unit. That is, the volume indicator is updated to the next reel or unit in the file.

c. Executes the standard start reel or unit label procedure.

15. The ADVANCING and END-OF-PAGE phrases for the print file perform vertical line control in the print page presentation. If the ADVANCING phrase is omitted, AFTER ADVANCING 1 LINE is assumed and the program automatically reforms line feed. When ADVANCING is specified, line feed is performed as explained below.

a. When a positive integer is specified in integer-1 or identifier-2, the specified number of lines is fed.

b. When a negative value is set in the identifier-2 data item, the result is not defined.

c. When zero is set in integer-1 or identifier-1, no line feed is made.

d. When mnemonic-name-1 is written, lines are skipped to channel 1 to 12 or line feed is suppressed according to the function name assigned to mnemonic-name-1. mnemonic-name-1 is assigned to the function name as shown below.

Function name SLC suppresses line feed.

Function names CHANNEL-01 to CHANNEL-12 skip lines to channel 1 to 12, respectively. Channel 1 moves to new page.

Function name CTL writes a control record that defines the page format. For writing the control record, see general rule 19).

e. The BEFORE phrase performs line feed according to rules (a) to (d), before writes lines.

f. The AFTER phrase performs line feed according to rules (a) to (d), after writes lines.

g. When the ADVANCING PAGE phrase is written and the LINAGE clause is written in the file description entry related to record-name-1, the statement writes a record to the logical page before (BEFORE) or after (AFTER) the column is set to the first line of the body in the next logical page specified in the LINAGE clause.

h. Otherwise, the statement writes a record to the physical page before (BEFORE) or after (AFTER) the column is set to the next physical page.

i. When the AFTER phrase is written in the WRITE statement for the print file with the FORMAT clause, the printer sets the column on the line defined in the screen form descriptor.

DS Sun Win 16. The results obtained by executing a WRITE statement with an ADVANCING phrase for a line sequential file are not prescribed.

HP If a WRITE statement with an ADVANCING phrase is executed for a line sequential file, the control character that corresponds to the ADVANCING phrase is output to the file. If an ADVANCING phrase is written, control characters are output to the file as follows:

a. If a positive number is specified in integer-1 or identifier-2, line feed codes that are equivalent to the specified value are output to the file.

b. If a negative number is specified in identifier-2, the result is undetermined.

c. If PAGE is written, a forms feed code is output to the file.

d. If mnemonic-name-1 is written, the action taken depends on the function-name associated with mnemonic-name-1, as follows:

When the function name is SLC or CTL, the action taken is the same as when 0 is specified in integer-1 or identifier-2.

When the function name is CHANNEL-01, the action taken is the same as when PAGE is written.

When the function name is any of CHANNEL-02 to CHANNEL-12, the action taken is the same as when 1 is specified in integer-1 or identifier-2.

e. If the BEFORE phrase is written, the record-name-1 record is output to the file before a control code is output according to rules (a) to (d).

f. If the AFTER phrase is written, a record created by adding a return code to the end of the record-name-1 record is output to the file after a control code is output according to rules (a) to (d).

g. If 0 is specified in integer-1 or identifier-2, a record created by adding a return code to the beginning and end of the record-name-1 record is output to the file, regardless of the presence or absence of the BEFORE or AFTER phrase.

DS HP Sun

17. The WRITE statement for the line sequential file may convert a part of data in the record to an external format. For the record conversion rules for the WRITE statement for the line sequential file, refer to the “COBOL85 User's Guide.”

18. For writing a control record, the WRITE statement that satisfies the following rules must be executed:

    a. In the WRITE statement for the print file having the FORMAT clause, record-name-1 must be the record specified in the CONTROL RECORDS clause.

    b. In the WRITE statement for the print file without the FORMAT clause, mnemonic-name-1 must be the mnemonic name corresponding to the function name CTL.

    For the format of the control record, refer to the “COBOL85 User's Guide.”

19. The WRITE statement for the control record must be followed by the WRITE statement having the AFTER ADVANCING PAGE phrase. The WRITE statement for the control record sets the attribute specified for the control record as the next page attribute. The page attribute remains unchanged unless the next control record is written.

20. When the CHARACTER TYPE or PRINTING POSITION clause is written in the record-name-1 record description entry or identifier-1 data description entry, the record is written according to the rules in the following table.

FROM phrase specified or not	C.T or P.P clause specified in record-name-1 record description entry	C.T or P.P clause specified in identifier-1 data description entry	Rules applied
Not specified	Not specified	-	The statement follows the C.T and P.P clause written in the record-name-1 record description entry
Specified	Not specified Specified	Specified	The statement follows the C.T and P.P clause written in identifier-1 data description entry
	Specified	Not specified	The statement follows the C.T and P.P clause written in record-name-1 record description entry are ignored.

C.T clause: CHARACTER TYPE clause
P.P clause: PRINTING POSITION clause

21. If an exception condition occurred during execution of the WRITE statement for the file without LINAGE clause or if an exception condition other than the page at end condition occurred during execution of the WRITE statement for the file having the LINAGE clause, the statement performs the following processing sequentially:

    a. Sets the I-O status of the file related to record-name-1.

    b. Executes the USE AFTER STANDARD EXCEPTION procedure related to the file related to record-name-1 when written, then passes control according to the USE statement rules. If no USE AFTER EXCEPTION procedure is written and the FILE STATUS clause is specified in the file related to record-name-1, the statement passes control to the end of the WRITE statement. If neither USE AFTER EXCEPTION procedure nor FILE STATUS clause is specified, the execution result is not defined.

22. If no exception condition occurred during execution of the WRITE statement, the procedure performs the following processing sequentially:

    a. Sets the file description entry of the file related to record-name-1.

    b. Writes the records.

    c. When NOT AT END-OF-PAGE phrase is specified, the procedure passes control to imperative-statement-2. After execution of imperative-statement-2, it passes control to the end of the WRITE statement. However, if a procedure branch or condition statement that explicitly passes control is executed in imperative-statement-2, the procedure passes control according to the rules of the statement. If the NOT AT END-OF-PAGE phrase is omitted, it passes control to the end of the WRITE statement.

Rules of Page at End and Page Overflow Conditions in the File Having the LINAGE Phrase

1. The imperative-statement-1 is executed if print page presentation reaches the logical end during execution of the WRITE statement with the END-OF-PAGE phrase. The logical end is specified in the LINAGE clause in the file description entry related to record-name-1.

2. When the WRITE statement is executed on the file for which the LINAGE clause is specified in the file description entry, a page at end condition or page overflow condition may occur. The page at end condition occurs only when the END-OF-PAGE phrase is written in the WRITE statement.

a. The page at end condition occurs during printing or line feed in the footing area of the page body. This is because the LINAGE-COUNTER value of the becomes equal to or greater than integer-2 or data-name-2 in the FOOTING phrase of the LINAGE clause after line feed is made by the WRITE statement. At this time, imperative-statement-1 with the END-OF-PAGE phrase is specified after execution of the WRITE statement.

b. The page overflow condition occurs if line feed by the WRITE statement exceeds the current page body. This is because the LINAGE-COUNTER value exceeds integer-1 or data-name-1 in the LINAGE clause after line feed is made by the WRITE statement. At this time, a record is written to the logical page before (BEFORE) or after (AFTER) the device is positioned at the first line on the body of the next logical page specified in the LINAGE clause according to the ADVANCING phrase. When the END-OF-PAGE phrase is written, the record is written, then imperative-statement-1 is executed.

c. The page overflow condition occurs if the LINAGE-COUNTER value exceeds both integer-2 or data-name-2 in the FOOTING phrase of the LINAGE clause and integer-1 or data-name-1 in the LINAGE clause after line feed is made by the WRITE statement.

Rules for WRITE Statements Corresponding to FORMAT Clause Print Files

1. When a form descriptor name is set in the form descriptor name field of the control record and then the WRITE statement is executed for the control record, the next page will be a fixed-format page.

2. When a blank is set in the form descriptor name field of the control record and then the WRITE statement is executed for the control record, the next page will be an undefined-format page. Even immediately after the print file having the FORMAT clause is opened, the next page will be assumed to be an undefined-format page.

3. For writing a chart record, a form descriptor name must be set in the data item specified in the FORMAT clause. At this time, the chart record is output on the fixed-format page defined with the form descriptor name and edited according to the form descriptor.

4. When writing a line record, a blank must be set in the data item specified in the FORMAT clause.

5. For writing a record to the fixed-format page, a partition name or item group name must be set in the data item specified in the GROUP clause.

6. If the form descriptor name set in the FORMAT clause data item differs from the current value when writing a chart record, the WRITE statement to be executed must have the AFTER ADVANCING PAGE phrase.

7. For writing a chart record at the head of the page, the ADVANCING phrase must be specified together with the AFTER ADVANCING PAGE phrase.

8. After the chart record has been written, the line is assumed to be fed to the last line defined in the form descriptor.

9. For writing a chart record to the floating partition, an exception condition may occur if the overall chart record could not be output at the output position after the ADVANCING phrase has been evaluated.

WRITE Statement (Relative and Indexed Files)

This statement writes records to a file.

[Format]

```
    WRITE record-name-1
 [FROM identifier-1]
 [INVALID KEY imperative-statement-1]
 [NOT INVALID KEY imperative-statement-2]
 [END-WRITE]
```

Syntax Rules

1. The record-name-1 must be the name of the record defined in the file section of the data division. It can be qualified by the file name.

2. When a data item is specified in identifier-1, the same area must not be specified for record-name-1 and identifier-1. A function-identifier specified in identifier-1 must be an alphanumeric function.

3. If no USE AFTER STANDARD EXCEPTION procedure of the file related to record-name-1 is written, the INVALID KEY phrase must be specified. In this compiler, however, both the USE AFTER STANDARD EXCEPTION procedure and the INVALID KEY phrase can be omitted.

General Rules

Rules Common to Both Relative and Indexed Files

1. The file related to record-name-1 must have been opened in output, input-output mode, or extended mode before the WRITE statement is executed.

2. After the WRITE statement has been executed successfully, the record of record-name-1 is not accessible in the record area. However, the record may still be accessible after execution of the WRITE statement when the file related to record-name-1 is specified in the SAME RECORD AREA clause of the I-O control paragraph. The program can use record-name-1.

3. The WRITE statement having the FROM phrase produce the same result as the following statements executed in the provided order:

   a. Statements according to the MOVE statement rules

      MOVE identifier-1 TO record-name-1

   b. WRITE statement without FROM phrase

4. After the WRITE statement has been executed successfully, data in the area of record-name-1 may not be accessible any more, except when the file related to record-name-1 is specified in the SAME RECORD AREA clause. Data in the area of identifier-1 is accessible.

5. The file position indicator remains unchanged after the WRITE statement has been executed.

6. The value of the I-O status of the file related to record-name-1 is updated after execution of the WRITE statement.

7. The record is passed to the i-o-control system after execution of the WRITE statement.

8. Data in record-name-1 must not exceed the maximum record length of the file related to record-name-1. Data in record-name-1 must not be less than the minimum record length of the file related to record-name-1. If these conditions are not satisfied, the WRITE statement will fail and no record will be written. At this time, the record area of the file related to record-name-1 remains unchanged. The I-O status of the file related to record-name-1 is set with a value indicating the cause of the state.

   Note that the compiler executes the WRITE successfully when a record smaller than the minimum record size is written.

9. If an invalid key condition occurs during execution of the WRITE statement, the statement may fail. The record area of the file related to record-name-1 remains unchanged. After a value indicating invalid key condition is set in the I-O status of the file related to record-name-1, control is passed according to the rules in the section titled “INVALID KEY phrase.”.

10. If an exception condition occurs other than an invalid key condition during execution of the WRITE statement, the file description entry of the file related to record-name-1 is set. Then, control is passed according to the rules in the section titled “INVALID KEY phrase.”.

11. If no exception condition occurs including invalid key condition during execution of the WRITE statement, the statement performs the following processing sequentially:

    a. Sets the I-O status of the file related to record-name-1.

    b. Writes the record.

c. Passes control according to the rules in the section titled “INVALID KEY phrase.”

12. When records are added to a file opened in extended and shared mode with the WRITE statement for the file connector of two or more files, the order of record addition may be undefined.

13. When AUTOMATIC is specified in the LOCK MODE phrase in the file control entry related to record-name-1, the existing record will be unlocked after the WRITE statement has been executed successfully.

Rules for Relative Files

1. The file position indicator remains unchanged after execution of the WRITE statement.

2. When the WRITE statement is executed for the sequential access file opened in output mode, it automatically sets the relative record number and writes record-name-1. Relative record numbers are given from the first record in an ascending order. When the RELATIVE KEY phrase is written in the ACCESS MODE clause for the file related to record-name-1, the i-o-control system sets the relative record number of the written record in the relative key item.

3. When the WRITE statement is executed for the random or dynamic access file opened in output mode, the relative record number of the record to be written must have been set in the relative key item before WRITE is executed.

4. When the WRITE statement is executed for the file opened in extended mode, the file must be a sequential access file. The WRITE statement adds record-name-1 to the file. The relative record number of the record in the first WRITE

statement is the maximum relative record number in the file plus 1.

The relative number is incremented by one when the next record is written. When the RELATIVE KEY phrase is written in the ACCESS MODE clause of the file related to record-name-1, the relative number of the written record is set in the relative key item.

5. When the WRITE statement is executed for the file opened in input-output mode, the file must be a random or dynamic access file. Before execution of WRITE, the relative record number of the record to be written must have been set in the relative key item. WRITE inserts record-name-1 into the file.

6. An invalid key condition may occur:

   a. If a record having the same relative key item already exists in a random or dynamic access file when the WRITE statement is executed for the file.

   b. If the statement attempts to write the record over the file segment externally defined explicitly or implicitly.

   c. If the number of significant digits of the relative record number exceeds the size of the relative key item.

Rules for Indexed Files

1. The file position indicator remains unchanged after execution of the WRITE statement. However, the file position indicator is not defined:

   a. If the DUPLICATES phrase is written in the RECORD KEY clause of the file related to record-name-1

   b. If the file position indicator is set in the START statement with the REVERSED ORDER phrase

2. Before the WRITE statement is executed, a required value must have been set in the prime record key item.

3. If no DUPLICATES phrase is written in the RECORD KEY clause of the file related to record-name-1, the value of the prime record key item must be unique through all records in the file. If DUPLICATES is written, the value may not be unique.

4. The WRITE statement writes data in the record area of record-name-1. The i-o-control system uses the value of the record key item to later call the record according to any record key.

5. For writing a record to the sequential access file, a value must be set in the prime record key item according to the rules listed below.

    a. When the file is opened in output mode.

       When the DUPLICATES phrase is omitted in the RECORD KEY clause, a value must be set in the prime record key item so that values of the prime record keys are in ascending order of the prime record key items. If DUPLICATES is specified in the RECORD KEY clause, the same value as in the prime record key of the previously written record can be set.

    b. When the file is opened in extended mode.

       If the DUPLICATES phrase is omitted in the RECORD KEY clause, a value greater than the maximum prime record key in the file must be set in the prime record key item. When DUPLICATES is written in the RECORD KEY clause, the same value as of the maximum prime record key in the file can be set.

6. When writing records to a random or dynamic access file, the records may be written in any sequence.

7. When the DUPLICATES phrase is written in the ALTERNATE RECORD KEY clause of the file related to record-name-1, the value of the alternate record key item need not be unique through file records. At this time, records are stored so that they can be read with a READ statement having the NEXT phrase in the order they are written.

8. An invalid key exception occurs:

   a. In the WRITE statement for a sequential access file having the RECORD KEY clause with no DUPLICATES phrase, if the record key value is equal to or less than that of the previous record.

   b. In the WRITE statement for a sequential access file having the RECORD KEY clause with the DUPLICATES phrase, if the record key value is less than that of the previous record.

   c. In the WRITE statement for a random or dynamic access file having the RECORD KEY clause with no DUPLICATES phrase, if the record key value is equal to that of a record in the file.

   d. In the WRITE statement for a random or dynamic access file having the RECORD KEY clause with no DUPLICATES phrase, if the alternate record key value is equal to that of a record in the file.

   e. If the statement attempts to write records over the file segment externally defined explicitly or implicitly.

WRITE Statement (Presentation File)

This statement writes records to a file.

[Format]

```
    WRITE record-name-1 [FROM identifier-1]
 [END-WRITE]
```

Syntax Rules

1. The record-name-1 must be the name of the record defined in the file section of the data division. It can be qualified by the file name.
2. When a data item is specified in identifier-1, the same area must not be specified for record-name-1 and identifier-1. A function-identifier specified in identifier-1 must be an alphanumeric function.

General Rules

1. The file related to record-name-1 must have been opened in output, input-output mode, or extended mode before the WRITE statement is executed.
2. After the WRITE statement has been executed successfully, the record of record-name-1 is not accessible in the record area. However, the record remains accessible after execution of the WRITE statement if the file related to record-name-1 is specified in the SAME RECORD AREA clause of the I-O control paragraph. The program can use record-name-1.

3. The WRITE statement having the FROM phrase produces the same result as the following statements executed in the provided order:

   a. Statement according to the MOVE statement rules

      MOVE identifier-1 TO record-name-1

   b. Same WRITE statement but without FROM phrase

4. After the WRITE statement has been executed successfully, data in the area of record-name-1 may not be accessible any more, except when the file related to record-name-1 is specified in the SAME RECORD AREA clause. Data in the area of identifier-1 is accessible.

5. The value of the I-O status of the file related to record-name-1 is updated after execution of the WRITE statement.

6. The record is passed to the i-o-control system after execution of the WRITE statement.

General Rules for Functions

Functions call a specific process using several data as arguments, then they store the process result (function value) in a temporary data item to return it to the program.

Function Call Format

The function name must be entered in the format shown below. This format is called function-identifier.

The function name is the name of a function. A function-name is a COBOL word.

The argument is a value used to determine the function value. For some functions, the range of arguments can be limited. The number, sequence, and value of arguments depends on each of functions.

The function-identifier can be written in the procedure division only.

[Format]

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

Types of Arguments

Arguments can be an identifier, arithmetic expression, or literal. Data that can be specified in arguments is determined by the type of argument.

There are five types of arguments: alphabetic, alphanumeric, numeric character, integer and national character. Arguments must be specified according to the rules listed below.

1. When the argument type is alphabetic, the argument must be an alphabetic class item or a nonnumeric literal containing alphabetic characters only. The argument size can also be used to determine the function value.
2. When the argument type is alphabetic, the argument must be an alphabetic data item, alphanumeric data item, alphanumeric edited data item, or nonnumeric literal. The argument size can also be used to determine the function value.
3. When the argument type is numeric, the argument must be an arithmetic expression. The value of the arithmetic expression (including sign) is used to determine the function value.
4. When the argument type is integer, the argument must be an arithmetic expression which always has an integer value. The value of the arithmetic expression (including sign) is used to determine the function value.
5. If the argument type is national, an elementary national data item must be specified. The argument size may be used to determine a function value.

Some functions may use other data items or literal.

The range of argument values depends on each of functions. If a value over a defined range is set in an argument, the function value may not be determined.

Rules Applied for Specifying a Table as Argument

When you want to write all the table elements of a dimension to repeat an argument, you can write the data-name containing ALL in subscript, instead of writing individual subscripted data-names. ALL specified as subscript handles all table elements of the dimension as if they are specified from left to right in the order of occurrence.

See the table example defined below. In this example, (a) and (b) as well as (c) and (d) will have the same value.

```
01  TBL.
   02  A  OCCURS  3.
     03  B  OCCURS  4.
       04  C  PIC  X.
```

```
(a)  FUNCTION  MAX  (A(1) A(2) A(3))
(b)  FUNCTION  MAX  (A(ALL))
(c)  FUNCTION  MAX  (C(1 1) C(1 2) C(1 3) C(1 4))
(d)  FUNCTION  MAX  (C(1 ALL))
```

When subscript ALL is for the OCCURS clause having the DEPENDING ON phrase, the upper limit of ALL follows the data-name having the DEPENDING ON phrase.

After evaluation of subscript ALL, at least one table element must be referenced. Otherwise, no function value is determined.

Function Types

The type of temporary data item containing the function value is called function type. The location of the function-identifier is determined by the function type.

There are five function types: alphanumeric, numeric, integer, pointer-data, and national. These functions are called depending on the function type: alphanumeric function, numeric function, integer function, pointer-data function, and national function.

Alphanumeric Function

The category and class of the temporary data item for the alphanumeric function are alphanumeric. The data item is used for display.

The alphanumeric function-identifier can be written in alphanumeric data items only in the procedure division. Note that it cannot be written in the receiving side item.

The size of the alphanumeric function value depends on each function.

Numeric Function

The category and class of the temporary data item for the numeric function are numeric. The data item has a sign.

The numeric function-identifier can be written only in arithmetic expressions only. Note that it cannot be written in integer items.

The attribute and precision of the numeric function value depends on each function.

Integer Function

The category and class of the temporary data item for the integer function are numeric. The data item has a sign.

The integer function-identifier can be written in integer items only in arithmetic expressions.

The attribute and precision of the integer function value depends on each function.

Pointer Data Function

The temporary data item for the pointer data function is used for pointer data.

The pointer data function-identifier can be written in pointer data items only in the procedure division. Note that it cannot be written in the receiving side item.

National Function

The class and category of a national function are national.

The function-identifier of a national function can only be written in a place where a national item can be written in the Procedure Division. However, it cannot be written in the receiving item.

The number of character positions for the function value of a national function depends on the function type.

Functions

This section describes each function.

ACOS Function

This function returns the arc cosine of argument-1.

[Format]

<u>FUNCTION</u> <u>ACOS</u> (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be within the following range:

$$-1 \leq \text{argument-1} \leq +1$$

Function Value

1. The function value is an approximate value of the arc cosine of argument-1. The unit is radian.
2. The range of the function value must be:

$$0 \leq \text{function value} \leq \pi$$

Function Type

The function type is numeric

.

ADDR Function

This function returns the head address of argument-1.

[Format]

<u>FUNCTION</u> <u>ADDR</u> (argument-1)

Argument

argument-1 must not be an internal Boolean item.

Function Value

The function value is the head address of argument-1.

Function Type

The function type is pointer data.

ADDR Function Specific Rules

The ADDR function can be written in the following places only:

- Pointer qualifier
- DISPLAY statement
- Sending side in the MOVE statement
- Relation condition in the IF and EVALUATE statements

If the ADDR function is written in the DISPLAY statement, the head address of argument-1 is displayed in hexadecimal notation.

ANNUITY Function

This function returns a uniform payment when argument-1 is the interest rate, argument-2 is the period of uniform payment, and the capital is 1.

[Format]

FUNCTION ANNUITY (argument-1 argument-2)

Arguments

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be zero or greater.
3. The type of argument-2 must be integer.
4. The value of argument-2 must be a positive integer.

Function Value

The function value is:

- When argument-1 is zero, it is an approximate value of the following arithmetic expression:

  1/argument-2

- When argument-1 is not zero, it is an approximate value of the following arithmetic expression:

  argument-1/(1 - (1 + argument-1) ** (- argument-2))

Function Type

The function type is numeric.

ASIN Function

This function returns the arc sign of argument-1.

[Format]

FUNCTION ASIN (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be within the following range:

$$-1 \leq \text{argument-1} \leq +1$$

Function Value

1. The function value is the arc sin of argument-1. The unit is radian.
2. The range of the function value must be:

$$-\pi/2 \leq \text{function value} \leq +\pi/2$$

Function Type

The function type is numeric.

ATAN Function

This function returns the arc tangent of argument-1.

[Format]

FUNCTION ATAN (argument-1)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is an approximate value of the arc tangent of argument-1. The unit is radian.
2. The range of the function value must be:

$$-\pi/2 < \text{function value} < +\pi/2$$

Function Type

The function type is numeric.

CAST-ALPHANUMERIC Function

This function returns the national or national edited character-string in argument-1 as an alphanumeric character-string. The function type is alphanumeric.

[Format]

FUNCTION CAST-ALPHANUMERIC (argument-1)

Argument

1. The length of argument-1 must be at least one character, and its class must be national.

Function Value

1. The function value is the same character-string as argument-1 because each national character is regarded as an alphanumeric character.
2. The number of characters in the function value is double the number of characters in argument-1.

CHAR Function

This function returns the character at argument-1 position in the collating sequence.

[Format]

FUNCTION CHAR (argument-1)

Argument

1. The type of argument-1 must be integer.
2. Assume n as the maximum precedence value. The value of argument-1 must be from 1 to n.

Function Value

1. The function value is the character of argument-1 position in the collating sequence. The collating sequence is defined in the ALPHABET clause in the special-names paragraph. If the ALPHABET clause is omitted, the function value is determined according to the collating sequence in the computer character set.
2. If two or more characters are found at argument-1 position in the collating sequence, the first literal character specified in the sequence of the ALPHABET clause is returned as function value.

Function Type

The function type is alphanumeric.

COS Function

This function returns the cosine of argument-1.

[Format]

<u>FUNCTION</u> <u>COS</u> (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The unit of argument-1 must be radian.

Function Value

1. The function value is an approximate value of the cosine of argument-1.
2. The range of the function value must be:

$$-1 \leq \text{function value} \leq +1$$

Function Type

The function type is numeric.

CURRENT-DATE Function

This function returns the current date and time.

[Format]

FUNCTION CURRENT-DATE

Function Value

The function value is 21 alphanumeric characters indicating the date and time when the function is to be executed as well as the time difference from the standard time. The meaning of each character in the function value is listed below.

Character Position	Value
1 to 4	4-digit number indicating the year.
5 to 6	2-digit number indicating the month. The value ranges from 01 to 12.
7 to 8	2-digit number indicating the day. The value ranges from 01 to 31.
9 to 10	2-digit number indicating hours. The value ranges from 00 to 23.
11 to 12	2-digit number indicating minutes. The value ranges from 00 to 59.
13 to 14	2-digit number indicating seconds. The value ranges from 00 to 59.
15 to 16	2-digit number indicating 1/100 seconds. The value ranges from 00 to 99. (*1)
17	Sign indicating whether the local time (time indicated in character positions 1 to 16) is ahead or behind the Greenwich mean time. A minus (-) sign indicates that the local time is behind Greenwich mean time, and a plus (+) sign indicates that it is ahead. (*2)
18 to 19	2-digit number indicating the number of hours ahead or behind Greenwich mean time. The value ranges from 00 to 12 when character position 17 is “-”; and ranges from 00 to 13 when it is “+.” (*2)
20 to 21	2-digit number indicating the number of minutes ahead or behind Greenwich mean time. The value ranges from 00 to 59. (*2)

*1 Character positions 15 and 16 are set to 00 if the system cannot read 1/100 seconds.

*2 Character positions 17 to 21 are set to 00000 if the system cannot read the time difference between the local time and Greenwich mean time.

Function Type

The function type is alphanumeric.

DATE-OF-INTEGER Function

This function converts and returns argument-1 to the standard format date.

[Format]

<u>FUNCTION</u> <u>DATE-OF-INTEGER</u> (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be a positive integer indicating a serial day starting from January 1, 1601.

Function Value

1. The function value is the standard format date equivalent to the serial day specified in argument-1.
2. The function value is set in the format of yyyymmdd; where yyyy is the year, mm is the month, and dd is the day.

Function Type

The function type is integer.

DAY-OF-INTEGER Function

This function converts and returns argument-1 to the year and day format.

[Format]

FUNCTION DAY-OF-INTEGER (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be a positive integer indicating a serial day starting from January 1, 1601.

Function Value

1. The function value is the year and day format date equivalent to the serial day specified in argument-1.
2. The function value is set in the format of yyyyddd; where yyyy is the year and ddd is the serial day in the year.

Function Type

The function type is integer.

FACTORIAL Function

This function returns the factorial of argument-1.

[Format]

FUNCTION FACTORIAL (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be zero or a positive integer.

Function Value

The function value is:

- When argument-1 is zero, it is 1.
- When argument-1 is a positive numeric value, it is the factorial of argument-1.

Function Type

The function type is integer.

INTEGER Function

This function returns the maximum integer equal to or less than argument-1.

[Format]

FUNCTION INTEGER (argument-1)

Argument

The type of argument-1 must be numeric.

Function Value

The function value is the maximum integer equal to or less than argument-1. For example, when the value of argument-1 is +1.5, the function value is +1. When argument-1 is -1.5, the function value is -2.

Function Type

The function type is integer.

INTEGER-OF-DATE Function

This function converts and returns argument-1 (standard format date) to a serial day.

[Format]

FUNCTION INTEGER-OF-DATE (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be an eight-digit integer in the format of yyyymmdd.
3. argument-1 must be in the standard format using the following arithmetic expression.

   (yyyy * 10000) + (mm * 100) + dd

   The values yyyy, mm, and dd must follow the following rules:

- yyyy is the numeric value indicating the year. It must be greater than 1600.
- mm is the numeric value indicating the month. It must be in the range from 1 to 12.
- dd is the numeric value indicating the day. It must be in the range from 1 to 31.
- A combination of yyyy, mm, and dd must be a value indicating a proper date.

Function Value

The function value is an integer indicating a serial day equivalent to argument-1 (standard format date). The serial day begins from January 1, 1601.

Function Type

The function type is integer.

INTEGER-OF-DAY Function

This function converts and returns argument-1 (date format) to a serial day.

[Format]

FUNCTION INTEGER-OF-DAY (argument-1)

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be a seven-digit integer in the format of yyyyddd.
3. argument-1 must be in the year and day format using the following arithmetic expression.

   (yyyy * 1000) + ddd

   The values yyyy and ddd must follow the following rules:

   a. yyyy is the numeric value indicating the year. It must be greater than 1600.
   b. ddd is the numeric value indicating the serial day in the year. It must be in the range from 1 to 366.
   c. A combination of yyyy and ddd must be a value indicating a proper date.

Function Value

The function value is an integer indicating a serial day equivalent to argument-1 (year and day format date). The serial day begins from January 1, 1601.

Function Type

The function type is integer.

INTEGER-PART Function

This function returns the integer part of argument-1.

[Format]

```
FUNCTION INTEGER-PART (argument-1)
```

Argument

The type of argument-1 must be numeric.

Function Value

The function value is:

1. When the value of argument-1 is 0, it is 0.
2. When the value of argument-1 is an integer value, it is the maximum integer equal to or less than argument-1. For example, when argument-1 is +1.5, the function value is +1.
3. When the value of argument-1 is a negative value, it is the minimum integer equal to or greater than argument-1. For example, when argument-1 is -1.5, the function value is -1.

Function Type

The function type is integer.

LENG Function

This function returns the length (number of bytes) of argument-1.

[Format]

FUNCTION LENG (argument-1)

Argument

The type of argument-1 must be a data item other than internal Boolean item, nonnumeric literal, hexadecimal nonnumeric literal, or national literal.

Function Value

The function value is the size (number of bytes) of argument-1.

Function Type

The function type is integer.

LENG Function Specific Rules

The LENG function can be written in the following places only:

- Source data item of MOVE statement
- Arithmetic expression
- Integer numeric literal items in the procedure division

LENGTH Function

This function returns the length (number of character positions or national character positions) of argument-1

[Format]

FUNCTION LENGTH (argument-1)

Argument

1. The type of argument-1 must be a nonnumeric literal, hexadecimal nonnumeric literal, national literal, or any data item.

2. If the OCCURS clause having the DEPENDING ON phrase is specified in an data item subordinate to argument-1, the function value is determined using the data-name having the DEPENDING ON phrase at evaluation of the LENGTH function.

Function Value

1. The function value is listed in the table below.

Argument-1	**Function Value**
Nonnumeric literal or hexadecimal nonnumeric literal	Number of character positions of argument-1 (number of bytes)
National literal	Number of national character positions of argument-1 (number of national characters)
Alphabetic, alphanumeric, alphanumeric edited, or numeric edited data items	Number of character positions of argument-1 (number of bytes)
Numeric, group, index, or pointer-data item	Number of bytes in storage area used by argument-1
National and national edited data items	Number of national character positions of argument-1 (number of national characters)
External Boolean data item	Number of Boolean character positions of argument-1 (number of bytes)
Internal Boolean data item	Number of bits of argument-1

2. When argument-1 is a group item including variable occurrence data items, the function value is determined by evaluating the data-name having the DEPENDING ON phrase in the OCCURS clause in the variable occurrence data item. Evaluation follows the rules for when the OCCURS clause is specified in the source data item.

3. When an implicit FILLER item exists, the function value contains the number of characters of the items.

Function Type

The function type is integer.

LOG Function

This function returns the log of argument-1 using e as the base.

[Format]

FUNCTION LOG (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be positive.

Function Value

The function value is an approximate value of the log (natural log) of argument-1 using e as the base.

Function Type

The function type is numeric.

LOG10 Function

This function returns the log of argument-1 using 10 as the base.

[Format]

FUNCTION LOG10 (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be positive.

Function Value

The function value is an approximate value of the log (common log) of argument-1 using 10 as the base.

Function Type

The function type is numeric.

LOWER-CASE Function

This function returns lowercase alphabetic characters equivalent to uppercase alphabetic characters in argument-1.

[Format]

FUNCTION LOWER-CASE (argument-1)

Argument

1. The type of argument-1 must be alphabetic or alphanumeric.
2. The length of argument-1 must be 1 character or more.

Function Value

1. The function value is a character string of lowercase alphabetic characters which have been converted from uppercase alphabetic characters. In other words, the result is the same as argument-1 except that the uppercase alphabetic characters have been replaced.
2. The length of the function value is the same as of argument-1.

Function Type

The function type is alphanumeric.

MAX Function

This function returns the maximum value of an argument-1.

[Format]

FUNCTION MAX ({argument-1}...)

Argument

1. The type of argument-1 is optional.
2. A combination of argument-1 elements must be one in the function type table shown below.

Function Value

1. The function value is the maximum value in the list of argument-1.
2. If two or more values in argument-1 have the same maximum value, the high order end is returned as function value.
3. Comparison for determining the maximum value follow the rules for relation conditions.
4. When the function type is alphanumeric, the length of the function value is the same as of argument-1 having the maximum value.

Function Types

The table below lists the function types.

Type of argument-1	**Function Type**
All alphabetic or alphanumeric characters	Alphanumeric character
All integers	Integer
All numeric or mix of numeric characters and integers	Numeric character

MEAN Function

This function returns the arithmetic mean value of an argument-1.

[Format]

FUNCTION MEAN ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

The function value is the arithmetic mean value of the list of argument-1. In other words, the value will be the sum of the list of argument-1 elements divided by the number of elements in argument-1.

Function Type

The function type is numeric.

MEDIAN Function

This function returns the median of an argument-1.

[Format]

FUNCTION MEDIAN ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is the median when the list of argument-1 is sorted in ascending order. The median is determined as follows:

   a. When the number of elements of argument-1 is odd-numbered, the middle value after sorting is returned.

   b. When the number of elements of argument-1 is even-numbered, the mean value of the two middle values after sorting is returned.

2. Comparison for sorting must follow the relation condition rules.

Function Type

The function type is numeric.

MIDRANGE Function

This function returns the arithmetic mean value of the minimum and maximum values of an argument-1.

[Format]

<u>FUNCTION</u> <u>MIDRANGE</u> ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

The function value is the arithmetic mean value of the maximum and minimum values in the list of argument-1. In other words, the value will be the sum of the minimum and maximum values divided by two. Comparison for determining the maximum and minimum values follow the relation condition rules.

Function Type

The function type is numeric.

MIN Function

This function returns the minimum value of an argument-1.

[Format]

FUNCTION MIN ({argument-1}...)

Argument

1. The type of argument-1 must be numeric.
2. A combination of argument-1 must be either of the function types listed in the table below.

Function Value

1. The function value is the minimum argument-1 in the list of argument-1 elements.
2. If more than one argument-1 has the same minimum value, the high order end argument is returned.
3. Comparison for determining the minimum value follows the relation condition rules.
4. When the function type is alphanumeric, the length of the function value is the same as that of argument-1 having the minimum value.

Function Type

The table below lists the function types.

Type of argument-1	**Function Type**
All alphabetic or alphanumeric characters	Alphanumeric character
All integer	Integer
All numeric or mix of numeric characters and integers	Numeric character

MOD Function

This function returns the integer value of argument-1 using argument-2 as modulus.

[Format]

FUNCTION MOD (argument-1 argument-2)

Argument

1. The type of argument-1 and argument-2 must be integer.
2. The value of argument-2 must not be zero.

Function Value

1. The function value is an integer value using argument-2 as modulus. It is determined using the following arithmetic expression:

   argument-1 - (argument-2 * FUNCTION INTEGER (argument-1 / argument-2))

2. Some function value examples are shown below.

argument-1	argument-2	Function Value
11	5	1
-11	5	4
11	-5	-4
-11	-5	-1

Function Type

The function type is integer.

NATIONAL Function

This function replaces the COBOL character set with national characters except the national characters in argument-1. The type of this function is national.

[Format]

FUNCTION NATIONAL (argument-1)

Argument

The length of argument-1 must be at least one character, and its class must be alphabetic, alphanumeric, or numeric. Note that only an zoned decimal item is accepted when the class is numeric.

Function Value

1. The function value is the same character-string as argument-1, except that the COBOL character set, other than national characters, is replaced with the corresponding national characters.
2. The number of characters in the function value is the same as the number of characters in argument-1.
3. If a signed zoned decimal item is specified in argument-1, the sign is ignored.
4. If an invalid character code is specified in argument-1, the result is unpredictable.

NUMVAL Function

This function converts argument-1 (character string in numeric literal format) to a numeric value and returns it.

[Format]

FUNCTION NUMVAL (argument-1)

Argument

1. argument-1 must be a nonnumeric literal or alphanumeric data item. The value of argument-1 must be in either of the formats shown below. The string-of-blanks indicates a sequence of one or more blanks, and the numeric-string indicates a sequence of one or more numeric values.

(a)

$$\text{[string-of-blanks]}\begin{bmatrix} + \\ \\ - \end{bmatrix}\text{ [string-of-blanks]}\left\{\begin{array}{l}\text{numeric-string [.[numeric-string]]}\\ \text{. numeric-string}\end{array}\right\}\text{ [string-of-blanks]}$$

(b)

$$\text{[string-of-blanks]}\left\{\begin{array}{l}\text{numeric-string [. [numeric-string]]}\\ \text{. numeric-string}\end{array}\right\}\text{ [string-of-blanks] }\begin{bmatrix} + \\ - \\ \text{CR} \\ \text{DB} \end{bmatrix}\text{[string-of-blanks]}$$

2. The sum of digits of argument-1; that is, the number of digits of numeric strings must be 18 or less.
3. When the DECIMAL-POINT IS COMMA clause is written in the special-names paragraph, the decimal point function is replaced with the comma function in argument-1. At this time, a comma must be entered instead of the period (.) to indicate the decimal point.

Function Value

The function value is the value of argument-1.

Function Type

The function type is numeric.

NUMVAL-C Function

This function converts argument-2 (currency symbol) or the character string in argument-1 containing a comma to a numeric value and returns it.

[Format]

FUNCTION NUMVAL-C (argument-1 [argument-2])

Argument

1. argument-1 must be a nonnumeric literal or alphanumeric data item. The value of argument-1 must be either of the formats shown below. The string-of-blanks indicates a sequence of one or more blanks, and numeric-string indicates a sequence of one or more numeric values. A currency sign indicates a character string (more than one character) identical to argument-2.

(a)

$$[\text{string-of-blanks}] \begin{bmatrix} + \\ \\ - \end{bmatrix} [\text{string-of-blanks}]\ [\text{currency-sign}]\ [\text{string-of-blanks}]$$

$$\begin{Bmatrix} \text{numeric-string [,numeric-string] ... [. [numeric-string]]} \\ \text{. numeric-string} \end{Bmatrix}$$

(b)

$$[\text{string-of-blanks}]\ [\text{currency-sign}]\ [\text{string-of-blanks}]$$

$$\begin{Bmatrix} \text{numeric-string [,numeric-string] ... [. [numeric-string]]} \\ \text{. numeric-string} \end{Bmatrix}$$

$$[\text{string-of-blanks}] \begin{bmatrix} + \\ - \\ \text{CR} \\ \text{DB} \end{bmatrix} [\text{string-of-blanks}]$$

2. The sum of digits of argument-1; that is, the number of digits of numeric strings must be 18 or less.

3. When the DECIMAL-POINT IS COMMA clause is written in the special-names paragraph, the decimal point function is replaced with the comma function in argument-1. At this time, a comma must be entered instead of the period (.) to indicate the decimal point.

4. argument-2 must be a nonnumeric literal or alphanumeric data item.

5. argument-2 must be set with a currency sign. If argument-2 is omitted, the currency sign specified in the CURRENCY SIGN clause in the special-names paragraph is assumed. If the CURRENCY SIGN clause is omitted, the currency sign in the COBOL character set is assumed.

Function Value

The function value is the numeric value of argument-1.

Function Type

The function type is numeric.

ORD Function

This function returns the order of argument-1 in the collating sequence.

[Format]

FUNCTION ORD (argument-1)

Argument

1. The type of argument-1 must be alphabetic or nonnumeric.
2. The length of argument-1 must be one character.

Function Value

The function value is the order of argument-1 elements in the collating sequence. The collating sequence is defined in the ALPHABET clause in the special-names paragraph. If the ALPHABET clause is omitted, the function value is determined according to the collating sequence in the computer's character set.

Function Type

The function type is integer.

ORD-MAX Function

This function returns the position of the argument-1 which has the maximum value.

[Format]

FUNCTION ORD-MAX ({argument-1}...)

Argument

1. The type of argument-1 is optional.
2. The combination of argument-1 types must be one of the following:
   a. All alphabetic characters
   b. All alphanumeric characters
   c. All integers
   d. All numeric characters
   e. Mix of numeric characters and integers

Function Value

1. The function value is the position of argument-1 having the maximum value in the list of argument-1 elements. For example, when the second argument-1 from the left has the maximum value in the list of argument-1 elements, 2 is returned.
2. If more than one argument-1 has the same maximum value, the one written furthest to the left is the maximum value. Accordingly, the function becomes the value of that position.
3. Comparison for determining the maximum value follow the relation condition rules.

Function Type

The function type is integer.ORD-MIN Function

This function returns the position of the argument-1 which has the minimum value.

[Format]

<u>FUNCTION</u> <u>ORD-MIN</u> ({argument-1}...)

Argument

1. The type of argument-1 is optional.
2. The combination of argument-1 types must be one of the following:
   a. All alphabetic characters
   b. All alphanumeric characters
   c. All integers
   d. All numeric characters
   e. Mix of numeric characters and integers

Function Value

1. The function value is the position of argument-1 having the minimum value in the list of argument-1 elements. For example, when the second argument-1 from the left has the minimum value in the list of argument-1 elements, 2 is returned.
2. If more than one argument-1 has the same minimum value, the one written furthest to the left is the minimum value.
3. Comparison for determining the minimum value follow the comparison condition rules.

Function Type

The function type is integer.

PRESENT-VALUE Function

This function returns the present value at the end of each term in the list of argument-2 using argument-1 as depreciation rate.

[Format]

FUNCTION PRESENT-VALUE (argument-1 {argument-2} ...)

Arguments

1. The type of argument-1 and argument-2 must be numeric.
2. argument-1 must be greater than -1.

Function Value

The function value is an approximate value of the sum of numeric strings configured from terms below. One term corresponds to each value in argument-2. Exponent n begins from 1 and is incremented by one for each term.

argument-2 / (1 + argument-1) ** n

Function Type

The function type is numeric.

RANDOM Function

This function returns a pseudo random value from uniform distributions.

[Format]

FUNCTION RANDOM [(argument-1)]

Argument

1. The type of argument-1 must be integer.
2. The value of argument-1 must be zero or a positive integer. It is used to generate a pseudo-random string.
3. The RANDOM function with argument-1 will generate a pseudo-random string using argument-1 as source.
4. If argument-1 is omitted in the RANDOM function first executed in the run unit, a pseudo-random string is generated using zero as source.
5. When determining a random value from the same pseudo random string after execution of the RANDOM function, argument-1 may be omitted. The value of argument-1 is valid until the next RANDOM function having argument-1 appears.

Function Value

1. The function value is any value in the current pseudo-random string.
2. The range of the function value is:

   $$0 \leq \text{function value} < 1$$

3. If the value of argument-1 (source value) is the same, the same pseudo-random string is used.

Function Type

The function type is numeric.

RANGE Function

This function returns the maximum value minus the minimum value of an argument-1.

[Format]

FUNCTION RANGE ({argument-1}...)

Argument

The type of argument-1 must be numeric or integer.

Function Value

1. The function value is the maximum value minus the minimum value of the list of argument-1.
2. Comparison for determining the maximum and minimum values follow the relation condition rules.

Function Type

The table below lists the function types:

Type of argument-1	**Function Type**
All integers	Integer
All numeric or mix of numeric characters and integers	Numeric character

REM Function

This function returns the remainder of argument-1 divided by argument-2.

[Format]

FUNCTION REM (argument-1 argument-2)

Arguments

1. The type of argument-1 and argument-2 must be numeric.
2. The value of argument-2 must not be zero.

Function Value

The function value is the remainder of argument-1 divided by argument-2. It is determined using the following arithmetic expression:

argument-1 - (argument-2 * FUNCTION INTEGER-PART (argument-1 / argument-2))

Function Type

The function type is numeric.

REVERSE Function

This function returns the reverse of the character string in argument-1.

[Format]

FUNCTION REVERSE (argument-1)

Argument

1. The type of argument-1 must be numeric or alphanumeric.
2. The length of argument-1 must be one character or more.

Function Value

1. The function value is the reverse of the character string in argument-1. Its length is the same as of argument-1.
2. Assume the length of the character string of argument-1 to be n. The character in the jth ($1 \leq j \leq n$) in argument-1 is returned as the $(n - j + 1)$th character in the function value.

Function Type

The function type is alphanumeric.

SIN Function

This function returns the sine of argument-1.

[Format]

FUNCTION SIN (argument-1)

Arguments

1. The type of argument-1 must be numeric.
2. The unit of argument-1 must be radian.

Function Value

1. The function value is an approximate value of the sine of argument-1.
2. The range of the function value is:

   $-1 \leq$ function value $\leq +1$

Function Type

The function type is numeric.

SQRT Function

This function returns the square root of argument-1.

[Format]

<u>FUNCTION</u> <u>SQRT</u> (argument-1)

Arguments

1. The type of argument-1 must be numeric.
2. The value of argument-1 must be zero or greater.

Function Value

The function value is the absolute value of an approximate value of the square root of argument-1.

Function Type

The function type is numeric.

STANDARD-DEVIATION Function

This function returns the approximate value of the standard deviation of an argument.

[Format]

<u>FUNCTION</u> <u>STANDARD-DEVIATION</u> ({argument-1} ...)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is the approximate value of standard deviation of the list of argument-1 elements.
2. It is determined using the procedures below.
   a. The difference between the mean of argument-1 elements and each argument-1 value is determined and each difference is squared.
   b. All values determined in (a) are added, then divided by the number of argument-1 elements.
   c. The square root of the quotient found in (b) is determined. The function value will be the absolute value of the quotient.
3. If all argument-1 elements have the same value, the function value is zero.

Function Type

The function type is numeric.

SUM Function

This function returns the sum of an argument.

[Format]

<u>FUNCTION</u> <u>SUM</u> ({argument-1}...)

Argument

The type of argument-1 must be numeric or integer.

Function Value

The function value is the sum of the list of argument-1 elements.

Function Type

The table below lists the function types.

Type of argument-1	**Function Type**
All integers	Integer
All numeric or mix of numeric characters and integers	Numeric character

TAN Function

This function returns the tangent of argument-1.

[Format]

FUNCTION TAN (argument-1)

Argument

1. The type of argument-1 must be numeric.
2. The unit of argument-1 must be radian.

Function Value

The function value is the approximate value of the tangent of argument-1.

Function Type

The function type is numeric.

UPPER-CASE Function

This function returns uppercase alphabetic characters equivalent to lowercase alphabetic characters in argument-1.

[Format]

<u>FUNCTION</u> <u>UPPER-CASE</u> (argument-1)

Argument

1. The type of argument-1 must be numeric or alphanumeric.
2. The length of argument-1 must be one character or more.

Function Value

1. The function value is the character string of argument-1 with uppercase alphabetic characters converted to lowercase alphabetic characters. In other words, it is the same as argument-1, except that lowercase alphabetic characters in argument-1 have been converted to uppercase alphabetic characters.
2. The length of the function value is the same as of argument-1.

Function Type

The function type is alphanumeric.

VARIANCE Function

This function returns the approximate value of the variance of an argument.

[Format]

FUNCTION VARIANCE ({argument-1}...)

Argument

The type of argument-1 must be numeric.

Function Value

1. The function value is the approximate value of the variance of argument-1. In other words, it is the square of the standard deviation of the list of argument-1 elements. For standard deviation, see the section titled “STANDARD-DEVIATION function.”
2. When all argument-1 elements have all the same value, the function value is zero.

Function Type

The function type is numeric.

WHEN-COMPLIED Function

This function returns the date and time at which the program has been compiled.

[Format]

FUNCTION WHEN-COMPILED

Function Value

1. The function value is 21 alphanumeric characters indicating the date and time when the program was compiled as well as the time difference from Greenwich mean time. The meaning of each of character in the function value is listed below.

Character Position	Value
1 to 4	4-digit number indicating the year
5 to 6	2-digit number indicating the month. The value ranges from 01 to 12.
7 to 8	2-digit number indicating the day. The value ranges from 01 to 31.
9 to 10	2-digit number indicating hours. The value ranges from 00 to 23.
11 to 12	2-digit number indicating minutes. The value ranges from 00 to 59.
13 to 14	2-digit number indicating seconds. The value ranges from 00 to 59.
15 to 16	2-digit number indicating 1/100 seconds. The value ranges from 00 to 99. (*1)
17	Sign indicating whether the local time (time indicated in character positions 1 to 16) is ahead or behind the Greenwich mean time. A minus (-) sign indicates that the local time is behind Greenwich mean time, and a plus (+) sign indicates that it is ahead. (*2)
18 to 19	2-digit number indicating the number of hours ahead or behind Greenwich mean time. The value ranges from 00 to 12 when character position 17 is "-"; and ranges from 00 to 13 when it is "+." (*2)
20 to 21	2-digit number indicating the number of minutes ahead or behind Greenwich mean time. The value ranges from 00 to 59. (*2)

*1 Character positions 15 and 16 are set to 00 if the system cannot read 1/100 seconds.

*2 Character positions 17 to 21 are set to 00000 if the system cannot read the time difference between the local time and Greenwich mean time.

2. The function value is the date and time at which the source program containing this function has been compiled. The WHEN-COMPILED function in an internal program will return the date and time at which the highest nesting level program has been compiled.

Function Type

The function type is alphanumeric.

Chapter 7. Source Text Manipulation

The source statement manipulation function copies or replaces part of a source program from the COBOL library. It provides two statements: COPY and REPLACE.

The COPY statement copies library text to the program containing the COPY statement. When copying, it can also replace library text parts. The REPLACE statement replaces source program text parts.

The COPY and REPLACE statements can be written anywhere in the source program. A source program can consist of only COPY statements. At compilation time, COPY and REPLACE are processed before other statements are compiled. They have no function at execution time.

Text

A source program line or a set of lines and the COBOL library are called “text.” Text in the source program is called “source program text,” and text in the COBOL library is called “library text.”

Text Word

Character strings in the COBOL library, source programs, and areas A and B in pseudo-text. The following character strings are called “text words”:

- A separator except for space, pseudo text delimiter, and separator quotation mark. The right and left parentheses of the separator are always handled as text words, regardless of their location.
- A literal is a text word. For a nonnumeric literal, national literal, Boolean literal, and hexadecimal nonnumeric literal, the character strings series between the start and end separators of a literal value is one text word.
- Any character string between separators, which are neither separators nor literal, is a text word. However, the comment line character string and the word COPY are not text words.

Pseudo-text

Any text word, comment line, and separator space between two pseudo-text delimiters in the source program and COBOL library are called “pseudo-text.” A “pseudo-text delimiter” is two consecutive equal signs (==). Pseudo-text contains no pseudo-text delimiter.

COPY Statement

This statement copies library text to a source program.

[Format 1] Copies library text as it is or partially modify it to copy.

```
          ⎧ text-name-1  ⎡ ⎧ OF ⎫ library-name-1 ⎤ ⎫
  COPY   ⎨               ⎣ ⎩ IN ⎭                ⎦ ⎬
          ⎩ text-name-literal-1                    ⎭
      [ REPLACING
        ⎧ ==pseudo-text-1== ⎫      ⎧ ==pseudo-text-2== ⎫
      { ⎨ identifier-1      ⎬  BY  ⎨ identifier-2      ⎬ } ... ]
        ⎪ literal-1         ⎪      ⎪ literal-2         ⎪
        ⎩ word-1            ⎭      ⎩ word-2            ⎭
```

[Format 2] Replaces the prefix or suffix in library text to copy.

```
          ⎧ text-name-1  ⎡ ⎧ OF ⎫ library-name-1 ⎤ ⎫
  COPY   ⎨               ⎣ ⎩ IN ⎭                ⎦ ⎬
          ⎩ text-name-literal-1                    ⎭
      { DISJOINING word-3
       JOINING word-4

        AS ⎧ PREFIX ⎫ } ...
           ⎩ SUFFIX ⎭
```

[Format 3] Adds the prefix or suffix to library text to copy.

$$
\begin{array}{l}
\text{COPY text-name-1} \left\{ \begin{matrix} \underline{\text{OF}} \\ \underline{\text{IN}} \end{matrix} \right\} \left\{ \begin{matrix} \underline{\text{XMDLIB}} \\ \underline{\text{XFDLIB}} \end{matrix} \right\} \\
\quad [\ \underline{\text{JOINING}} \text{ word-4 AS} \left\{ \begin{matrix} \underline{\text{PREFIX}} \\ \underline{\text{SUFFIX}} \end{matrix} \right\}] \\
\quad [\ \underline{\text{REPLACING}} \\
\qquad \{ \left\{ \begin{matrix} \text{literal-1} \\ \text{word-5} \end{matrix} \right\} \ \underline{\text{BY}} \ \left\{ \begin{matrix} \text{literal-2} \\ \text{word-2} \end{matrix} \right\} \} \ldots]
\end{array}
$$

Syntax Rules

1. When two or more COBOL libraries are used in a compilation unit, text-name-1 must be qualified by the library name of the COBOL library containing the text. For specifying COBOL editing, refer to “COBOL85 User’s Guide.”

2. The text-name-1 must be unique in a COBOL library.

3. For text names and text name literal description rules, see the section titled “Basic Overview of Language”.

4. The COPY statement must begin after a space and end with a separator period.

5. The pseudo-text-1 and pseudo-text-2 must follow the reference format.

6. The pseudo-text-1 must contain more than one text word.

7. The pseudo-text-2 must contain more than one text word. Text words may be omitted.

8. The pseudo-text-1 and pseudo-text-2 can be written on more than one line according to the reference format. However, identifier-1 and identifier-2 cannot be written on more than one line.

9. The word-1, word-2, word-3 and word-4 must be character string following the COBOL word rules. However, the word COPY must not be specified. If national characters are used to specify word-3 and word-4, character-strings consisting of alphabetic-upper, alphabetic-lower, and numeric characters, JIS non-kanji characters can be written.

10. The COPY statement can be written at any position where a separator other than the right quotation mark can be written. However, a COPY statement must not be written in a COPY statement.

11. The pseudo-text-1, pseudo-text-2, and a text word in library text must range from 1 to 324 characters.

12. The pseudo-text-1 must not consist of only a separator comma or semicolon.

13. The word COPY written in a comment item, comment line, or nonnumeric literal is handled as part of a character string configuring the comment item, the comment line, or the nonnumeric literal.

14. When the user-defined words in library text use alphanumeric characters, word-4 must also be an alphanumeric character. When the user-defined word in library text is national language, word-4 must be national language.

15. For details on the maximum length of word-4, see Appendix B.

16. In Format 3, the JOINING phrase and the REPLACING phrase must not be specified concurrently.

General Rules

Rules Common to Formats 1 to 3

1. The COPY statement copies library text to a program at compilation time. Library text is specified in text-name-1 or text-name-literal-1.

   Library text is copied according to the rules listed below:

   When the REPLACE phrase is omitted in Format 1

   The COPY statement copies existing library text.

   When the REPLACE phrase is written in Format 1

   The COPY statement searches text words in library text for a character string matching the BY subject then replaces the matching string with the BY object before copying. The copy statement copies unmatching strings .

   For Format 2

   When writing the PREFIX phrase, the COPY statement searches library text for strings having the prefix in text words (strings from the left end of a text word to a hyphen) matching word-3 then replaces those matching strings with word-4 before copying.

   When writing the SUFFIX phrase, the statement searches library text for strings having the suffix in text words (strings from the left end of text word to a hyphen) matching word-3 then replaces matching strings with word-4 to before copying.

For Format 3

The COPY statement replaces all data-names written immediately after the level-number or the REDEFINES clause in library text. The PREFIX phrase then inserts and copies the prefix (word-4 plus hyphen) before replacing the leftmost data-name character . The SUFFIX phrase inserts and copies the prefix (hyphen plus word-4) after replacing the rightmost data-name character. The COPY statement copies non-replaced strings unchanged.

2. Source program compilation containing COPY statements is the same as source program compilation after processing all COPY statements. At compilation time, the entire COPY statement starting with COPY and ending with a period is replaced with library text or partially edited library text. The COPY statement has no function at execution time.

3. Literary text must follow the COBOL reference format.

4. COPY statements can be written in library text. However, no new COPY statement must be executed after the REPLACING or JOINING phrase processing.

Rules for Format 1

1. Text is replaced when the REPLACING phrase is written. See the procedures below.

   Replacement comparison starts from the first text word other than separator commas or semicolons in library text. This text word is called “present text word.” Spaces, separator commas and semicolons placed before the text word are copied unchanged.

The BY object is compared with library text beginning with the present text word, character by character. If two or more BY phrases are written, all operands having the BY phrase are compared in the BY phrases specified order. Phrases are compared as follows:

- When identifier-1, word-1, or literal-1 is written, it is compared with the present text word.
- When pseudo-text-1 is written, the entire string in pseudo-text-1 is compared with library text beginning from the present text word. Each string of separator commas, semicolons, or spaces in pseudo-text-1 and library text is assumed to be a single blank.

Step (b) is repeated until the comparison result matches or all operands on the left of BY are processed.

If no matching string is found in Step (b), the COPY statement copies the present text word unchanged. It then uses the next text word as “present text word” and returns to Step (b).

When any matching string is found in Step (b), either of the following occurs:

- When identifier-1, word-1, or literal-1 is written, the COPY statement replaces the present text word with the BY object to copy. Then, the next library text word is set as the present text word and returns to Step (b).
- When pseudo-text-1 is written, the COPY statement compares the strings matching pseudo-text-1 with the BY object to copy. Then the last text word matching pseudo-text-1 is set as the present text word and returns to Step (b).

Steps (b) to (e) are repeated until the final text word in library text is compared.

2. Comment line and blank lines in library text and pseudo-text-1 are ignored during replacement comparison.
3. Library text, pseudo-text-1, and pseudo-text-2 can contain debugging lines. During comparison, “D” in the indicator area in library text and pseudo-text-1 is ignored.
4. Text words not replaced in library text are copied and positioned in the same area as in library text. In other words, a text word starting from area A in library text is copied to also start from area A; and a text word starting from area B is copied to also start from area B. However, if two or more text words exist in area A and one of them is replaced to a longer text word, it is copied to start from area B if it cannot start from area A.
5. Matching words in library text are replaced with a new text word (identifier-2, literal-2, or word-2, or each text word in pseudo-text-2) and copied according to the reference format.
6. When text words in library text are replaced with identifier-2, literal-2, or word-2, the new text word (identifier-2, literal-2, or word-2) is copied to start from the same area in library text as the original text word.
7. When each text word in library text isi replaced with text word in pseudo-text-2, each pseudo-text-2 text word is copied to starts from the same area as pseudo-text-2. Spaces between text words in pseudo-text-2 are copied unchanged.

8. The following text words are copied to the debug line:

- When writing a COPY statement in a debug line, the text word in library text copied by the COPY statement
- The text word in the library text debug line
- The text word first replaced in debug line library text on the BY object
- The text word in the pseudo-text-2 debug line
- The above text word in the line containing the character “D” in the indicator area designating the debug line. It is the same even if a library text line crosses more than one line after text word replacement.

9. If literal-2, pseudo-text-2, or literal written in library text is copied in more than one line following text word replacement by the COPY statement, the indicator area on the next line shows “-” indicating line continuation. However, a literal must not be copied to continue on the debug line.

10. If a new line is added after text replacement by the COPY statement, the the new line indicator area reflects the same character as the original text word, except for cases (9) and (10). However, if the the line indicator area is a hyphen, a blank is set in the indicator area on the new line.

11. Comment lines and blank lines in library text are copied unchanged. However, no copies are processed if both exist between a series of COPY statements matching pseudo-text-1.

12. Comment lines and blank lines in pseudo-text-2 are always copied when pseudo-text-2 is copied.

Rules for Format 2

1. Text is replaced by the COPY statement in Format 2 as shown in the procedures below.

   Replacement comparison starts from the first text word other than any separator comma or semicolon in library text. This text word is called “present text word.” Spaces, separator commas and separator semicolons before text words are copied unchanged.

   word-3 is compared with the present text word. If word-3 is written more than once, all occurrences of word-3 are compared in the order written. Comparison is performed as shown below:

   - When word-3 is an alphanumeric string, the present text word is inspected to find whether it consists of only alphanumeric characters contianing the prefix or the suffix. The PREFIX phrase compares the present text word with word-3 of a hyphen plus one or more characters. The SUFFIX phrase compares the present text word with word-3 of one or more characters including a hyphen.

   - When word-3 is a National string, the present text word is inspected to find whether it is a character string consisting of only National character containing the prefix or suffix. The PREFIX phrase compares the present text word with word-3 of a JIS non-Kanji minus sign plus one or more national characters. The SUFFIX phrase compares the present text word with word-3 of one or more national characters plus a JIS non-Kanji minus sign.

Step (b) is repeated until the comparison result match or all occurrences of word-3 are processed.

If no matching string is found in Step (b), the present text word is copied unchanged. The next text word is used as the“present text word” and returns to Step (b).

When any matching string is found in Step (b) and copied, the present text word matching word-3 is replaced with word-4. Then, the next library text word is set as the present text word and processing returns to Step (b).

Steps (b) to (e) are repeated until the last text word in the library set is compared.

2. Matching text words in library text are replaced with a new text words (prefix or suffix replaced with word-4) and copied according to the reference format.
3. When text words in library text are replaced, new text words (prefix or suffix replaced with word-4) are copied to start from the same area started in the library text.

Rules for Format 3

1. All data-names immediately after level-number or in the library text REDEFINES clause are replaced according to the rules below. Other text words are copied unchanged. The “old data-name” is all data-name written immediately after level-number or in the REDEFINES clause.

   When word-4 is an alphanumeric string, the old data-name must also be an alphanumeric string. The PREFIX phrase replaces and copies the old data-name with word-4 of a hyphen plus the old data-name. The SUFFIX phrase replaces and copies the old data-name with word-4 of the old data-name plus a hyphen.

   When word-4 is a National string, the old data-name must be a National string. The PREFIX phrase replaces and copies the old data-name with word-4 of a JIS non-Kanji minus sign plus the old data-name. The SUFFIX phrase replaces and copies the old data-name with word-4 of the old data-name adding a JIS non-Kanji minus sign.

2. Matching text words in library text are replaced with a new text word (prefix or suffix replaced with word-4) and copied according to the reference format.

3. When text words in library text are replaced, new text words (prefix or suffix replaced with word-4) are copied and start from the same area as started in library text.

4. text-name-1 is treated as a screen form descriptor name when XMDLIB is specified. It is treated as a file descriptor name when XFDLIB is specified.

DS Win XFDLIB can be specified.

REPLACE Statement

This statement replaces the source program text.

[Format 1] Specifies the character string to be replaced and declares start of replacement

REPLACE {== pseudo-text-1 == BY == pseudo-text-2 ==} ...

[Format 2] Declares end of text replacement

REPLACE OFF

Syntax Rules

1. The REPLACE statement can be written in any position where character strings can be written in the source program. It must follow a separator period, except when it is the first statement in a separately compiled program.
2. The REPLACE statement must end with a separator period.
3. The pseudo-text-1 and pseudo-text-2 must follow the reference format.
4. The pseudo-text-1 must contain one or more text words.
5. The pseudo-text-2 can contain one or more text words. Text words may be omitted.
6. Character strings in pseudo-text-1 and pseudo-text-2 can cross more than one line according to the reference format.
7. Each text word in pseudo-text-1 and pseudo-text-2 must range from 1 to 324 characters.

8. The pseudo-text-1 must not consist of only separator commas or semicolons.

9. If the word REPLACE is written in a comment item, comment line, or nonnumeric literal, the word REPLACE is handled as part of the character string of the comment item, comment line, or nonnumeric literal.

General Rules

1. The REPLACE statement replaces the source program text at compilation time. The REPLACE statement in Format 1 specifies replacing any character string matching pseudo-text-1 in the source program with pseudo-text-2 and declares the start of text replacement. The REPLACE statement in Format 2 declares end of text replacement. The replacement rules for the REPLACE statement in Format 1 are valid until the next REPLACE statement appears or up to the end of the separately compiled program.

2. The REPLACE statement is processed after the COPY statement at compilation time. It has no function at run time.

3. No new REPLACE statements can be written to generate new REPLACE statements following execution of the REPLACE statement.

4. Text is replaced using the procedures below.

   The REPLACE statement starts replacement comparison from the first text word in the source program. This text word is called “present text word.”

The entire string in pseudo-text-1 is compared with the source program text starting from the present text word comparing character by character. If pseudo-text-1 is written more than once, all pseudo-text-1 elements are compared in the order they appear. At this time, each string of separator commas, semicolons, and separator spaces in pseudo-text-1 and the source program text is assumed to be one space.

Step (b) is repeated until a matching string is found or all the pseudo-text-1 elements are processed.

If no matching string is found in Step (b), REPLACE sets the next text word as the present text word then returns to Step (b).

If any matching string is found in Step (b), REPLACE replaces strings matching pseudo-text-1 with pseudo-text-2 according to the reference format. Then, it sets the text word following the last string matching pseudo-text-1 as present text word and returns to Step (b).

The REPLACE statement repeats Steps (b) to (e) until the last text word in library text.

5. The REPLACE statement ignores any comment or blank line in the source program and pseudo-text-1 during replacement comparison.

6. A debug line can be written in pseudo-text-1 and pseudo-text-2. The statement ignores “D” in the indicator area in pseudo-text-1 and pseudo-text-2 during comparison.

7. REPLACE replaces strings matching pseudo-text-1 in the source program text with pseudo-text-2 and inserts each pseudo-text-2 text word according to the reference format. It inserts spaces between pseudo-text-1 text words.

8. If literal in pseudo-text-2 are inserted across more than one line during text replacement by the REPLACE statement, “-” is set in the indicator area on the next line to indicate line continuation. However, a literal inserted as the debug line must not continue on the next line.

9. If new lines are added by the REPLACE statement text replacement , the same character as the line indicator area containing the old text is set in the new line indicator area. However, if the line indicator area shows a hyphen, a blank is set in the new line indicator area .

10. Any comment or blank line in pseudo-text-2 is inserted unchanged whenever pseudo-text-2 is inserted.

11. The REPLACE statement must not change the program name paragraph and end program header.

Win

Chapter 8. Database (SQL)

The database function (SQL) can access various types of databases and manipulate data by using embedded SQL statements as defined in COBOL85.

Database

A relational database is a set of two-dimensional tables. A table contains a set of rows and columns.

Data in the database can be sorted dynamically by values indicated by the user. The user can access data in a table, regardless of the physical location and order of the data. Data is processed by using a table as a logical user interface.

Table

Actual tables and view tables are generically referred to as tables. Data is stored in actual tables. View tables are virtual tables used for data manipulation. Virtual tables do not exist as physical data.

Embedded SQL

Embedded SQL is designed to manipulate databases within application programs. Embedded SQL contains the following:

- Embedded SQL declarative section

  This section starts with an embedded SQL begin declare and ends with embedded SQL end declare. Host variables are defined in the embedded SQL declarative section.

- Embedded SQL statement

  This statement manipulates databases and describes exception handling.

The embedded SQL statements defined in this chapter are COBOL85-defined embedded SQL statements. Embedded SQL statement operation details and its description depends on the particular database specifications.

Host Variable

Host variables are data items used for sending and receiving data between databases and application programs.

Host variables are used to store data in databases within application programs and to read data from databases.

Reference Format of Embedded SQL

Overall Rules

1. The embedded SQL begin declare, embedded SQL end declare, and embedded SQL statement must be enclosed by the SQL prefix (“EXEC SQL”) and the SQL terminator (“END-EXEC”).
2. The embedded SQL begin declare, embedded SQL end declare, and embedded SQL statement must all be described in Area B.

Continuation of Line

The continuation rule (continuation of line) of an embedded SQL begin declare, embedded SQL end declare, and embedded SQL statement conforms to the COBOL reference format rule .

COBOL Comment Line and In-line Comment

COBOL comment lines, debugging lines, and blank lines can be included in embedded SQL; however, in-line comments cannot.

Data Division

In the Data Division, the embedded SQL declare section describes where host variables are defined.

Embedded SQL Declare Section

Host variables are declared in the embedded SQL declare section.

[Format]

```
EXEC SQL
   BEGIN DECLARE SECTION END-EXEC.
[ { host variable definition } ... ]
EXEC SQL
   END DECLARE SECTION END-EXEC.
```

General Rules

1. An embedded SQL declare section can be included in the Working-Storage Section or the Linkage Section of the Data Division.
2. An embedded SQL begin declare (EXEC SQL BEGIN DECLARE SECTION END-EXEC) and an embedded SQL end declare (EXEC SQL END DECLARE SECTION END-EXEC) cannot be omitted.

3. An embedded SQL begin declare must be paired with an embedded SQL end declare. They may not be nested.
4. An embedded SQL declare section may not be included in another embedded DCSQL declare section. An embedded DCSQL declare section may not be included in an embedded SQL declarative section. This rule is effective in [Win16].

Host Variable Definitions

Host variable definitions specify the host variable characteristics.

[Format]

```
level-number      variable-name
    [ IS   EXTERNAL]
    [ IS   GLOBAL]
    [ { PICTURE }  IS  character-string ]
    [ { PIC     }                       ]
    [                  { BINARY               } ]
    [                  { COMPUTATIONAL        } ]
    [                  { COMP                 } ]
    [                  { COMPUTATIONAL-5      } ]
    [ [USAGE  IS]      { COMP-5               } ]
    [                  { COMP-1               } ]
    [                  { COMP-2               } ]
    [                  { DISPLAY              } ]
    [                  { PACKED-DECIMAL       } ]
    [ [ SIGN  IS]
     { LEADING SEPARATE  CHARACTER  }
     { TRAILING                     } ].
```

Syntax Rules

1. A host variable must be declared as an elementary item with level number 01 or 77.
2. A host variable corresponding to a variable-length character string data in a database should be declared as a level number 01 group item. In this case, items directly belonging to level number 01 must be level number 49 elementary items.
3. The Syntax rules of each clause is the same as each clause in the COBOL data description entry.

General Rules

1. The General rules of each clause is the same as each clause of the COBOL data description entry.
2. Host variables must be one of the following data types:
   a. Signed binary integer item of 4 or 9 digits
   b. Signed packed decimal item of 18 or less digits
   c. Signed zoned decimal item of 18 or less digits
   d. Alphanumeric data item
   e. National data item
   f. Single-precision internal floating-point item

g. Double-precision internal floating-point item

h. Group item with the following format (refer to the "COBOL85 User's Guide" for the m or n value)

```
01  data-name-1.
    49  data-name-2 PIC S9(m) BINARY.
    49  data-name-3 PIC X(n).
```

or

```
01  data-name-1.
    49  data-name-2 PIC S9(m) COMP-5.
    49  data-name-3 PIC X(n).
```

or

```
01  data-name-1.
    49  data-name-2 PIC S9(m) BINARY.
    49  data-name-3 PIC N(n).
```

or

```
01  data-name-1.
    49  data-name-2 PIC S9(m) COMP-5.
    49  data-name-3 PIC N(n).
```

3. Whether national data items can be used depends on the database and its related products.

Referencing Host Variables

Host variables can be referenced either in embedded SQL statements or in general COBOL statements. If a host variable is referenced in an embedded SQL statement, a colon (:) must be added in front of the host variable name. For example, if a host variable A is referenced, it must be declared as :A in the embedded SQL statement.

If a host variable is referenced in a general COBOL statement, do not add the colon.

SQLSTATE/SQLCODE

If an exception event occurs during the execution of an SQL statement, a code indicating the nature of the event is posted to the application program. SQLSTATE/SQLCODE is the area where the code is stored. The user can take appropriate action for the exception event by referencing SQLSTATE/SQLCODE.

SQLSTATE

SQLSTATE must be defined in an embedded SQL declare section as a 5 character alphanumeric data item.

SQLCODE

When using SQLCODE, a 9 digit signed binary integer item must be defined in an embedded SQL declare section .

SQLMSG

If an exception event occurs during the execution of an SQL statement, a message indicating the nature of the event is posted to the application program. SQLMSG is the area where this message is stored. The user can print or display the SQLMSG contents by using an output statement.

When using SQLMSG, an alphanumeric data item must be defined in an embedded SQL declare section.

Procedure Division

In the Procedure Division, embedded SQL statements are described.

The following symbols are used in this section:

<> : Name of an element forming a statement

::= : Operator forming an element. The element to be defined appears to the left of the operator and the formula defining the element appears to the right of the operator.

[] : Optional elements

{} : The user selects one of the elements described in the {}. The selectable elements are either separated by a vertical bar (“ | ”) or written vertically.

| : The element written after the vertical bar can be used instead of the element before the vertical bar. The vertical bar is given in braces.

... : The preceding element is repeated.

Character

The following characters can be described in SQL statements:

- Alphabetic character
- Digit
- Special character
- National character

Alphabetic Character

See the section titled “Characters and Character Sets” for information on alphabetic characters.

Digit

See the section titled “Characters and Character Sets” for information on digits.

Special Character

The following 18 special characters are available:

., <, (, +, *,), ;, -, / , ,, %, _, >, ?, :, =, “, '

National Character

Whether national characters can be used depends on the database and related products.

Literals

Values, excluding null value, are specified as literals in the embedded SQL. The following literal types are available:

- Literal
  - Numeric literal
    - Exact numeric literal
    - Approximate literal
  - Character string literal
  - National character string literal

Numeric Literal

A numeric literal has a numeric value.

[Format]

```
<numeric literal> ::=
   <exact numeric literal> | <approximate literal>

<exact numeric literal> ::=
   [+|-]{ <unsigned integer>[.<unsigned integer>]
     |<unsigned integer>.|.<unsigned integer>}

<approximate literal> ::=
   <mantissa>E<exponent>

<mantissa> ::=
   <exact numeric literal>

<exponent> ::=
   [+|-]<unsigned integer>

<unsigned integer> ::=
   <digit> ...
```

[Referenced section]

Element	Section
<digit>	"Character"

Syntax Rules

1. The exact numeric literal is a character string containing the digit 0 to 9 and a decimal point.
2. A positive or negative sign can be specified for the exact numeric literal.
3. The exact numeric literal data type is exact numeric. The precision of exact numeric literal is the number of digits in the literal. The exact numeric literal scale is the number of digits at the right of the decimal point.
4. The precision and scale of exact numeric literals conform to the rules of the exact numeric host variable.
5. The approximate literal is a character string containing the digits 0 to 9, a decimal point, and the character E.
6. A positive or negative sign can be specified for approximate literal.
7. The approximate literal data type is approximate. The precision of approximate literal is of its mantissa. The mantissa is specified in the same format as exact numeric literal.
8. The precision and scale of approximate literal conform to the approximate host variable .

General Rules

1. An exact numeric literal represents fixed point numeric data.
2. The interpreted numeric value of an exact numeric literal is derived from the normal mathematical interpretation of signed positional decimal notation.
3. An approximate represents floating-point numeric data.
4. The interpreted numeric value of an approximate literal is approximately the product of the exact numeric value, represented by the mantissa, with the number obtained by raising the number 10 to the power represented by the exponent.

Character String Literal

A character string literal has characters as its value.

[Format]

'<character>...'

[Referenced section]

Element	Section
<character >	“Character”

Syntax Rules

1. A character string literal is a character string starting and ending with a quotation mark. This literal specifies character data.
2. The character string literal data type is character string.
3. The character string literal length conforms to the rule for the host variable length whose data type is character string.

General Rules

The character string literal value is the characters series the literal contains.

National Character String Literal

A national character string literal is a literal having each national characters as its value. Whether the national character string literal can be used depends on the database and their related products.

[Format]

N'<national character>...'

[Referenced section]

Element	**Section**
<national character>	“Character”

Syntax Rules

1. A national character string literal is a national character string that starts with a list of N and a quotation mark and ends with a quotation mark. This literal specifies each national character.
2. The national character string literal data type is national character string. The national character string literal length is the number of national characters included in the literal.
3. The national character string literal length conforms to the rule of the host variable length whose data type is a national character string type.

General Rules

The national character string literal value is the series of national characters the literal contains.

Token

The token specifies the minimum unit constructing an SQL statement.

[Format]

```
<token> ::=
    <non-delimiting token> | <delimiting token>
<non-delimiting token> ::=
    <identifier> | <key word> | <numeric literal>
<identifier> ::=
    <alphabetic character>[{[<underscore>]
    {<alphabetic character> | <digit>}}...] |
    <national character identifier>
```

```
<national character identifier> ::=
   <national character string literal>
<underscore> ::=
   _
<key word> ::=
   Refer to “COBOL85 User’s Guide.”
<delimiting token> ::=
   <character string literal> | <national character string literal>
   | , | ( | ) | < | > | . | : | = | * | + | - | / | <> | >= | <=
<separator> ::=
   {<comment> | <space> | <new-line>}...
<comment> ::=
   <comment introducer>
   [{<character> | <national character>}...]
<comment introducer> ::=
   --
```

[Referenced section]

Element	**Section**
<alphabetic character>	“Character”
<digit>	“Character”
<character>	“Character”
<national character>	“Character”
<numeric literal>	“Numeric literal”
<character string literal>	“Character string literal”
<national character string literal>	“National character string literal”

Syntax Rules

The comment introducer is a list of two or more consecutive hyphens not separated by any spaces or new lines and is not included as a literal.

Names

The following types are available as embedded SQL names:

- Table name
- Cursor name
- Column name
- Correlation name
- SQL statement identifier

Table Name

The table name is the name of an actual table. The table name is used to specify the table used for data manipulation. The table name can be qualified by separating the qualified table from the qualifying table by inserting a period (.).

Cursor Name

A cursor name is assigned to a cursor. The cursor is the row indicator specifying a row in a table. The name is defined in a cursor declarative.

A cursor name must be an alphanumeric or a national character string containing 18 characters or less.

Column Name

The column name is the name of a column in a table. The column name can be qualified by the table or correlation name. The column name is qualified by separating the qualified column from the qualifying name by inserting a period (.).

Correlation Name

A correlation name is an alias of a table.

SQL Statement Identifier

The SQL statement identifier is an statement identifier used in a dynamic SQL statement. The SQL statement identifier is an alphanumeric or a national character string containing 18 characters or less.

Value Specification and Target Specification

A value specification specifies a host variable, an indicator variable, or a literal.

A target specification specifies the host variable and the indicator variable assigned to the value.

[Format]

```
<value specification> ::=
   <variable specification> | <literal>

<variable specification> ::=
   <host variable name> [<indicator variable>]

<dynamic parameter specification> ::= ?

<indicator variable> ::=

   [INDICATOR] <host variable name>

<host variable name> ::= <host variable>

<host variable> ::=
   See each rule.

<target specification> ::= <variable specification>
```

[Referenced section]

Element	Section
<literal>	“Literals”

Syntax Rules

1. The host variable name is the name given to a host variable. The host variable can be declared where an identifier or a data-name can be declared in an embedded SQL statement and in COBOL ordinary format.
2. The host variable name must be an alphanumeric or a national character string containing 30 characters or less. Its structure conforms to the COBOL user-defined words rule.
3. The host variable must be defined in an embedded SQL declare section.
4. A host variable specified by an indicator variable becomes an indicator variable.
5. The indicator variable must be a 4 digit signed binary integer item.
6. The indicator variable can be specified by pairing it with a host variable to be set in a table column or with a host variable to reading a value from the column.
7. The dynamic parameter specification is used to specify a variable as a value specification in a statement being prepared. The dynamic parameter specification specifies a question mark with a specified value. A dynamic parameter specification in a prepared statement is equivalent to a variable specification in an embedded SQL statement.

General Rules

1. When the host variable value related by an indicator variable is set in a table column, whether null value is set can be controlled by setting the following values for the indicator variable:

   a. 0 or a positive value: The host variable value is set in the column.

   b. Negative value: A null value is set in the column. In this case, the host variable contents are ignored.

2. When a value is read from a column to a host variable related to an indicator variable, the following values are set in the indicator variable by the database system after the SQL statement is executed:

   a. 0 or a positive value: The input value is not a null value and is stored as the host variable.

   b. Negative value: The input value is a null value. In this case, the host variable contents have no meaning.

Column Specification

A column specification references the column specified by the column name. See the section titled “Column name” for information on column names.

Set Function Specification

A set function specification specifies a value determined by applying a function.

[Format 1] Returns the total number of rows in a table

COUNT(*)

[Format 2] DISTINCT set function

$$\left\{\begin{matrix}\text{AVG}\\ \text{MAX}\\ \text{MIN}\\ \text{SUN}\\ \text{COUNT}\end{matrix}\right\} \text{ (DISTINCT column specification)}$$

[Format 3] ALL set function

{AVG | MAX | MIN | SUM} ([ALL] value expression)

[Referenced section]

Element	**Section**
<column specification>	Column Specification
<value expression>	Value Expression

General Rules

1. The set function in format 1 returns the total number of rows, including columns having null value.
2. The DISTINCT set function returns the result obtained by excluding columns having null value and counting columns with a unique value.
3. The ALL set function returns the result obtained by excluding columns having null value.

4. The functions of set functions in format 2 and 3 are the following:
   a. AVG: Obtains the average value.
   b. MAX: Obtains the maximum value.
   c. MIN: Obtains the minimum value.
   d. SUM: Obtains the sum.
   e. COUNT: Obtains the cardinal number of a table.

Value Expression

A value expression specifies a value.

[Format]

```
<value expression> ::=
   <term> | <value expression>+<term> |
   <value expression>-<term>

<term> ::=
   <factor> | <term>*<factor> | <term> / <factor>

<factor> ::=
   [+ | -]<primary>

<primary> :: =
   <column specification> |
   <dynamic parameter specification> | <literal> |
   <set function specification> | (<value expression>)
```

[Referenced section]

Element	**Section**
<column specification>	“Column specification”
<dynamic parameter specification>	“Value specification and Target specification”
<literal>	“Literals”
<set function specification>	“Set function specification”

Syntax Rules

Do not use successive unary arithmetic operators.

General Rules

1. An operator can be specified for a literal of exact numeric or approximate.
2. The unary plus (+) does not affect the element immediately following. The unary minus (-) indicates the element immediately following is to be multiplied by -1.
3. Binary arithmetic operators +, -, *, and / indicate addition subtraction, multiplication, and division respectively.
4. Parenthesized expressions, unary arithmetic operators, multiplication and division, and addition and subtraction are evaluated in this order. Operators at the same level are evaluated from left to right.
5. A divisor may not be 0.

Predicate

A predicate specifies conditions for creating a truth table including true, false, and unknown. A predicate has the following types:

- Comparison predicate
- BETWEEN predicate
- IN predicate
- LIKE predicate
- NULL predicate

- Quantified predicate
- EXISTS predicate

Comparison Predicate

Specify comparison of two row values.

[Format]

```
<comparison predicate> ::=
    <value expression><comp op>{<value expression>|
    <subquery>}

<comp op> ::=
    {=|<|<=|>|>=|<>}
```

[Referenced section}

Element	Section
<value expression> <subquery>	“Value expression” “Subquery”

General Rules

The first value expression data type and the second value expression or subquery must be respectively comparable. The degree of the subquery result must be 1.

BETWEEN Predicate

Specify a range comparison.

[Format]

```
<BETWEEN predicate> ::=
    <value expression> [NOT] BETWEEN <value expression>
    AND <value expression>
```

[Referenced section]

Element	Section
<value expression>	“Value expression”

General Rules

1. The three value expression data types must be respectively comparable.
2. Variable specifications and dynamic parameter specifications cannot be included in the value expressions.

IN Predicate

Specify a quantified comparison.

[Format]

```
<IN predicate> ::=
   <value expression> [NOT] IN {(<subquery>) |
   (<in value list>)}

<in value list> ::=
   <in value>{,<in value>}...

<in value> ::=
   {<literal> | <variable specification> | USER |
   <dynamic parameter specification>}
```

[Referenced section]

Element	Section
<value expression>	“Value expression”
<subquery>	“Subquery”
<literal>	“Literals”
<variable specification>	“Value specification and Target specification”
<dynamic parameter specification>	“Value specification and Target specification”

General Rules

The first value expression data type and the value specification in the subquery or in the value list must be respectively comparable.

LIKE Predicate

Specify a pattern-match comparison in the character type data.

[Format]

```
<LIKE predicate> ::=
  <mach value> [NOT] LIKE <pattern>

<match value> ::=
  <column specification>

<pattern> ::=
  <character string literal> | <variable specification> | USER |
  <dynamic parameter specification>
```

[Referenced section]

Element	Section
<column specification>	“Column specification”
<character string literal>	“Literals”
<variable specification>	“Value specification and Target specification”
<dynamic parameter specification>	“Value specification and Target specification”

General Rules

1. The column data type specified by column specification must be character or national character string type.
2. If the column data type specified by the column specification is character string type, a character string must be specified as pattern. If the column data type

specified by the column specification is national character string type, a national character string must be specified as pattern.

3. The pattern can consist of a character string and characters % (percent) and _ (underscore). It also can consist of a national character string and the national characters “%” and “_”.
4. The percent character corresponds to 0 or more arbitrary characters.
5. The underscore character corresponds to one arbitrary characters.

NULL Predicate

Specify a test for a null value.

[Format]

```
<NULL predicate> ::=
    <column specification> IS [NOT] NULL
```

[Referenced section]

Element	Section
<column specification>	“Column specification”

Quantified Predicate

Specify a quantified comparison.

[Format]

```
<quantified predicate> ::=
    <value expression> <comp op> {ALL | ANY} <subquery>
```

[Referenced section]

Element	Section
<value expression> <comp op> <subquery>	“Value expression” “Comparison predicate” “Subquery”

General Rules

The value expression and the subquery data types must be respectively comparable.

EXISTS Predicate

Specify a test for a non-empty set.

[Format]

```
<EXISTS predicate> ::=
   EXISTS <subquery>
```

[Referenced section]

Element	Section
<Subquery>	“Subquery”

Search Condition

Specify a condition containing the truth value true, false, or unknown, depending on the result of applying Boolean operators to specified conditions.

[Format]

```
<search condition> ::=
   <Boolean term> [OR <search condition>]

<Boolean term> ::=
   <Boolean factor> [AND <Boolean term>]
```

```
<Boolean factor> ::=
   [NOT] <Boolean primary>

<Boolean primary> ::=
   <predicate> | (<search condition>)
```

[Referenced section]

Element	Section
<predicate>	"Predicate"

Table Expression

Specify a table or a grouped table.

[Format]

```
<table expression> ::=
   <FROM clause>
   [<WHERE clause>]
   [<GROUP BY clause>]
   [<HAVING clause>]
```

FROM Clause

Specify a table derived from one or more named tables.

[Format]

```
<FROM clause> ::=
   FROM <table reference> [{,<table reference>}... ]

<table reference> ::=
   <table name> [<correlation name>]
```

[Referenced table]

Element	Section
<table name>	“Table name”

WHERE Clause

Specify a table derived by applying the search condition to the result of the preceding FROM clause.

[Format]

```
<WHERE clause> ::=
   WHERE <search condition>
```

[Referenced section]

Element	Section
<search condition>	“Search condition”

GROUP BY Clause

Specify a grouped table derived by applying the GROUP BY clause to the result specified by the FROM and WHERE clauses.

[Format]

```
<GROUP BY clause> ::=
   GROUP BY <column specification>
   [{,<column specification>}... ]
```

[Referenced section]

Element	Section
<column specification>	“Column specification”

HAVING Clause

Specify the groups selection meeting the grouped table search condition derived from the preceding GROUP BY or FROM clauses result.

[Format]

```
<HAVING clause> ::=
    HAVING <search condition>
```

[Referenced section]

Element	Section
<search condition>	“Search condition”

Query Specification

Specify a table derived from the table expressions result.

[Format]

```
SELECT [ALL | DISTINCT] <select list> <table expression>

<select list> ::=
    * | <select sublist> [{,<select sublist>}...]

<select sublist> ::=
    <value expression> | <table name>.* | <correlation name>.*
```

[Referenced section]

Element	Section
<table expression>	“Table expression”
<FROM clause>	“FROM clause”
<WHERE clause>	“WHERE clause”
<value expression>	“Value expression”
<table name>	“Table name”
<correlation name>	“Correlation name”

Query Expression

Specify a table.

[Format]

```
<query expression> ::=
   {<query specification> | (<query expression>) }|
   <query expression> UNION [ALL]
   {<query specification> | (<query expression>) }
```

[Referenced section]

Element	Section
<query specification>	“Query specification”

Subquery

Specify a value, a row or a table derived from the table expression.

[Format]

```
<subquery> ::=
   (SELECT [ALL | DISTINCT] <select list>
   <table expression>)
```

[Referenced section]

Element	Section
<select list> <table expression>	“Query specification” “Table expression”

Embedded Exception Declaration

Specify the action taken when an SQL statement causes an exception event.

[Format]

$$\text{WHENEVER} \left\{ \begin{array}{l} \text{SQLERROR} \\ \text{NOT FOUND} \end{array} \right\} \left\{ \begin{array}{l} \text{GO TO : procedure-name} \\ \text{CONTINUE} \end{array} \right\}$$

General Rules

1. If SQLERROR is specified, execution occurs when SQLSTATE / SQLCODE indicates neither normal termination nor no data.
2. If NOT FOUND is specified, execution occurs when SQLSTATE / SQLCODE indicates no data.
3. The embedded exception declaration has an effect on all SQL statements after the declaration in the source text sequence. The effect is valid until the end of the program or until the same conditioned declaration appears.
4. The embedded exception declaration scope is the program unit.

Data Manipulation without Using Cursor

The following statements manipulate data without using the cursor:

- SELECT statement
- DELETE statement (searched)
- INSERT statement
- UPDATE statement (searched)

SELECT Statement

The SELECT statement fetches the value from a single row in a table.

[Format]

```
<SELECT statement> ::=
   SELECT [ALL | DISTINCT] <select list>
   INTO <select target list>
   <table expression> [UNION <SELECT statement>... ]

<select target list> ::=
   <target specification> [{,<target specification>}...]
```

[Referenced section]

Element	Section
<select list>	“Query specification”
<table expression>	“Table expression”
<target specification>	“Value specification and Target specification”

Syntax Rules

1. The select list degree and the target specifications number in the select target list must be equal.
2. The select list column and the target specification data types must be assignable.

General Rules

The SELECT statement searched result is assigned to the host variable specified in select target list. The assignment begins at the leftmost column of the select list and at the leftmost target specification of select target list, and continues in order.

DELETE Statement (Searched)

The DELETE statement (searched) deletes table rows meeting the search condition.

[Format]

```
DELETE FROM <table name> [WHERE <search condition>]
```

[Referenced section]

Element	Section
<table name>	"Table name"
<search condition>	"Search condition"

Syntax Rules

The table identified by the table name must be a read-write table.

General Rules

1. If a search condition is specified, rows meeting the search condition are deleted.
2. If a search condition is omitted, all rows in the table are deleted.
3. A search condition is evaluated before rows in the table are deleted.

INSERT Statement

The INSERT statement inserts new rows into an existing table.

[Format]

```
INSERT INTO <table name> [(<inserted column list>)]
    {VALUES (<insert value list>) | <query specification> }

<inserted column list> ::=
    <column name> [{,<column name>}... ]

<insert value list> ::=
    <insert value> [{,<insert value>}... ]

<insert value> ::=
    <value specification> | <dynamic parameter specification> | NULL
```

[Referenced section]

Element	**Section**
<query specification>	"Query specification"
<table name>	"Table name"
<column name>	"Column name"
<search condition>	"Search condition"
<value specification>	"Value specification and Target specification"
<dynamic parameter specification>	"Value specification and Target specification"

Syntax Rules

1. The column name in the insert value list must exist in the table identified by table name, and must be unique.
2. If insert column list is omitted, the value is set in all table columns .
3. If insert column list is specified, the column numbers in the insert column list must be equal to the query specification degree. If insert column list is omitted, the table degree must be equal to the query specification.
4. Every column in query specification must be respectively assignable to the object table.

General Rules

The query specification result must not be empty.

UPDATE Statement (Searched)

The UPDATE statement (searched) updates rows meeting the search condition.

[Format]

```
UPDATE <table name>
   SET <set clause:searched> [{,<set clause:searched>}... ]
   [WHERE <search condition>]

<set clause:searched> ::=
   <object column:searched> = {<value expression> | NULL}
```

<object column:searched> ::=
  <column name>

[Referenced section]

Element	Section
<table name>	“Table name”
<search condition>	“Search condition”
<value expression>	“Value expression”
<column name>	“Column name”

Syntax Rules

1. The table identified by table name must be a read-write table.
2. The value expression in set clause must not include a set function specification.
3. The column name in set clause must be a column name existing in the object table.
4. The column name specified as the object column must be a column name existing in the object table.
5. In set clause, the value expression on the right side of the equal sign must be assignable to the column identified by column name on the left side of the equal sign.

General Rules

1. If the WHERE clause is specified, rows meeting the search condition are updated.
2. If the WHERE clause is omitted, all rows in the table are updated.
3. The set clause updates the row value to the value specified on the right side of the equal sign.
4. The column specification value included in the set clause value expression indicates the value before the UPDATE statement (searched) update.

Data Manipulation Using the Cursor

The following statements manipulate data using the cursor:

- Declare cursor
- OPEN statement
- CLOSE statement
- FETCH statement
- DELETE statement (positioned)
- UPDATE statement (positioned)

Declare Cursor

Define a cursor.

[Format]

```
  DECLARE <cursor name> CURSOR FOR
     <cursor specification>

<cursor specification> ::=
     {<query expression> [<ORDER BY clause>]
      | SELECT [ALL | DISTINCT] <select list>
     <FROM clause> [<WHERE clause>]
     FOR UPDATE

 <ORDER BY clause> ::=
     ORDER BY <sort specification> [{,<sort specification>}....]
  <sort specification> ::=
     {<unsigned integer> | <column specification>} [ASC | DESC]
```

[Referenced section]

Element	Section
<cursor name>	“Cursor name”
<query expression>	“Query expression”
<select list>	“Query specification”
<FROM clause>	“FROM clause”
<WHERE clause>	“WHERE clause”
<column specification>	“Column specification”

Syntax Rules

1. The cursor name must be unique within the compilation unit.
2. If the ORDER BY clause is specified, rows in the derived table are ordered specified by the ORDER BY clause specification.
3. The sort specification in the ORDER BY clause must reference a column in the derived table. The effects of the sort specification are as follows:
    a. If the sort specification contains a column specification, the sort specification must specify the derived table column name.

    b. If the sort specification contains an unsigned integer, the sort specification must specify the derived table column with the ordinal position specified by the unsigned integer. The unsigned integer must be greater than 1 and not greater than the derived table degree.

    c. Column specification and an unsigned integer can be mixed in a sort specification.

General Rules

1. A cursor declaration must appear before an SQL statement using the cursor in source codes.
2. The derived table is a virtual table specified by a query expression. Once a derived table is defined by a cursor declaration, the table is always valid within the compilation unit.
3. The cursor is in the closed state following the cursor definition.
4. The column name, data type, precision, scale, and length of a derived table are equal to the query expression values specified in the cursor declaration.
5. The ORDER BY clause effects are the following:
   a. If ASC is specified, rows are given in ascending order.
   b. If DESC is specified, rows are given in descending order.
   c. The sort priority among sort specifications is determined by the order specified.
   d. If sorted columns have the same value, they are given in the order applied when no ORDER BY clause is specified.
6. The DELETE statement (positioned) and UPDATE statement (positioned) cannot affect a read-only derived table.

OPEN Statement

The OPEN statement opens and validates a cursor.

[Format]

OPEN <cursor name>

[Referenced section]

Element	Section
<cursor name>	“Cursor name”

General Rules

1. The cursor is opened by the OPEN statement.
2. Only the cursor in closed state can be opened.
3. Following open, the cursor is positioned before the first row.

CLOSE Statement

The CLOSE statement closes and invalidates the cursor.

[Format]

CLOSE <cursor name>

[Referenced section]

Element	Section
<cursor name>	“Cursor name”

General Rules

1. Only the cursor in an opened state can be closed.
2. A cursor in a closed state cannot be used until it is again opened by an OPEN statement.

FETCH Statement

The FETCH statement positions the cursor at the specified row and fetches the value from the row.

[Format]

```
FETCH <cursor name> INTO <fetch target list>

<fetch target list> ::=
   <target specification> [{,<target specification>}... ]
```

[Referenced section]

Element	Section
<cursor name>	"Cursor name"
<target specification>	"Value specification and Target specification"

Syntax Rules

1. The columns degree in the derived table must be equal to the host variables number specified in fetch target list.
2. Each column in the derived table must be assignable to the host variable specified in fetch target list.

General Rules

1. The cursor must be in the open state.
2. The row values are read from the leftmost column of the derived table, and are assigned to host variables in the order specified at the fetch target list.

DELETE Statement (Positioned)

The DELETE statement (positioned) deletes one row specified by the cursor.

[Format]

```
DELETE FROM <table name>
   WHERE CURRENT OF <cursor name>
```

[Referenced section]

Element	Section
<table name>	“Table name”
<cursor name>	“Cursor name”

Syntax Rules

The table name identified by table name must be the table deriving the cursor.

General Rules

The DELETE statement (positioned) deletes the row from the derived current row of the cursor .

UPDATE Statement (Positioned)

The UPDATE statement (positioned) updates the table row specified by the cursor.

[Format]

```
UPDATE <table name>
   SET <set clause:positioned> [{,<set clause:positioned>}... ]
   WHERE CURRENT OF <cursor name>

<set clause:positioned> ::=
   <object column:positioned> = {<value expression> | NULL}

<object column:positioned> ::=
   <column name>
```

[Referenced section]

Element	Section
<table name>	“Table name”
<cursor name>	“Cursor name”
<value expression>	“Value expression”
<column name>	“Column name”

Syntax Rules

1. The table name identified by table name must be the table deriving the cursor.
2. The following rules are applied to set clause:
   a. The column name in set clause must be the column name existing in the table deriving the cursor.
   b. The column name specified for set clause must be unique.
   c. The value expression specified in set clause must assignable to the associated column.

General Rules

1. The UPDATE statement (positioned) updates the value of the row from the derived current row of the cursor .
2. The row value is updated to the value specified on the right side of the equal sign.
3. The column specification value included in a value expression indicates the value before the UPDATE statement (positioned) update.

Dynamic SQL

INTO Clause / USING Clause

The INTO clause / USING clause describes the dynamic SQL statement output variable and input parameter .

[Format]

```
{INTO | USING} {<target specification>
     [{,<target specification>}...]
```

[Referenced section]

Element	**Section**
<target specification>	"Value specification and Target specification"

PREPARE Statement

The PREPARE statement prepares an SQL statement for execution.

[Format]

```
PREPARE <SQL statement identifier> FROM
   <SQL statement variable>

<SQL statement variable> ::=
   <host variable name>
```

[Referenced section]

Element	**Section**
<SQL statement identifier> <host variable name>	"SQL statement identifier" "Value specification and Target specification"

Syntax Rules

The SQL statement variable data type must be a fixed-length or variable-length character string.

General Rules

1. EXEC SQL, END-EXEC, and host variable specification cannot be included in the SQL statement variable contents.
2. Instead of variable specification, the dynamic parameter specification can be included in the SQL statement variable contents.
3. The preparation for executing the SQL statement is complete to execute the PREPARE statement after setting the SQL statement as the SQL statement variable contents. The statement prepared with the PREPARE statement is called a prepared statement.
4. If the contents of the SQL statement variable is a dynamic SELECT statement, the cursor related to the SQL statement identifier must be in the closed state.

EXECUTE Statement

The EXECUTE statement associates input parameters and output targets with a prepared statement and executes the statement.

[Format]

```
EXECUTE <SQL statement identifier> [<INTO clause>]
     [<USING clause>]
```

[Referenced section]

Element	Section
<SQL statement identifier>	“SQL statement identifier”
<INTO clause>	“INTO clause / USING clause”
<USING clause>	“INTO clause / USING clause”

General Rules

1. The prepared statement associated with an SQL statement identifier must be previously prepared within the same compilation unit by the PREPARE statement.
2. The following six prepared statements can be executed by the EXECUTE statement:
    a. SELECT statement
    b. DELETE statement (searched)
    c. DELETE statement (positioned)
    d. INSERT statement
    e. UPDATE statement (searched)
    f. UPDATE statement (positioned)
3. If a dynamic parameter specification is specified in a prepared statement, the USING clause must be specified.
4. Target specification in a USING clause must be set to a value in advance.
5. If a prepared statement is a dynamic single row SELECT statement, the INTO clause must be specified.
6. The EXECUTE statement associates the input parameter specified in the USING clause with the dynamic parameter specification value and executes the prepared statement. If the prepared statement is a dynamic single row SELECT statement, the execution result value is set in the INTO clause host variable.

EXECUTE IMMEDIATE Statement

The EXECUTE IMMEDIATE statement dynamically prepares and executes a prepared statement.

[Format]

EXECUTE IMMEDIATE <SQL statement variable>

[Referenced section]

Element	Section
<SQL statement variable>	“PREPARE statement”

Syntax Rules

The SQL statement variable data type must be fixed-length or variable-length character string.

General Rules

1. The following five SQL statements can be specified as the SQL statement variable contents:
    a. DELETE statement (searched)
    b. DELETE statement (positioned)
    c. INSERT statement
    d. UPDATE statement (searched)
    e. UPDATE statement (positioned)
2. EXEC SQL, END-EXEC, host variable specification, and dynamic parameter specification cannot be included in the contents of the SQL statement variable.

Dynamic SELECT Statement

The query expression or subquery formats can be applied to the dynamic SELECT statement. For information on query expression or subquery formats , see the section titled “Query expression” and the section titled “Subquery.”

The dynamic SELECT statement can include dynamic parameter specifications.

Dynamic Declare Cursor

Declare the cursor associated with the SQL statement identifier prepared as a dynamic SELECT statement.

[Format]

```
DECLARE <cursor name> CURSOR FOR
    <SQL statement   identifier>
```

[Referenced section]

Element	**Section**
<cursor name> <SQL statement identifier>	“Cursor name” “SQL statement identifier”

Syntax Rules

1. A cursor name must be unique within a compilation unit.
2. The SQL statement identifier specified in the dynamic cursor declaration must be prepared with the PREPARE statement within the same compilation unit.

General Rules

1. The prepared statement associated with the specified dynamic cursor declaration SQL statement identifier must be a cursor specification (dynamic SELECT statement). The cursor specification related to the SQL statement identifier can specify dynamic parameter specification.
2. The cursor defined by a dynamic cursor declaration can be used in the same way as a cursor defined by a cursor declaration.

Dynamic OPEN Statement

The dynamic OPEN statement associates input parameters with the cursor specification, and opens the cursor.

[Format]

```
OPEN <cursor name> [<USING clause>]
```

[Referenced section]

Element	**Section**
<cursor name>	“Cursor name”
<USING clause>	“INTO clause / USING clause”

Syntax Rules

The OPEN statement syntax rules can be applied to the dynamic OPEN statement by replacing a cursor declaration with a dynamic cursor declaration and replacing an OPEN statement with a dynamic OPEN statement.

General Rules

1. The prepared statement associated with a cursor name must be a cursor specification.
2. If the cursor specification associated with a cursor name includes a dynamic parameter specification, a USING clause must be specified in the dynamic OPEN statement.
3. The dynamic parameter specification value in the prepared statement is set in the USING clause target specification .
4. The OPEN statement general rules can be applied to the dynamic OPEN statement by replacing a cursor declaration with a dynamic cursor declaration.

Dynamic CLOSE Statement

The dynamic CLOSE statement closes the cursor.

[Format]

CLOSE <cursor name>

[Referenced section]

Element	Section
<cursor name>	“Cursor name”

Syntax Rules

The CLOSE statement syntax rules can be applied to the dynamic CLOSE statement by replacing a cursor declaration with a dynamic cursor declaration and replacing a CLOSE statement with a dynamic CLOSE statement.

General Rules

The CLOSE statement general rules can be applied to the dynamic CLOSE statement.

Dynamic FETCH Statement

The dynamic FETCH statement fetches a row for a cursor declared with a dynamic cursor declaration.

[Format]

```
FETCH <cursor name> <INTO clause>
```

[Referenced section]

Element	Section
<cursor name>	“Cursor name”
<INTO clause>	“INTO clause / USING clause”

Syntax Rules

The FETCH statement syntax rules can be applied to the dynamic FETCH statement by replacing a cursor declaration with a dynamic cursor declaration.

General Rules

The FETCH statement general rules can be applied to the dynamic FETCH statement by replacing a cursor declaration with a dynamic cursor declaration.

Dynamic DELETE Statement (Positioned)

The dynamic DELETE statement (positioned) deletes a row of a table.

[Format]

```
DELETE FROM <table name>
   WHERE CURRENT OF <cursor name>
```

[Referenced section]

Element	Section
<table name>	“Table name”
<cursor name>	“Cursor name”

Syntax Rules

The DELETE statement (positioned) syntax rules can be applied to the dynamic DELETE statement (positioned) by replacing a cursor declarative with a dynamic cursor declarative and replacing a DELETE statement (positioned) with a dynamic DELETE statement (positioned).

General Rules

The DELETE statement (positioned) general rules can be applied to the dynamic DELETE statement (positioned) by replacing a DELETE statement (positioned) with a dynamic DELETE statement (positioned).

Dynamic UPDATE Statement (Positioned)

The dynamic UPDATE statement (positioned) updates a row of a table.

[Format]

```
UPDATE <table name>
   SET <set clause:positioned> [{,<set clause:positioned>}...]
   WHERE CURRENT OF <cursor name>
```

[Referenced section]

Element	Section
<table name>	"Table name"
<set clause:positioned>	"UPDATE Statement (positioned)"
<cursor name>	"Cursor name"

Syntax Rules

The UPDATE statement (positioned) syntax rules can be applied to the dynamic UPDATE statement (positioned) by replacing a cursor declaration with a dynamic cursor declaration and replacing an UPDATE statement (positioned) with a dynamic UPDATE statement (positioned).

General Rules

The UPDATE statement (positioned) general rules can be applied to the dynamic UPDATE statement (positioned) by replacing an UPDATE statement (positioned) with a dynamic UPDATE statement (positioned).

Transaction Management

COMMIT Statement

Terminates the current transaction with commit.

[Format]

COMMIT [WORK]

General Rules

1. The current transaction is terminated.
2. Any changes made by the current transaction are reflected in the database.

ROLLBACK Statement

Terminates the current transaction with rollback.

[Format]

ROLLBACK [WORK]

General Rules

1. The current transaction is terminated.
2. Any changes made by the current transaction are canceled.

Connection Management

CONNECT Statement

Establish an SQL connection.

[Format]

```
CONNECT TO
    <connection target>

<connection target> ::=
   {<server name> [AS <connection name>]
   [USER <user name>]
    | DEFAULT}

<server name> ::= See the General rules.

<connection name> ::= See the General rules.

<user name> ::= See the General rules.
```

General Rules

1. The server name, connection name, and user name must be a character string literal or a host variable whose data type is fixed-length character string.
2. The server name is the same name specified as the server information in the operating environment file.
3. The maximum length of a server name is 32 bytes. However, this rule may be restricted by the user system .
4. The connection name identifies the connection.

5. The maximum length of the connection name is 18 bytes. However, this rule may be restricted by the system being used.

6. If the connection name is omitted, the server name is assumed to be the connection name.

7. A user name consists of a user-ID and a password, separated by a slash (/). A user-ID and a password must be catalogued in a server.

8. The maximum length of a user-ID and a password is 32 bytes each. However, this rule may be restricted by the system being used.

9. Spaces before and after a server name, a connection name, a user-ID, and a password are ignored.

10. If USER is omitted, the default connection information in the operating environment file is validated.

11. If DEFAULT is specified, the default connection information in the operating environment file is validated.

12. Execution of a CONNECT statement establishes a connection to a server.

13. If two or more connections are established, the last connection is the current connection.

14. Two or more connections cannot have the same connection name. Also, two or more connections can not specify DEFAULT.

SET CONNECTION Statement

Selects one connection from the available connections.

[Format]

```
SET CONNECTION
    <connection object>

<connection object> ::=
    <connection name> | DEFAULT

<connection name> ::= See the General rules.
```

General Rules

1. A connection name must be a character string literal or a host variable whose data type is a fixed-length character string.
2. The maximum length of a connection name is 18 bytes. However, this rule may be restricted by the system being used.
3. A connection name specifies the available connection.
4. If DEFAULT is specified, the default connection was established previously.

DISCONNECT Statement

Terminates a SQL connection.

[Format]

```
DISCONNECT
    <disconnection object>

<disconnection object> ::=
    <connection name> | DEFAULT | ALL | CURRENT

<connection name> ::= See the General rules.
```

General Rules

1. A connection name must be a character string literal or a host variable whose data type is fixed-length character string.
2. The maximum length of a connection name is 18 bytes. However, this rule may be restricted by the system being used.
3. If a connection name is specified, the connection identified by the name is disconnected.
4. If DEFAULT is specified, the default connection is disconnected.
5. If ALL is specified, all available connections are disconnected.

6. If CURRENT is specified, the current connection is disconnected.
7. Before the DISCONNECT statement is executed, the transaction must be terminated.
8. If the current connection is disconnected, there is no current connection.

Win16

Chapter 9. Communication Database

The communication database function transmits and receives data via the client server model communication database.

The communication database function is specific to [Win16].

General Overview

In a client server model, the client sends unprocessed data to the server application and requesting to process the data. The server application honors the request and returns the processing results to the client. In this communication, the server application and client do not directly communicate with each other but do send and receive data via the communication database.

When client data is sent , the client transmits data instruction directed to the communication database. The server application issues a data receive instruction directed to the communication database, and receives the data transmitted from the client.

When server data is sent from the application, the server application transmits data instruction directed to the communication database. The client issues a data receive instruction directed to the communication database, and receives data transmitted from the server application.

The communication database allows the user to communicate with a location destination, irrespective of the internal processing taking place, to the server application, or to where the actual client is.

The communication database function is specific to [Win16]. Both server and client must be running the PowerAIM products and its communication software to use the communication database.

Communication Database

The communication database is a virtual database. However, because the communication database is accessed using SQL, it functions as a real database. The user sends or receives data directed via the communication database.

The communication database provides the necessary services and tables for linkage between the server application and the client.

Services

Service information is held in tables available to the user.

When a service is specified by the client, the corresponding server application is activated so the client can communicate with it.

A service is associated one to one with a server application.

Tables

Tables define the communication data attributes . Tables are open to users sending or receiving data conforming to table definitions.

There are input tables and output tables. An input table defines the data attributes received from the client by the server application. An output table defines the data attributes sent to the client by the server application.

Embedded DCSQL

Embedded DCSQL is designed to communicate within application programs using the communication database. DCSQL is part of SQL used for accessing the communication database.

A description in embedded DCSQL is enclosed by “EXEC DCSQL” and “END-EXEC” in the COBOL program.

Host Variable

Host variables are data items used for sending and receiving data between server applications and clients.

For further details regarding host variables, see the section titled “Host Variable Names” and the section titled “Host Variable Definitions.”

Basic Elements of Embedded DCSQL

This section explains the basic elements of embedded DCSQL in a COBOL program.

Available Characters

The following six characters types can be used for embedded DCSQL:

- Digit
- Alphabetic character
- Alphanumeric character
- Special character
- Extended character
- National character

Digit

The characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9

Alphabetic Character

Uppercase and lowercase alphabetic characters are the following:

- Uppercase alphabetic characters (A, B, C, ...,Z)
- Lowercase alphabetic characters (a, b, c, ...,z)

Alphanumeric Character

An alphanumeric character is a set of alphabetic and numeric characters

Special Character

The following 16 special characters are available:

.	<	(	+	*	)	;	-
/	,	%	_	>	?	:	“

Extended Character

The following 3 extended characters are available:

@, \ , #

National Character

National character is available.

Quotation Marks, Key Words, and Separators

In embedded DCSQL, an apostrophe (') is used as the quotation mark. To write a quotation mark in a character string literal, specify two quotation marks in succession. Within a character string literal, the compiler uses a single apostrophe in terms of value and length.

The rules for key words and separators in embedded DCSQL statements follow the rules for embedded SQL.

Communication Database Names

The communication database name must conform to the following rules:

- 18 or less alphanumeric characters
- 8 or less national characters

Service Names

The service name must conform to the following rules:

- 18 or less alphanumeric characters
- 18 or less national characters

Table Names

Table names must conform to the following rules:

- 18 or less alphanumeric characters
- 18 or less national characters

Host Variable Names

Host variable names are names the user assigns to host variables. Host variable names can be specified for any sections permitting the specification of identifiers or data-names in embedded DCSQL statements of the Procedure Division and COBOL programs.

A host variable name must be a list of 18 or less alphanumeric characters or national characters. The configuration characters must follow the user rules defined words in COBOL.

Literals

The following literal can be specified in the embedded DCSQL:

- Character string literal
- National character string literal
- Exact numeric literal
- Approximate literal

Character String Literal

A character string literal is a character string starting and ending with a quotation mark. The character string literal specifies character data.

[Format]

```
'{character-1}...'
```

Syntax Rules

1. To include a quotation mark in a character string literal, specify two quotation marks in succession. Within a character string literal, the compiler uses a single apostrophe in terms of value and length.
2. The character string literal data type is character string.
3. The character string literal length conforms to the host variable length rules whose data type is character string.

General Rules

The character string literal value is the character series the literal contains.

National Character String Literal

A national character string literal is a national character string starting with a list of N and a quotation mark and ends with a quotation mark. This literal specifies each national character.

[Format]

```
N'{national-character-1}...'
```

Syntax Rules

1. The national character string literal data type is national character string. The national character string literal length is the number of national characters included in the literal.
2. The national character string literal length conforms to the host variable length rules whose data type is national character string type.

General Rules

The national character string literal value is the national characters series the literal contains.

Exact Numeric Literal

An exact numeric literal is the character strings consisting of digits 0 to 9 and a decimal point. A positive or negative sign can be specified for an exact numeric literal.

[Format]

$$\begin{bmatrix}+\\-\end{bmatrix}\underbrace{[\text{digit-string-1}]}_{\text{integer-part}}\ \underbrace{[.\text{digit-string-2}]}_{\text{decimal-part}}$$

Syntax Rules

1. The exact numeric literal data type is an exact numeric. The precision of an exact numeric literal is defined by the number of digits contained in it. The scale of an exact numeric literal is the number of digits to the right of the decimal point.
2. The precision and scale of an exact numeric literal conform to the host variables rules for precision and scale where the data type is an exact numeric.

General Rules

1. An exact numeric literal represents fixed-point numeric data.
2. The exact numeric literal numeric value is the value derived from normal numeric interpretation of signed positional decimal notation.

Approximate Literal

An approximate literal is a character string consisting of digits 0 to 9, a decimal point, and the character E. A positive or negative sign can be specified for an approximate literal.

[Format]

[+|-]mantissa E exponent

Syntax Rules

1. The approximate literal data type is the approximate. The approximate precision is the precision of its mantissa.
2. Specify the mantissa in exact numeric literal format.
3. Specify the exponent in the following format:

   [+|-]{numeric}...

4. The precision and scale of an approximate literal conform to the host variable rules for precision and scale where the data type is the approximate.

General Rules

1. An approximate represents floating-point numeric data.
2. The approximate literal value is the number product of the exact numeric expressed by the mantissa and the power of ten expressed by the exponent.

Reference Format of Embedded DCSQL

This section explains the reference format of the embedded DCSQL described in COBOL programs.

Overall Rules for Description

1. The embedded DCSQL begin declare, embedded DCSQL end declare, and embedded DCSQL statement must be enclosed by the DCSQL prefix (EXEC DCSQL) and the DCSQL terminator (END-EXEC).
2. The embedded DCSQL begin declare, embedded DCSQL end declare, and embedded DCSQL statement must all be described in Area B.

Continuation of Line

The rule of continuation (continuation of line) of an embedded DCSQL begin declare, embedded DCSQL end declare, and embedded DCSQL statement conform to the reference format rules of COBOL.

COBOL Comment Line and In-line Comment

COBOL comment lines, debugging lines, and blank lines can be included in embedded DCSQL; however, in-line comments cannot.

Comments in Embedded DCSQL

A comment in embedded DCSQL begins with two consecutive hyphens (--) and ends at the character position immediately to the left of Margin R. A comment in embedded DCSQL may begin at any position in Area A and Area B of embedded DCSQL statements, however a space or a comma must immediately precede the DCSQL comment. A comment in embedded DCSQL cannot be continued. To continue a word, literal, or PICTURE clause character string immediately preceding embedded DCSQL comments, place a hyphen in the indicator area.

Embedded DCSQL

Embedded DCSQL is designed to communicate within application programs using the communication database. This section describes the formats and rules for the embedded DCSQL declarative section and the embedded DCSQL statement.

Data Division

In the Data Division, variables used in the embedded DCSQL statement are defined.

Embedded DCSQL Declare Section

In the embedded DCSQL declare section, declare the following:

- Host variables used for communication between the server application and client.
- Variables used for receiving the processing result status reported by the system
- Variables used for receiving system messages

[Format]

```
EXEC DCSQL
   BEGIN DECLARE SECTION END-EXEC.
[{host variable definition}...]
EXEC DCSQL
   END DECLARE SECTION END-EXEC.
```

General Rules

1. An embedded DCSQL declare section can be included in the Working-Storage Section or the Linkage Section of the Data Division.
2. An embedded DCSQL begin declare (EXEC DCSQL BEGIN DECLARE SECTION END-EXEC) and an embedded DCSQL end declare (EXEC DCSQL END DECLARE SECTION END-EXEC) cannot be omitted.
3. An embedded DCSQL begin declare must be paired with an embedded DCSQL end declare. They may not be nested.
4. An embedded DCSQL declare section may not be included in another embedded SQL declare section. An embedded SQL declare section may not be included in an embedded DCSQL declare section.

Host Variable Definitions

Host variable definitions specify the host variable characteristics.

When a floating-point item is defined as the host variable, COMP-1 or COMP-2 is specified in the USAGE clause.

[Format]

```
level-number  host-variable-name
[IS EXTERNAL]
[IS GLOBAL]
[{ PICTURE } IS  character-string ]
[{ PIC     }                      ]
[                  { BINARY        } ]
[                  { COMPUTATIONAL } ]
[                  { COMP          } ]
[ [ USAGE   IS]    { COMP-1        } ]
[                  { COMP-2        } ]
[                  { DISPLAY       } ]
[                  { PAKED-DECIMAL } ]
[ [SIGN   IS]
    LEADING SEPARATE   CHARACTER]
[VALUE  IS    literal-1].
```

Syntax Rules

1. A host variable must be declared as an elementary item with level number 01 or 77.
2. A host variable corresponding to the item defined in the table as data type CHARACTER VARYING or NATIONAL CHARACTER VARYING must be declared as a level-number 01 group item. In this case, items directly belonging to level-number 01 must be level-number 49 elementary items.
3. The syntax rule of each clause must conform to the syntax rule of each clause in the COBOL data description entry.

General Rules

1. A PICTURE clause can only be specified at an elementary item level.
2. The general rule of each clause must conform to the general rule of each clause in the COBOL data description entry.
3. When a host variable is specified, it must be one of the following elementary items:
   a. Signed binary data item of 4 or 9 digits
   b. Signed packed decimal item of 18 or less digits
   c. Signed zoned decimal item of 18 or less digits (SIGN clause must be specified.)
   d. Alphanumeric data item
   e. National data item
   f. Single-precision internal floating-point item
   g. Double-precision internal floating-point item

4. Host variables corresponding to an item defined in a table as data type CHARACTER VARYING must the following:

   ```
   01   host-variable-name-1.
      49  host-variable-name-2 PIC S9(9) BINARY.
      49  host-variable-name-3 PIC X(n).
   ```

5. Host variables corresponding to an item defined in a table as data type NATIONAL CHARACTER VARYING must the following:

   ```
   01   host-variable-name-1.
      49  host-variable-name-2  PIC S9(9)  BINARY.
      49  host-variable-name-3  PIC N(n).
   ```

6. A host variable cannot be a filler.

Referencing Host Variables

Host variables can be referenced either in embedded DCSQL statements or in general COBOL statements. If a host variable is referenced in an embedded DCSQL statement, a colon (:) must be added in front of the host variable name. For example, if host variable A is referenced, it must be declared as :A in the embedded DCSQL statement. If a host variable is referenced in a general COBOL statement, do not add the colon.

DCSQLSTATE

When a DCSQL statement is executed, information on successful or failed execution is coded and set in DCSQLSTATE. The application program references DCSQLSTATE to determine the execution state of the DCSQL statement. The application program is allowed to assemble processing based on this information.

When the DCSQL statement is executed, the server system sets the DCSQLSTATE value.

DCSQLSTATE must be defined in an embedded DCSQL declare section as a 5 character with variable name DCSQLSTATE alphanumeric data item.

DCSQLMSG

When the DCSQL statement is executed, a message indicating a successful or failed execution is stored in DCSQLMSG. The user can display or print the message stored in DCSQLMSG using the output statement.

To use DCSQLMSG, an alphanumeric data item with variable name DCSQLMSG must be defined in embedded DCSQL declarative sections. The maximum length of DCSQLMSG is 255 bytes.

Procedure Division

In the Procedure Division, the statements defining the connection or the disconnection to the database, the transaction management, and the transmission and receiving of the data are described.

Chapter 10. Micro Focus Native Functions

This chapter describes Micro Focus native functions and formats.

Screen Functions

Environment Division

Special-names Paragraph

The special-names paragraph defines symbolic-constants, symbolic-characters, alphabet-names, character sets, and collating sequences, and associates function-names with mnemonic-names.

The CONSOLE IS CRT clause can be specified as a screen handling module.

[Format]

```
SPECIAL-NAMES.
    [CONSOLE IS CRT clause]
```

General Rules

All clauses in the special-names paragraph apply to a program explicitly or implicitly specifying them and their internal programs.

CONSOLE IS CRT Clause

The CONSOLE IS CRT clause defines the implicitly-specified input-output destination.

[Format]

<u>CONSOLE</u> IS <u>CRT</u>

The CONSOLE IS CRT clause regards operands in the ACCEPT statement where the FROM clause is not specified, and operands in the DISPLAY statement where the UPON clause is not specified, as screen items.

Data Division

Screen Data Description Entries

[Format 1] Defines a group screen item.

```
level-number ⎡ data-name-1          ⎤
             ⎣ FILLER               ⎦
             [ AUTO      clause]
             [ BACKGROUND-COLOR   clause]
             [ BLANK   SCREEN   clause]
             [ BLANK   clause]
             [ FOREGROUND-COLOR   clause]
             [ FULL      clause]
             [ GRID      clause]
             ⎡ HIGHLIGHT  clause ⎤
             ⎣ LOWLIGHT   clause ⎦
             [ LEFTLINE   clause]
             [ OCCURS     clause]
             [ OVERLINE   clause]
             [ PROMPT     clause]
             [ REQUIRED   clause]
             [ REVERSE-VIDEO   clause]
             [ SECURE     clause]
             [ SIGN       clause]
             [ UNDERLINE  clause]
             [ USAGE      clause]
             [ ZERO-FILL  clause].
```

[Format 2] Defines literal items.

```
level-number [ data-name-1       ]
             [ FILLER            ]
             [ BACKGROUND-COLOR   clause]
             [ BELL      clause]
             [ BLANK   LINE   clause      ]
             [ BLANK   SCREEN    clause   ]
             [ BLANK   clause]
             [ COLUMN      NUMBER    clause]
             [ ERASE    clause]
             [ FOREGROUND-COLOR    clause]
             [ GRID      clause]
             [ HIGHLIGHT  clause ]
             [ LOWLIGHT   clause ]
             [ LEFTLINE      clause]
             [ LINE  NUMBER clause]
             [ OCCURS        clause]
             [ OVERLINE     clause]
             [ REVERSE-VIDEO     clause]
             [ SIZE       clause]
             [ UNDERLINE  clause]
               VALUE   clause.
```

[Format 3] Defines input items, output items, or update items.

```
level-number ┌ data-name-1      ┐
             └ FILLER           ┘
             [ AUTO     clause]
             [ BACKGROUND-COLOR  clause]
             [ BELL     clause]
             ┌ BLANK  LINE  clause     ┐
             └ BLANK  SCREEN  clause   ┘
             [ BLANK  WHEN   ZERO    clause]
             [ BLANK  clause]
             [ COLUMN     NUMBER   clause]
             [ ERASE    clause]
             [ FOREGROUND-COLOR  clause]
             [ FULL     clause]
             [ GRID     clause]
             ┌ HIGHLIGHT  clause   ┐
             └ LOWLIGHT   clause   ┘
             [ JUSTIFIED   clause]
             [ LEFTLINE    clause]
             [ LINE  NUMBER      clause]
             [ OCCURS      clause]
             [ OVERLINE    clause]
             [ PICTURE     clause]
             [ PROMPT      clause]
             [ REQUIRED    clause]
             [ REVERSE-VIDEO     clause]
             [ SECURE      clause]
             [ SIGN        clause]
             [ SIZE        clause]
             [ UNDERLINE   clause]
             [ USAGE       clause]
             [ ZERO-FILL   clause].
```

Syntax Rules

See the section titled “Screen Data description Entry” for syntax rules.

General Rules

1. Use the PICTURE clause to classify an input item, output item, or update item.
2. When the SECURE clause is specified in a group screen item, all input items belonging to that group screen item apply.
3. When one of the following clauses is specified in a group screen item, the clause applies to all input items and update items belonging to the specified group screen item:
   a. AUTO clause
   b. FULL clause
   c. REQUIRED clause
   d. PROMPT clause
   e. ZERO-FILL clause
4. In LINE NUMBER and COLUMN NUMBER clauses, do not specify overlapping areas on the screen. Also, do not specify areas exceeding the physical screen area.

COLUMN NUMBER Clause

The COLUMN NUMBER clause specifies the rows where screen items are arrayed. + and - can be used to specify relative row numbers.

[Format]

$$\underline{\text{COLUMN}}\ \ \text{NUMBER}\ \ \text{IS}\left[\begin{array}{c}\underline{\text{PLUS}}\\ +\\ -\end{array}\right]\left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\}$$

The format and rules for + and - conform to PLUS. See the section titled “COLUMN NUMBER clause” for syntax rules and general rules.

GRID Clause

The GRID clause specifies a vertical line appended to the left of each character in a screen item.

[Format]

$\underline{\text{GRID}}$

Syntax Rules

1. The GRID clause can be specified for any screen item.
2. When the GRID clause is specified at the group item level all elementary items belonging to that group item applies.

General Rules

When the GRID clause is specified for a screen item, a vertical line is appended to the left of each character in the displayed screen item .

LEFTLINE Clause

The LEFTLINE clause specifies a vertical line appended to the left of the character at the high order end of a screen item.

[Format]

LEFTLINE

Syntax Rules

1. The LEFTLINE clause can be specified for any screen item.
2. When the LEFTLINE clause is specified at the group item level, all elementary items belonging to that group item applies.

General Rules

When the LEFTLINE clause is specified for a screen item, a vertical line is appended to the left of the character at the high order end of the displayed screen item .

LINE NUMBER Clause

The LINE NUMBER clause specifies lines where screen items are arrayed. + and - can be used to specify relative line numbers.

[Format]

$$\underline{\text{LINE}}\ \ \text{NUMBER}\ \ \text{IS}\ \begin{bmatrix}\underline{\text{PLUS}}\\ +\\ -\end{bmatrix}\begin{Bmatrix}\text{identifier-1}\\ \text{integer-1}\end{Bmatrix}$$

Syntax Rules

The format and rules for + and - conform to PLUS. See the section titled “LINE NUMBER clause” for syntax rules and general rules.

OCCURS Clause

The OCCURS clause defines a same data structure repetition.

See the section titled “OCCURS clause” for details.

[Format]

$\underline{\text{OCCURS}}$ integer-1 TIMES

Syntax Rules

The OCCURS clause can be specified for any screen item.

General Rules

1. When the OCCURS clause is specified, the screen items belonging to that group item are defined repeatedly integer-1 times.
2. When the absolute position is specified in an elementary item belonging to the group item where the OCCURS clause is specified, the same screen position is specified repeatedly for the elementary item.

OVERLINE Clause

The OVERLINE clause specifies a horizontal line at the top of a screen item.

[Format]

```
    OVERLINE
```

Syntax Rules

1. The OVERLINE clause can be specified for any screen item.
2. When the OVERLINE clause is specified at the group item level, all elementary items belonging to that group item applies.

General Rules

When the OVERLINE clause is specified for a screen item, a horizontal line is appended at the top of each character screen item.

PROMPT Clause

The PROMPT clause replaces blank character positions in a screen item by filler characters.

[Format]

$$\underline{\text{PROMPT}} \left[\text{CHARACTER IS} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{integer-1} \end{array} \right\} \right]$$

Syntax Rules

1. The PROMPT clause can be specified for an input item and update item.
2. When the PROMPT clause is specified at the group item level, it applies to input items and update items belonging to that group item.
3. The filler character, identifier-1 must be one alphabetic, alphanumeric, or national character.
4. The filler character literal-1, must be one mnemonic literal, national literal, or figurative constant.

General Rules

1. When the PROMPT clause is specified, blank characters are replaced with filler characters in the input item or update item.
2. Characters displayed with the PROMPT clause are not validated as input data.
3. When identifier-1 or literal-1 does not conform to the category specified in the PICTURE clause, system operation cannot be guaranteed.

4. If no filler CHARACTER is specified, the PROMPT clause is ignored.

SIZE Clause

The SIZE clause specifies the screen item logical size.

[Format]

$$\underline{\text{SIZE}}\quad \text{IS} \left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\}$$

Syntax Rules

1. The SIZE clause can only be written for elementary items.
2. The SIZE clause must be specified as an integer.

General Rules

1. When the value specified in the SIZE clause is 0 or a negative integer, the SIZE clause is ignored.
2. When the value specified in the SIZE clause is less than the value implicitly indicated by the related PICTURE clause or VALUE clause, only that screen item part from the left end of the specified size is validated. If the JUSTIFIED clause is concurrently specified, that screen item part from the right end of the specified size is validated.
3. When the value specified in the SIZE clause is greater than the value implicitly indicated by the related PICTURE clause or VALUE clause, spaces are displayed to the left of the screen item.

ZERO-FILL Clause

The ZERO-FILL clause specifies non-input characters are replaced with zeros instead of spaces.

[Format]

ZERO-FILL

Syntax Rules

1. The ZERO-FILL clause can be specified for an input item or update item.
2. When the ZERO-FILL clause is specified at the group item level, input items and update items belonging to that group item applies.

General Rules

When the ZERO-FILL clause is specified, if the data item size received from the screen is less than the specified screen item size, all excessive spaces to the left are filled with zeros. If the JUSTIFIED clause is concurrently specified, all excessive spaces to the left are filled with zeros.

Procedure Division

ACCEPT Statement (Screen Operation)

The ACCEPT statement enters data from the screen.

[Format 1]

$$\underline{\text{ACCEPT}}\ \text{data-name-1}$$
$$\left[\begin{array}{l} \text{AT}\left\{\begin{vmatrix} \underline{\text{LINE}}\ \text{NUMBER}\left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\} \\ \underline{\text{COLUMN}}\ \text{NUMBER}\left\{\begin{array}{l}\text{identifier-2}\\ \text{integer-2}\end{array}\right\}\end{vmatrix}\right\} \\ \underline{\text{AT}}\left\{\begin{vmatrix}\text{identifier-3}\\ \text{integer-3}\end{vmatrix}\right\}\end{array}\right]$$
$$\begin{array}{l}[\text{ON}\ \underline{\text{EXCEPTION}}\ \text{imperative-statement-1}]\\ [\underline{\text{NOT}}\ \text{ON}\ \underline{\text{EXCEPTION}}\ \text{imperative-statement-2}]\\ [\underline{\text{END-ACCEPT}}]\end{array}$$

[Format 2]

$$\underline{\text{ACCEPT}}\ \text{identifier -1}$$
$$\left[\begin{array}{l} \text{AT}\left\{\begin{vmatrix} \underline{\text{LINE}}\ \text{NUMBER}\left\{\begin{array}{l}\text{identifier-2}\\ \text{integer-1}\end{array}\right\} \\ \underline{\text{COLUMN}}\ \text{NUMBER}\left\{\begin{array}{l}\text{identifier-3}\\ \text{integer-2}\end{array}\right\}\end{vmatrix}\right\} \\ \underline{\text{AT}}\left\{\begin{vmatrix}\text{identifier-4}\\ \text{integer-3}\end{vmatrix}\right\}\end{array}\right]$$
$$\begin{array}{l}[\underline{\text{FROM}}\ \underline{\text{CRT}}]\\ [\underline{\text{MODE}}\ \text{IS}\ \underline{\text{BLOCK}}]\end{array}$$

$$\left[\text{WITH} \left\{ \begin{array}{l}
\underline{\text{AUTO}} \\
\underline{\text{BELL}} \\
\underline{\text{BLINK}} \\
\underline{\text{FULL}} \\
\underline{\text{GRID}} \\
\left\{ \begin{array}{l} \underline{\text{HIGHLIGHT}} \\ \underline{\text{LOWLIGHT}} \end{array} \right\} \\
\underline{\text{LEFTLINE}} \\
\underline{\text{OVERLINE}} \\
\underline{\text{PROMPT}} \left[\text{CHARACTER IS} \left\{ \begin{array}{l} \text{identifier-5} \\ \text{literal-1} \end{array} \right\} \right] \\
\underline{\text{REQUIRED}} \\
\underline{\text{REVERSE-VIDEO}} \\
\underline{\text{SEQURE}} \\
\underline{\text{SIZE}} \text{ IS } \left\{ \begin{array}{l} \text{identifier-6} \\ \text{integer-4} \end{array} \right\} \\
\underline{\text{UNDERLINE}} \\
\underline{\text{FOREGROUND-COLOER}} \text{ IS integer-5} \\
\underline{\text{BACKGROUND-COLOER}} \text{ IS integer-6} \\
\underline{\text{LEFT-JUSTIFY}} \\
\underline{\text{RIGHT-JUSTIFY}} \\
\underline{\text{SPACE-FILL}} \\
\underline{\text{TRAILING-SIGN}} \\
\underline{\text{UPDATE}} \\
\underline{\text{ZERO-FILL}}
\end{array} \right\} \ldots \right]$$

$$\begin{array}{l}
[\text{ON } \underline{\text{EXCEPTION}} \text{ imperative-statement-1}] \\
[\underline{\text{NOT}} \text{ ON } \underline{\text{EXCEPTION}} \text{ imperative-statement-2}] \\
[\underline{\text{END-ACCEPT}}]
\end{array}$$

Syntax Rules

RULES FOR FORMAT 1

1. identifier-3 must be an unsigned zoned decimal item of four or six digits.
2. integer-3 must have four or six digits.
3. See the section titled “ACCEPT statement (screen operation)” for other syntax rules.

RULES FOR FORMAT 2

1. identifier-1 must be an item defined in the working-storage section.
2. identifier-1 must be a fixed-length group item or elementary item. Alphabetic characters, alphanumeric characters, zoned decimals, or national data items can be declared as elementary items.
3. Words and clauses following identifier-1 can be specified in any order.
4. integer-1, integer-2, and integer-3 must be unsigned.
5. identifier-2 and identifier-3 must be unsigned integer items.
6. identifier-4 must be an unsigned zoned decimal item of four or six digits.
7. integer-3 must have four or six digits.
8. The SPACE-FILL, ZERO-FILL, LEFT-JUSTIFY, RIGHT-JUSTIFY, and TRAILING-SIGN clauses can be specified only for elementary items.

General Rules

RULES FOR FORMAT 1

See the section titled “ACCEPT statement (screen operation)” for general rules.

RULES FOR FORMAT 2

1. The ACCEPT statement reads data from the area corresponding to the item specified in identifier-1 on the screen.
2. Press the Enter key to complete execution of the ACCEPT statement.
3. AT specifies the item position to be input on the physical screen.
4. When AT is not specified, the item is input from column 1 on line 1.
5. For a four-digit identifier-4, specify the line number in the first two digits and the column number in the remaining two digits. For a six-digit identifier-4, specify the line number in the first three digits and the column number in the remaining three digits.
6. For a four-digit integer-3, specify the line number in the first two digits and the column number in the remaining two digits. For a six-digit integer-3, specify the line number in the first three digits and the column number in the remaining three digits.
7. The FROM CRT specification indicates the ACCEPT statement is handled in format 2.
8. When identifier-1 is a variable-length item, it is always assumed to be maximum length.

9. When identifier-1 is a group item and the MODE IS BLOCK clause is specified, it is handled as one elementary item.

10. When identifier-1 is a group item and the MODE IS BLOCK clause is not specified, the elementary item is assumed to belong to that group item. However, fillers and items defined for purposes other than display are ignored.

11. The LEFT-JUSTIFY, RIGHT-JUSTIFY, SPACE-FILL, TRAILING-SIGN, and UPDATE specifications are treated as comments.

12. See the section titled “Screen data description entry” in chapter 5 or the section titled “Screen data description entry” in chapter 10 for other clauses.

Win32

CALL Statement (Inter-program Communication)

The CALL statement transfers control to another program in the run unit.

[Format 1] ON OVERFLOW phrase

$$\underline{\text{CALL}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} [\underline{\text{WITH}} \left\{\begin{array}{l}\underline{\text{C}}\\ \underline{\text{PASCAL}}\\ \underline{\text{STDCALL}}\end{array}\right\} \underline{\text{LINKAGE}}]$$

$$\left[\underline{\text{USING}} \left\{\begin{array}{l}[\text{BY } \underline{\text{REFERENCE}}]\ \{\text{identifier-2}\}\ \ldots\\ \text{BY } \underline{\text{CONTENT}} \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\}\ \ldots\\ \text{BY } \underline{\text{VALUE}} \left\{\begin{array}{l}\text{identifier-3}\\ \text{literal-3}\end{array}\right\}\ldots\end{array}\right\}\ \ldots\right]$$

$$[\text{ON } \underline{\text{OVERFLOW}} \text{ imperative-statement-1}]$$
$$[\underline{\text{END-CALL}}]$$

[Format 2] ON EXCEPTION phrase

$$\underline{\text{CALL}} \left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\end{array}\right\} [\underline{\text{WITH}} \left\{\begin{array}{l}\underline{\text{C}}\\ \underline{\text{PASCAL}}\\ \underline{\text{STDCALL}}\end{array}\right\} \underline{\text{LINKAGE}}]$$

$$\left[\underline{\text{USING}} \left\{\begin{array}{l}[\text{BY } \underline{\text{REFERENCE}}]\ \{\text{identifier-2}\}\ \ldots\\ \text{BY } \underline{\text{CONTENT}} \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\end{array}\right\}\ \ldots\\ \text{BY } \underline{\text{VALUE}} \left\{\begin{array}{l}\text{identifier-3}\\ \text{literal-3}\end{array}\right\}\ldots\end{array}\right\}\ \ldots\right]$$

$$[\text{ON } \underline{\text{EXCEPTON}} \text{ imperative-statement-1}]$$
$$[\underline{\text{NOT}} \text{ ON } \underline{\text{EXCEPTON}} \text{ imperative-statement-2}]$$
$$[\underline{\text{END-CALL}}]$$

Syntax Rules

1. literal-3 must be an nine-digit-or-less numeric literal.
2. See the section titled “CALL statement (inter-program communication)” for other syntax rules.

General Rules

See the section titled “CALL Statement (Inter-program Communication)” for general rules.

DISPLAY Statement (Screen)

The DISPLAY statement displays data on the screen.

[Format 1]

$$
\begin{array}{l}
\underline{\text{DISPLAY}}\ \text{data-name-1} \\
\quad\left[\begin{array}{l}
\text{AT}\left\{\left|\begin{array}{l}
\underline{\text{LINE}}\ \text{NUMBER}\left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\} \\
\underline{\text{COLUMN}}\ \text{NUMBER}\left\{\begin{array}{l}\text{identifier-2}\\ \text{integer-2}\end{array}\right\}
\end{array}\right|\right\} \\
\underline{\text{AT}}\left\{\left|\begin{array}{l}\text{identifier-3}\\ \text{integer-3}\end{array}\right|\right\}
\end{array}\right] \\
\quad[\underline{\text{END-DISPLAY}}]
\end{array}
$$

[Format 2]

```
DISPLAY { identifier-1 }
          literal-1

   [    { | LINE NUMBER { identifier-2 }       | } ]
   [ AT { |               integer-1            | } ]
   [    { |                                    | } ]
   [    { | COLUMN NUMBER { identifier-3 }     | } ]
   [    { |                 integer-2          | } ]
   [ AT { | identifier-4 | }                       ]
   [      | integer-3    |                         ]
   [ UPON { CRT       } ]
   [        CRT-UNDER   ]
   [MODE IS BLOCK]
   [      { BELL                                   }     ]
   [      { BLINK                                  }     ]
   [      {  GRID                                  }     ]
   [      { { HIGHLIGHT }                          }     ]
   [      {   LOWLIGHT                             }     ]
   [      { LEFTLINE                               }     ]
   [      { OVERLINE                               }     ]
   [ WITH { REVERSE-VIDEO                          } ... ] }...
   [      { SIZE IS { identifier-6 }               }     ]
   [      {           integer-4                    }     ]
   [      { UNDERLINE                              }     ]
   [      { FOREGROUND-COLOER IS integer-5         }     ]
   [      { BACKGROUND-COLOER IS integer-6         }     ]
   [      { BLANK { SCREEN }                       }     ]
   [      {         LINE                           }     ]
   [END-DISPLAY]
```

Syntax Rules

RULES FOR FORMAT 1

1. identifier-3 must be an unsigned zoned decimal item containing four or six digits.
2. integer-3 must have four or six digits.
3. See the section titled “DISPLAY statement (screen operation)” for other syntax rules.

RULES FOR FORMAT 2

1. identifier-1 must be an item defined in the working-storage section.
2. identifier-1 must be a fixed-length group item or elementary item. Alphabetic characters, alphanumeric characters, zoned decimal, or national data item can be declared as elementary items.
3. Words and clauses following identifier-1 can be specified in any order.
4. integer-1, integer-2, and integer-3 must be unsigned.
5. identifier-2 and identifier-3 must be unsigned integer items.
6. identifier-4 must be an unsigned zoned decimal item containing four or six digits.
7. integer-3 must have four or six digits.

General Rules

RULES FOR FORMAT 1

See the section titled “DISPLAY statement (screen)” for general rules.

RULES FOR FORMAT 2

1. The DISPLAY statement displays data on the screen in the area corresponding to the item specified in identifier-1.
2. AT specifies the physical screen position of the item to be input.
3. When AT is not specified, the item is displayed from column 1 on line 1.
4. For four-digit identifier-4, specify the line number in the first two digits and the column number in the remaining two digits. For six-digit identifier-4, specify the line number in the first three digits and the column number in the remaining three digits.
5. For four-digit integer-3, specify the line number in the first two digits and the column number in the remaining two digits. For six-digit integer-3, specify the line number in the first three digits and the column number in the remaining three digits.
6. When UPON CRT is specified, the DISPLAY displays in format 2.
7. In addition, when UPON CRT-UNDER indicates the DISPLAY statement is handled in format 2, displayed items are underlined.

8. When identifier-1 is a variable-length item, it is always assumed to be maximum length.

9. When identifier-1 is a group item and the MODE IS BLOCK clause is specified, it is displayed as one elementary item.

10. When identifier-1 is a group item and MODE IS BLOCK clause is not specified, the elementary item is assumed to belong to that group item. However, fillers and items defined for purposes other than display are ignored.

11. See the section titled “Screen Data Description Entry” in Chapter 5 or the section titled “Screen Data Description Entry” in Chapter 10 for other clauses.

National Functions

Procedure Division

Class Condition

The class condition is used to check the class of the data item contents.

JAPANESE check can be specified.

[Format]

$$\text{identifier-1 IS [}\underline{\text{NOT}}\text{]} \left\{ \begin{array}{l} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \\ \underline{\text{ALPHABETIC-LOWER}} \\ \underline{\text{JAPANESE}} \\ \underline{\text{BOOLEAN}} \\ \text{class-name} \end{array} \right\}$$

1. The JAPANESE check is true when the data item specified in identifier-1 contains national characters or half-width kana characters.
2. identifier-1 must be one of the following:
   a. Data items for display
   b. Alphanumeric and national function-identifiers
3. When specifying a JAPANESE check, the following data items cannot be specified in identifier-1.
   a. Item specified as numeric characters or Boolean expressions

Input-Output Statement

Procedure Division

READ Statement (Relative File and Indexed File)

The READ statement reads records from a file.

[Format 1] Reads records sequentially (from relative file and indexed file).

```
READ  file-name-1 [ { NEXT     } ] RECORD
                  [ { PREVIOUS } ]
    [INTO  identifier-1]
    [WITH  [NO]  LOCK]
    [AT  END  imperative-statement-1]
    [NOT  AT  END  imperative-statement-2]
    [END-READ]
```

[Format 2] Reads records at any position (from relative file).

```
READ  file-name-1  RECORD
    [INTO  identifier-1]
    [WITH  [NO]  LOCK]
    [INVALID  KEY  imperative-statement-3]
    [NOT  INVALID  KEY  imperative-statement-4]
    [END-READ]
```

[Format 3] Reads records at any position (from indexed file).

```
READ  file-name-1  RECORD
    [INTO  identifier-1]
    [WITH  [NO]  LOCK]
    [KEY  IS  {data-name-1} ...]
    [INVALID  KEY  imperative-statement-3]
    [NOT  INVALID  KEY  imperative-statement-4]
    [END-READ]
```

Syntax Rules

RULES COMMON TO EACH FILE

See the section titled “READ statement (sequential file, relative file, and indexed file)”.

RULES FOR THE RELATIVE FILE

1. Format 1 is used to access a sequential file. PREVIOUS cannot be specified for a sequential file.
2. Format 1 is used to access records in a dynamic access file, using NEXT to read records sequentially.
3. Format 1 is used to access records in a dynamic access file, using PREVIOUS to read records in reverse sequence.
4. Format 2 is used to access records in a random access file and for random access to records in a dynamic access file.

RULES FOR AN INDEXED FILE

1. Format 1 is used to access a sequential access file. PREVIOUS cannot be specified for sequential access.
2. Format 1 is used to access records in a dynamic access file, using NEXT to read records sequentially.
3. Format 1 is used to access records in a dynamic access file, using PREVIOUS to read records in reverse sequence.
4. Format 3 is used to access records in a random access file and for random access to records in a dynamic access file.

General Rules

RULES COMMON TO FORMATS 1 TO 3

See the section titled “READ Statement (Sequential File, Relative File, and Indexed File).

RULES FOR FORMAT 1

Rules Common to File Types

1. When sequential access relative file, or sequential access indexed file is specified for file-name-1, the NEXT phrase can be omitted. The NEXT phrase does not affect the READ statement execution.

2. When a sequential file, sequential access relative file, or sequential access indexed file is specified, the next record is accessed in the file specified for file-name-1.

3. When a dynamic access relative file or dynamic access indexed file is specified in file-name-1, NEXT specifies the next record in the specified file be read, and PREVIOUS specifies the preceding record in the specified file be read.

4. When the READ statement is executed, the file position identifier value is determined according to the usable record defining rules . The operation to be performed next is determined by the file position indicator value.

Rules for Determining Usable Records when PREVIOUS is not Specified

See the section titled “READ statement (sequential file, relative file, and indexed file)”.

Rules for Determining Usable Records when PREVIOUS is Specified (for a Relative File)

1. The file position indicator value to start READ statement execution is used to define usable records according to the following rules. The relative key number is used for record relation in a relative file.

   a. When the file position indicator indicates there is no effective preceding record, the READ statement is not executed successfully.

   b. When the file position indicator indicates there is no optional file, an at end condition occurs.

   c. When the file position indicator was set by a previously-executed OPEN statement, an at end condition occurs.

   d. When the file position indicator was set by a previously-executed START statement, the record with a relative record number equal to the file position indicator is selected.

   e. When the file position indicator was set by a previously-executed READ statement, the first record with a relative record number smaller than the file position indicator is selected.

2. If a record satisfying conditions (1)(d) or (e) is not found, an at end condition occurs. A value indicating there is no next record is set in the file position indicator.

3. When a record satisfying conditions (1) (d) or (e) (called a selected record) is found, the following processing is performed:

a. When a RELATIVE KEY phrase is declared for file-name-1 and the number of significant digits in the relative record number of the selected record is greater than the relative key data item size, a value indicating this condition is set in the file position indicator and an at end condition occurs.

   When the number of significant digits in the relative record number is greater than the relative key item size, the selected record is made usable in the record area specified in file-name-1. In this case, the relative record number of the usable record is set in the file position indicator. The relative record number of the usable record is transcribed to the relative key item according to the transcription rules.

b. When a RELATIVE KEY phrase is declared in the ACCESS MODE clause specified in file-name-1, the selected record is made usable in the record specified in file-name-1. In this case, the relative record number of the usable record is set in the file position indicator.

Rules for Determining a Usable Record when a PREVIOUS Phrase is Declared (for an Indexed File)

1. The file position indicator value to start executing the READ statement is used for defining usable records. Records in the indexed file are compared using the current reference key values collating sequence.

   a. When the file position indicator indicates there is no preceding record, the READ statement is not executed successfully.

   b. When the file position indicator indicates there is no optional file, an at end condition occurs.

c. When the file position indicator was set by a previously-executed OPEN statement, an L at end condition occurs.

d. When the file position indicator was set by a previously-executed START statement, a record satisfying one of the following conditions is selected:

   A record positioned by the START statement

   When a record satisfying this condition is not found or is not the selection object, the record logically following or preceding that record is selected.

   Whether the record following or preceding that record is selected depends on the relation condition specified in the START statement.

   However, if the file position indicator is set by the START statement with a REVERSED ORDER phrase, the record logically preceding that record is selected. This selected record is logically the next record in the direction of the end record to the first record in the file.

e. When the file position indicator was set by a previously-executed READ statement, a record satisfying one of the following conditions is selected:

   When the reverse search direction is specified by executing the START statement with REVERSED ORDER specified, or when PREVIOUS has been specified, the record logically preceding the record made usable by the previously-executed READ statement is selected. This record is logically the next record in the direction of the end record to the first record in the file.

   When the reverse search direction is specified by executing the START statement with REVERSED ORDER specified and PREVIOUS has also been specified, the record logically following the record made usable by the previously-executed READ statement is selected.

2. If a record satisfying conditions (1) (d) or (e) is not found, an at end condition occurs. A value indicating there is no next record is set in the file position indicator.

3. When a record satisfying condition (1) (d) or (e) is found, the record is made usable in the record area specified in file-name-1 and the current reference key value of the record made usable is set in the file position indicator.

4. When the READ statement is not executed successfully, no reference key value is set.

5. When the READ statement is executed using the prime record key or alternate record key as the reference key, the record holding the reference key value may be duplicated. In this case, records are accessed in the following order:

   a. When the file position indicator has been set by a previously-executed START statement with REVERSED ORDER specified, or when PREVIOUS has been specified in the READ statement with the file positioned on the last record when duplicated records created by the relation conditions specified in the START statement exist, records are accessed in reverse order to the order used when records having duplicated reference key values are created using the WRITE or REWRITE statement.

   b. In other cases, records are accessed in the order in which records with duplicated reference key values are created using the WRITE or REWRITE statement.

6. When the READ statement in format 1 is executed after execution of a READ statement in format 3 for a dynamic access file, the reference key set by executing the READ statement in format 3 is used in the READ statement in format 1 until the reference key is changed, and records are accessed in the following search direction:

   a. Records are accessed in normal order until a START statement with a REVERSED ORDER phrase is executed.

   b. When a START statement with a REVERSED ORDER phrase is executed, records are accessed in reverse order.

   c. When PREVIOUS is specified, the record logically preceding that file is accessed. However, if REVERSED ORDER has been specified with the preceding START statement, so records are logically searched in reverse order, records are accessed in normal order.

Transfer of Control for an Exceptional Condition

1. If an at end condition occurs during READ statement execution, the READ statement is not executed successfully. After the value indicating the at end condition is set in the i-o status of file-name-1, control is transferred according to the rules described in the section titled “AT-END phrase”.

2. During READ statement execution, if the file position indicator indicates the next record is not found, when the number of significant digits in the relative record number is greater than the size of the relative key item, or when no optional file exists, the following operations are performed in this order. However, if PREVIOUS is specified, the following operations are performed in this order when the file position indicator indicates no preceding record is found or there is no optional file:

   a. The file position indicator and i-o status of file-name-1 are set.

   b. Control is transferred according to the rules described in the section titled “AT END specification”.

3. If neither an at end condition nor any other exceptional conditions occur during execution of the READ statement, the following processing is performed in this order:

   a. The file position indicator and i-o status of file-name-1 are set.

   b. Records read are made usable in the record area. When INTO is specified, they can be implicitly transcribed.

   c. Control is transferred according to rules described in the section titled “AT END specification”.

RULES FOR FORMAT 2 AND FORMAT 3

See the section titled “READ statement (sequential file, relative file, and indexed file)”.

START Statement (Relative File)

The START statement logically positions a file for sequential access to records.

[Format]

```
START file-name-1
    [        { IS EQUAL TO                          }               ]
    [        { IS =                                 }               ]
    [        { IS GREATER THAN                      }               ]
    [        { IS >                                 }               ]
    [        { IS NOT LESS THAN                     }               ]
    [        { IS NOT <                             }               ]
    [        { IS GREATER THAN OR EQUAL TO          }               ]
    [  KEY   { IS >=                                }  data-name-1  ]
    [        { IS LESS THAN                         }               ]
    [        { IS <                                 }               ]
    [        { IS NOT GREATER THAN                  }               ]
    [        { IS NOT >                             }               ]
    [        { IS LESS THAN OR EQUAL TO             }               ]
    [        { IS <=                                }               ]
  [INVALID KEY imperative-statement-1]
  [NOT INVALID KEY imperative-statement-2]
  [END-START]
```

Remarks: “=“, “>“, “<“, “>=“, and “<=“ are key words. These symbols are not underlined to avoid confusion with other symbols.

See the section titled “START statement (relative file)” for both syntax rules and general rules.

START Statement (Indexed File)

The START statement logically positions a file for sequential access to records.

[Format 1] Logically positions a file for sequential access of records in normal order.

```
START file-name-1

 [        { IS EQUAL TO                        }                     ]
 [        { IS =                               }                     ]
 [        { IS GREATER THAN                    }                     ]
 [        { IS >                               }                     ]
 [        { IS NOT LESS THAN                   }                     ]
 [        { IS NOT <                           }                     ]
 [        { IS GREATER THAN OR EQUAL TO        }                     ]
 [  KEY   { IS >=                              }  {data-name-1} ...  ]
 [        { IS LESS THAN                       }                     ]
 [        { IS <                               }                     ]
 [        { IS NOT GREATER THAN                }                     ]
 [        { IS NOT >                           }                     ]
 [        { IS LESS THAN OR EQUAL TO           }                     ]
 [        { IS <=                              }                     ]

   [INVALID KEY imperative-statement-1]
   [NOT INVALID KEY imperative-statement-2]
   [END-START]
```

[Format 2] Logically positions a file for sequential access to records in reverse order.

```
START file-name-1
  ⎡        ⎧ IS EQUAL TO                      ⎫                    ⎤
  ⎢        ⎪ IS =                             ⎪                    ⎥
  ⎢        ⎪ IS LESS THAN                     ⎪                    ⎥
  ⎢        ⎪ IS <                             ⎪                    ⎥
  ⎢  KEY   ⎨ IS NOT GREATER THAN              ⎬ {data-name-1}...   ⎥
  ⎢        ⎪ IS NOT >                         ⎪                    ⎥
  ⎢        ⎪ IS LESS THAN OR EQUAL TO         ⎪                    ⎥
  ⎣        ⎩ IS <=                            ⎭                    ⎦
    WITH REVERSED ORDER
  [INVALID KEY imperative-statement-1]
  [NOT INVALID KEY imperative-statement-2]
  [END-START]
```

[Format 3] Logically positions a file on the record at the start or the end of the reference key.

```
START file-name-1

    FIRST RECORD

    [KEY IS {data-name-1}...]

    [WITH REVERSED ORDER]

    [INVALID KEY imperative-statement-1]

    [NOT INVALID KEY imperative-statement-2]

    [END-START]
```

Remarks:
“=“, “>“, “<“, “>=“, and “<=“ are key words. These symbols are not underlined to avoid confusion with other symbols.

Syntax Rules

See the section titled "START statement (indexed file)".

General Rules

RULES COMMON TO FORMATS 1 TO 3

See the section titled "START statement (indexed file)"

RULES COMMON TO FORMATS 1 AND 2

1. The record key contained in the file with file-name-1 is compared with the following data items using the conditions specified in the KEY phrase.

   a. When the KEY phrase is specified, the record key is compared with the data item with data-name-1.

   b. When the KEY phrase is omitted, the record key is compared with the data item written in the RECORD KEY clause of file-name-1. "IS EQUAL TO" is assumed as the relational operator for the KEY phrase.

2. In the comparison described in (1), data items are compared with the record reference keys in the file arranged in ascending order according to the character collating sequence. When operands have different lengths, the right side of the longer operand is truncated to be equal to the length of the shorter operand for mnemonic comparison. As a result of the comparison, the following processing is performed:

   a. In format 1, the value indicated by the first or last record satisfying the relation condition is set in the file position indicator. In this case, whether the value of the first or last record satisfies the relation condition depends on the relational operator.

b. In format 2, the value indicated by the last record satisfying the relation condition is set in the file position indicator.

c. If no records in the file satisfy these conditions, an invalid key condition occurs and the START statement is not executed successfully.

RULES FOR FORMAT 3

See the section titled “START Statement (Indexed File)”.

Appendix A. List of Reserved Words

The appendix lists COBOL reserved words.

Notations in the tables below

Column A: COBOL85

Column B: CODASYL 1988 (reference)

Column C: 85 ANSI (or 92 JIS)

Column D: 74 ANSI

O in columns A to D: Reserved word in each specification

Word	A	B	C	D
ACCEPT	O	O	O	O
ACCESS	O	O	O	O
ACTUAL	O			
ADD	O	O	O	O
ADDRESS	O			
ADVANCING	O	O	O	O
AFTER	O	O	O	O
ALL	O	O	O	O
ALPHABET	O	O	O	
ALPHABETIC	O	O	O	O
ALPHABETIC-LOWER	O	O	O	
ALPHABETIC-UPPER	O	O	O	
ALPHANUMERIC	O	O	O	
ALPHANUMERIC-EDITED	O	O	O	
ALSO	O	O	O	O
ALTER	O		O	O
ALTERNATE	O	O	O	O
AND	O	O	O	O
ANY	O	O	O	
APPLY	O			
ARE	O	O	O	O
AREA	O	O	O	O

Word	A	B	C	D
AREAS	O	O	O	O
ARITHMETIC	O	O		
AS	O			
ASCENDING	O	O	O	O
ASSIGN	O	O	O	O
AT	O	O	O	O
AUTHOR	O		O	O
AUTO	O			
AUTOMATIC	O			
B-AND	O	O		
B-EXOR	O	O		
B-LESS	O	O		
B-NOT	O	O		
B-OR	O	O		
BASED	O			
BACKGROUND-COLOR	O			
BASED-STORAGE	O			
BEFORE	O	O	O	O
BEGINNING	O			
BELL	O			
BINARY	O	O	O	
BIT	O	O		
BITS	O	O		
BLANK	O	O	O	O
BLINK	O			
BLOCK	O	O	O	O
BOOLEAN	O	O		
BOTTOM	O	O	O	O
BY	O	O	O	O
CALL	O	O	O	O
CANCEL	O	O	O	O
CBL	O			
CD	O	O	O	O
CF	O	O	O	O
CH	O	O	O	O
CHANGED	O			
CHARACTER	O	O	O	O
CHARACTERS	O	O	O	O
CLASS	O	O	O	
CLOCK-UNITS	O		O	O
CLOSE	O	O	O	O

Word	A	B	C	D
COBOL	O		O	O
CODE	O	O	O	O
CODE-SET	O	O	O	O
COLLATING	O	O	O	O
COLUMN	O	O	O	O
COMMA	O	O	O	O
COMMAND	O			
COMMIT	O	O		
COMMON	O	O	O	
COMMUNICATION	O	O	O	O
COM-REG	O			
COMP	O	O	O	O
COMP-1	O			
COMP-2	O			
COMP-3	O			
COMP-4	O			
COMP-5	O			
COMP-n		O		
COMPLEX	O			
COMPUTATIONAL	O	O	O	O
COMPUTATIONAL-1	O			
COMPUTATIONAL-2	O			
COMPUTATIONAL-3	O			
COMPUTATIONAL-4	O			
COMPUTATIONAL-5	O			
COMPUTATIONAL-n		O		
COMPUTE	O	O	O	O
CONFIGURATION	O	O	O	O
CONNECT	O	O		
CONSTANT	O			
CONTAINED	O	O		
CONTAINS	O	O	O	O
CONTENT	O	O	O	
CONTINUE	O	O	O	
CONTROL	O	O	O	O
CONTROL-CHARACTER	O			
CONTROLS	O	O	O	O
CONVERTING	O	O	O	
COPY	O	O	O	O
CORE-INDEX	O			
CORR	O		O	O

Word	**A**	**B**	**C**	**D**
CORRESPONDING	O		O	O
COUNT	O	O	O	O
CRP	O			
CRT	O			
CRT-UNDER	O			
CURRENCY	O	O	O	O
CURRENT	O	O		
CURSOR	O			
DATA	O	O	O	O
DATE	O	O	O	O
DATE-COMPILED	O		O	O
DATE-WRITTEN	O		O	O
DAY	O	O	O	O
DAY-OF-WEEK	O	O	O	
DB	O	O		
DB-ACCESS-CONTROL-KEY	O	O		
DB-DATA-NAME	O	O		
DB-EXCEPTION	O	O		
DB-RECORD-NAME	O	O		
DB-SET-NAME	O	O		
DB-STATUS	O	O		
DE	O	O	O	O
DEAD-LOCK	O			
DEBUG-CONTENTS	O		O	O
DEBUG-ITEM	O		O	O
DEBUG-LINE	O		O	O
DEBUG-NAME	O		O	O
DEBUG-SUB-1	O		O	O
DEBUG-SUB-2	O		O	O
DEBUG-SUB-3	O		O	O
DEBUGGING	O	O	O	O
DECIMAL-POINT	O	O	O	O
DECLARATIVES	O	O	O	O
DEFAULT	O	O		
DELETE	O	O	O	O
DELIMITED	O	O	O	O
DELIMITER	O	O	O	O
DEPENDING	O	O	O	O
DESCENDING	O	O	O	O
DESTINATION	O	O	O	O
DESTINATION-1	O			

Word	A	B	C	D
DESTINATION-2	O			
DESTINATION-3	O			
DETAIL	O	O	O	O
DEVICE	O			
DIRECT	O			
DISABLE	O	O	O	O
DISCONNECT	O	O		
DISJOINING	O			
DISPLAY	O	O	O	O
DISPLAY-1	O			
DISPLAY-EXIT	O			
DISPLAY-n		O		
DIVIDE	O	O	O	O
DIVISION	O	O	O	O
DOWN	O	O	O	O
DUPLICATE	O	O		
DUPLICATES	O	O	O	O
DYNAMIC	O	O	O	O
EDIT-COLOR	O			
EDIT-CURSOR	O			
EDIT-MODE	O			
EDIT-OPTION	O			
EDIT-STATUS	O			
EGCS	O			
EGI	O	O	O	O
EJECT	O			
ELSE	O	O	O	O
EMI	O	O	O	O
EMPTY	O	O		
ENABLE	O	O	O	O
END	O	O	O	O
END-ACCEPT	O			
END-ADD	O	O	O	
END-CALL	O	O	O	
END-COMPUTE	O	O	O	
END-DELETE	O	O	O	
END-DISABLE		O		
END-DISPLAY	O			
END-DIVIDE	O	O	O	
END-ENABLE		O		
END-EVALUATE	O	O	O	

Word	A	B	C	D
END-EXEC	O			
END-IF	O	O	O	
END-MULTIPLY	O	O	O	
END-OF-PAGE	O	O	O	O
END-PERFORM	O	O	O	
END-READ	O	O	O	
END-RECEIVE	O	O	O	
END-RETURN	O	O	O	
END-REWRITE	O	O	O	
END-SEARCH	O	O	O	
END-SEND		O		
END-START	O	O	O	
END-STRING	O	O	O	
END-SUBTRACT	O	O	O	
END-TRANSCEIVE		O		
END-UNSTRING	O	O	O	
END-WRITE	O	O	O	
ENDCOBOL	O			
ENDING	O			
ENTER	O		O	O
ENTRY	O			
ENVIRONMENT	O	O	O	O
EOL	O			
EOP	O	O	O	O
EOS	O			
EQUAL	O	O	O	O
EQUALS	O	O		
ERASE	O	O		
ERROR	O	O	O	O
ESI	O	O	O	O
EVALUATE	O	O	O	
EVERY	O		O	O
EXACT		O		
EXAMINE	O			
EXCEEDS	O	O		
EXCEPTION	O	O	O	O
EXCLUSIVE	O	O		
EXEC	O			
EXIT	O	O	O	O
EXOR	O			
EXTEND	O	O	O	O

Word	**A**	**B**	**C**	**D**
EXTERNAL	O	O	O	
FALSE	O	O	O	
FD	O	O	O	O
FETCH	O	O		
FILE	O	O	O	O
FILE-CONTROL	O	O	O	O
FILE-LIMIT	O			
FILE-LIMITS	O			
FILES	O			
FILLER	O	O	O	O
FINAL	O	O	O	O
FIND	O	O		
FINISH	O	O		
FIRST	O	O	O	O
FLADD	O			
FOOTING	O	O	O	O
FOR	O	O	O	O
FOREGROUND-COLOR	O			
FORM		O		
FORMAT	O	O		
FORMATTED	O			
FREE	O	O		
FROM	O	O	O	O
FULL	O			
FUNCTION	O	O		
GENERATE	O	O	O	O
GET	O	O		
GIVING	O	O	O	O
GLOBAL	O	O	O	
GO	O	O	O	O
GOBACK	O			
GREATER	O	O	O	O
GRID	O			
GROUP	O	O	O	O
HEADING	O	O	O	O
HIGHLIGHT	O			
HIGH-VALUE	O	O	O	O
HIGH-VALUES	O	O	O	O
I-O	O	O	O	O
I-O-CONTROL	O	O	O	O
ID	O			

Word	**A**	**B**	**C**	**D**
IDENTIFICATION	O	O	O	O
IF	O	O	O	O
IN	O	O	O	O
INCLUDE	O			
INDEX	O	O	O	O
INDEX-n		O		
INDEXED	O	O	O	O
INDICATE	O	O	O	O
INITIAL	O	O	O	O
INITIALIZE	O	O	O	
INITIATE	O	O	O	O
INPUT	O	O	O	O
INPUT-OUTPUT	O	O	O	O
INSPECT	O	O	O	O
INSTALLATION	O		O	O
INTO	O	O	O	O
INVALID	O	O	O	O
IS	O	O	O	O
JAPANESE	O			
JOB	O			
JOINING	O			
JUST	O	O	O	O
JUSTIFIED	O	O	O	O
KANJI	O			
KEEP	O	O		
KEY	O	O	O	O
LABEL	O		O	O
LAST	O	O	O	O
LD	O	O		
LEADING	O	O	O	O
LEFT	O		O	O
LEFTLINE	O			
LEFT-JUSTIFY	O			
LENGTH	O	O	O	O
LESS	O	O	O	O
LIMIT	O	O	O	O
LIMITED	O			
LIMITS	O	O	O	O
LINAGE	O	O	O	O
LINAGE-COUNTER	O	O	O	O
LINE	O	O	O	O

Word	A	B	C	D
LINE-COUNTER	O	O	O	O
LINES	O	O	O	O
LINKAGE	O	O	O	O
LOCALLY	O	O		
LOCK	O	O	O	O
LOW-VALUE	O	O	O	O
LOW-VALUES	O	O	O	O
LOWLIGHT	O			
MANUAL	O			
MEMBER	O	O		
MEMORY	O		O	O
MERGE	O	O	O	O
MESSAGE	O	O	O	O
MODE	O	O	O	O
MODE-1	O			
MODE-2	O			
MODE-3	O			
MODIFY	O	O		
MODULES	O		O	O
MORE-LABELS	O			
MOVE	O	O	O	O
MULTICON	O			
MULTICONVERSATION-MODE	O			
MULTIPLE	O	O	O	O
MULTIPLY	O	O	O	O
NAMED	O			
NATIONAL	O			
NATIONAL-EDITED	O			
NATIVE	O	O	O	O
NEGATIVE	O	O	O	O
NEXT	O	O	O	O
NO	O	O	O	O
NOMINAL	O			
NONE	O			
NOT	O	O	O	O
NOTE	O			
NULL	O	O		
NULLS	O			
NUMBER	O	O	O	O
NUMERIC	O	O	O	O
NUMERIC-EDITED	O	O	O	

Word	**A**	**B**	**C**	**D**
OBJECT-COMPUTER	O	O	O	O
OCCURS	O	O	O	O
OF	O	O	O	O
OFF	O		O	O
OMITTED	O		O	O
ON	O	O	O	O
ONLY	O	O		
OPEN	O	O	O	O
OPTIONAL	O	O	O	O
OR	O	O	O	O
ORDER	O	O	O	
ORGANIZATION	O	O	O	O
OTHER	O	O	O	
OTHERWISE	O			
OUTPUT	O	O	O	O
OVERFLOW	O	O	O	O
OVERLINE	O			
OWNER	O	O		
PACKED-DECIMAL	O	O	O	
PADDING	O	O	O	
PAGE	O	O	O	O
PAGE-COUNTER	O	O	O	O
PARAGRAPH		O		
PASSWORD	O			
PERFORM	O	O	O	O
PF	O	O	O	O
PH	O	O	O	O
PIC	O	O	O	O
PICTURE	O	O	O	O
PLUS	O	O	O	O
POINTER	O	O	O	O
POSITION	O	O	O	O
POSITIONING	O			
POSITIVE	O	O	O	O
PREFIX	O			
PRESENT	O	O		
PREVIOUS	O			
PRINTING	O	O	O	O
PRIOR	O	O		
PROCEDURE	O	O	O	O
PROCEDURES	O		O	O

Word	A	B	C	D
PROCEED	O		O	O
PROCESSING	O			
PROGRAM	O	O	O	O
PROGRAM-ID	O	O	O	O
PROGRAM-STATUS	O			
PROMPT	O			
PROTECTED	O	O		
PURGE	O	O	O	
QUEUE	O	O	O	O
QUOTE	O	O	O	O
QUOTES	O	O	O	O
RANDOM	O	O	O	O
RANGE	O			
RD	O	O	O	O
READ	O	O	O	O
READY	O	O		
REALM	O	O		
RECEIVE	O	O	O	O
RECONNECT	O	O		
RECORD	O	O	O	O
RECORD-NAME	O	O		
RECORD-OVERFLOW	O			
RECORDING	O			
RECORDS	O	O	O	O
REDEFINES	O	O	O	O
REEL	O	O	O	O
REFERENCE	O	O	O	
REFERENCES	O		O	O
RELATION	O	O		
RELATIVE	O	O	O	O
RELEASE	O	O	O	O
RELOAD	O			
REMAINDER	O	O	O	O
REMARKS	O			
REMOVAL	O	O	O	O
RENAMES	O	O	O	O
REORG-CRITERIA	O			
REPEATED	O	O		
REPLACE	O	O	O	
REPLACING	O	O	O	O
REPORT	O	O	O	O

Word	A	B	C	D
REPORTING	O	O	O	O
REPORTS	O	O	O	O
RERUN	O		O	O
REQUIRED	O			
RESERVE	O	O	O	O
RESET	O	O	O	O
RETAINING	O	O		
RETRIEVAL	O	O		
RETURN	O	O	O	O
RETURN-CODE	O			
REVERSE-VIDEO	O			
REVERSED	O		O	O
REWIND	O	O	O	O
REWRITE	O	O	O	O
RF	O	O	O	O
RH	O	O	O	O
RIGHT	O	O	O	O
RIGHT-JUSTIFY	O			
ROLL-OUT	O			
ROLLBACK	O	O		
ROUNDED	O	O	O	O
RUN	O	O	O	O
SA	O			
SAME	O	O	O	O
SAVED-AREA	O			
SCREEN	O			
SD	O	O	O	O
SEARCH	O	O	O	O
SECTION	O	O	O	O
SECURITY	O		O	O
SECURE	O			
SEEK	O			
SEGMENT	O	O	O	O
SEGMENT-LIMIT	O		O	O
SELECT	O	O	O	O
SELECTED	O			
SELECTIVE	O			
SEND	O	O	O	O
SENTENCE	O	O	O	O
SEPARATE	O	O	O	O
SEQUENCE	O	O	O	O

Word	A	B	C	D
SEQUENTIAL	O	O	O	O
SERVICE	O			
SESSION	O			
SESSION-ID		O		
SET	O	O	O	O
SHARED	O	O		
SHIFT-IN	O			
SHIFT-OUT	O			
SIGN	O	O	O	O
SIMPLE	O			
SINGLE	O			
SIZE	O	O	O	O
SKIP1	O			
SKIP2	O			
SKIP3	O			
SORT	O	O	O	O
SORT-CONTROL	O			
SORT-CORE-SIZE	O			
SORT-FILE-SIZE	O			
SORT-MESSAGE	O			
SORT-MERGE	O	O	O	O
SORT-MODE-SIZE	O			
SORT-RETURN	O			
SORT-STATUS	O			
SOURCE	O	O	O	O
SOURCE-COMPUTER	O	O	O	O
SPACE	O	O	O	O
SPACE-FILL	O			
SPACES	O	O	O	O
SPECIAL-NAMES	O	O	O	O
STANDARD	O	O	O	O
STANDARD-1	O	O	O	O
STANDARD-2	O	O	O	
START	O	O	O	O
STATION	O			
STATIONS	O			
STATUS	O	O	O	O
STOP	O	O	O	O
STORE	O	O		
STRING	O	O	O	O
SUB-QUEUE-1	O	O	O	O

Word	**A**	**B**	**C**	**D**
SUB-QUEUE-2	O	O	O	O
SUB-QUEUE-3	O	O	O	O
SUBRANGE	O			
SUB-SCHEMA	O	O		
SUBSCHEMA-NAME	O			
SUBTRACT	O	O	O	O
SUCCESSIVE	O			
SUFFIX	O			
SUM	O	O	O	O
SUPPRESS	O	O	O	O
SYMBOLIC	O	O	O	O
SYNC	O		O	O
SYNCHRONIZED	O		O	O
TABLE	O	O	O	O
TALLY	O			
TALLYING	O	O	O	O
TAPE	O	O	O	O
TENANT	O			
TENNANT		O		
TERMINAL	O	O	O	O
TERMINATE	O	O	O	O
TEST	O	O	O	
TEXT	O	O	O	O
THAN	O	O	O	O
THEN	O	O	O	
THROUGH	O	O	O	O
THRU	O	O	O	O
TIME	O	O	O	O
TIMEOUT		O		
TIMES	O	O	O	O
TITLE	O			
TO	O	O	O	O
TOP	O	O	O	O
TRACE	O			
TRACK	O			
TRACK-AREA	O			
TRACK-LIMIT	O			
TRACK-OVERFLOW	O			
TRACKS	O			
TRAILING	O	O	O	O
TRAILING-SIGN	O			

Word	**A**	**B**	**C**	**D**
TRANSACTION	O			
TRANSCEIVE		O		
TRUE	O	O	O	
TYPE	O	O	O	O
UNDERLINE	O			
UNEQUAL	O	O		
UNIT	O	O	O	O
UNLOCK	O			
UNSTRING	O	O	O	O
UNTIL	O	O	O	O
UP	O	O	O	O
UPDATE	O	O		
UPON	O	O	O	O
USAGE	O	O	O	O
USAGE-MODE	O	O		
USE	O	O	O	O
USING	O	O	O	O
VALID	O	O		
VALIDATE	O	O		
VALUE	O	O	O	O
VALUES	O	O	O	O
VARYING	O	O	O	O
WAIT	O	O		
WHEN	O	O	O	O
WHEN-COMPILED	O			
WITH	O	O	O	O
WITHIN	O	O		
WORDS	O		O	O
WORKING-STORAGE	O	O	O	O
WRITE	O	O	O	O
WRITE-ONLY	O			
ZERO	O	O	O	O
ZERO-FILL	O			
ZEROES	O	O	O	O
ZEROS	O	O	O	O
+	O		O	O
-	O		O	O
*	O		O	O
/	O		O	O
**	O		O	O
>	O		O	O

Word	A	B	C	D
<	O		O	O
=	O		O	O
>=	O		O	
<=	O		O	
&	O			
->	O			

Appendix B. System Quantitative Restrictions

This appendix provides quantitative restrictions in the COBOL processing system. Restrictions on program operation depend on the operating system and the I-O control system.

Reference Format

Item	Value
Maximum length of variable-format (in bytes)	251

Nucleus Data Division

Item		**Value**
Maximum length of data item (in bytes)	32-Bit	2147483647
	16-Bit	64770
Maximum length of alphabetic and alphanumeric item (in characters)	32-Bit	2147483647
	16-Bit	64770
Maximum length of alphanumeric edited item (in characters)	32-Bit	2147483647
	16-Bit	64770
Maximum length of Boolean item (in characters)	32-Bit **S** **HP** **Sun** **Win32**	2147483647
	16-Bit	64770
Maximum length of numeric edited item (in characters)		160
Maximum length of national item (in characters)	32-Bit **Sun** **W**	1073741823
	16-Bit	32385
Maximum length of national edited item (in characters)	32-Bit	1073741823
	16-Bit	32385
Maximum value of subscript	32-Bit	2147483647
	16-Bit	64770
Maximum value of dimension of subscript		7
Maximum repeat count of OCCURS clause		2147483647
Maximum length of key item specified in the KEY IS phrase of OCCURS clause (in bytes)		256
Maximum number of keys for a table element		2147483647
Maximum number of index-names with the INDEX BY phrase in OCCURS clause		60
Maximum depth of implicit pointer qualification		7

Nucleus Procedure Division

Item	Value
Maximum size of identifier-1 when the function name CONSOLE is assigned to mnemonic-name in “ACCEPT identifier-1 FROM mnemonic-name” (in bytes)	No restriction
Maximum number of identifiers in an INITIALIZE statement	No restriction
Number of data items or variable-length items subscripted, pointed, or reference modified using the identifier in an INITIALIZE procedure	No restriction
Maximum size of identifier-6 and identifier-7 in "INSPECT CONVERTING identifier-6 TO identifier-7"(in bytes) - When identifier-6 and identifier-7 are neither national item nor national edited item - When identifier-6 and identifier-7 are national item and / or national edited item	 256 No restriction
Maximum value of identifier-1 or integer-1 in "PERFORM identifier-1 / integer-1 TIMES"	No restriction

Sequential File

Item	Value
Maximum record length (in bytes)	32760
Maximum number of lines per logical page specified in LINAGE clause	No restriction
Maximum file capacity (in bytes)	1073741824

Relative File

Item	Value
Maximum record length (in bytes)	32760
Maximum file capacity (in bytes)	1073741824

Indexed File

Item	Value
Maximum record length (in bytes)	32760
Maximum number of data-name in RECORD KEY clause	254 *1
Maximum value of the sum of length of data-name in RECORD KEY clause (in bytes)	254 *2
Maximum number of data-name in ALTERNATE RECORD KEY clause	254 *1
Maximum value of the sum of length of data-name in an ALTERNATE RECORD KEY clause (in bytes)	254 *2
Maximum number of ALTERNATE RECORD KEY clauses	125
Maximum file capacity (in bytes)	1073741824

*1 The total number of data-name in the RECORD KEY and ALTERNATE RECORD KEY clauses are maximum 255.

*2 The total length of data-name in the RECORD KEY and ALTERNATE RECORD KEY clauses are maximum 255.

Inter-program Communication

Item	Value
Maximum number of header parameters in the procedure division	No restriction
Maximum number of parameters having the USING phrase in the CALL statement	No restriction
Maximum number of parameters having the USING phrase in the ENTRY statement	No restriction

Sort and Merge

Item		Value
Maximum record length (in bytes)	32-Bit	32760
	16-Bit	21484
Maximum number of key data items specified in the SORT statement and the MERGE statement		64
Maximum value of the sum of length of key data items specified in the SORT statement and the MERGE statement (in bytes)		Maximum record length
Maximum length of alphabetic and alphanumeric item specified as a key data item in the SORT statement and the MERGE statement (in characters)		16382
Maximum length of national item specified as a key data item in the SORT statement and the MERGE statement (in characters)		8191
Maximum number of input files specified with USING in the SORT statement and the MERGE statement		16

Source Statement Manipulation

Item	Value
Maximum length of text words in the COPY statement (in characters)	324
Maximum length of word-4 in "COPY ... JOINING word-4" (in characters)	28
Maximum length of text word in the REPLACE statement	324

Presentation File

Item	Value
Maximum record length (in bytes)	32767

Appendix C. Code Tables

This appendix lists internal codes for characters in the following code systems:

- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- ASCII (American National Standard Code for Information Interchange X3.4 - 1968)
- JIS 8-bit Code (conforming to ISO 640)

EBCDIC, ASCII, and JIS 8-bit codes correspond to EBCDIC, STANDARD-1, and STANDARD-2, respectively in the ALPHABET clause in the special-names paragraph. Read the tables as follows:

- A character is represented using a one-byte internal code. Characters 0 to F in the high-order 4 bits and low-order 4 bits columns in the table represent bits 1 to 4 and bits 5 to 8 in hexadecimal notation. Characters in the table are represented in combinations of the high-order 4 bits and low-order 4 bits.
- The location of characters in the table shows the collating sequence. Characters in the left columns are greater than those in the right columns; and characters in lower rows are greater than those in higher rows.

Internal Codes for EBCDIC Characters

Low-order 4 Bits	High-order 4 Bits															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0					SP	&	-			*/1			{	}	$	0
1						*1			a *3	j *3	~		A	J		1
2						*1			b *3	k *3	s *3		B	K	S	2
3						*1			c *3	l *3	t *3		C	L	T	3
4						*1			d *3	m *3	u *3		D	M	U	4
5						*1			e *3	n *3	v *3		E	N	V	5
6					*1	*1			f *3	o *3	w *3		F	O	W	6
7					*1				g *3	p *3	x *3		G	P	X	7
8					*1				h *3	q *3	y *3		H	Q	Y	8
9					*1			`	i *3	r *3	z *3		I	R	Z	9
A					*2	!	¦	:	*1	*1	*1	*1				
B					.	$ \	,	#				*1				
C					<	*	%	@	*1		*1	*1				
D					(	)	_	‘	*1	*1	*1	*1				
E					+	;	>	=	*1	*1	*1	*1				
F					|	¬	?	“	*1	*1	*1	*1				

*1. Japanese character

*2. The pound sign

*3. Two kinds of characters are represented with the same internal code, one for lowercase, the other for Japanese Kana.

Internal Codes for ASCII Characters

Low-order 4 Bits	High-order 4 Bits							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0			SP	0	@	P	`	p
1			!	1	A	Q	a	q
2			“	2	B	R	b	r
3			#	3	C	S	c	s
4			$	4	D	T	d	t
5			%	5	E	U	e	u
6			&	6	F	V	f	v
7			‘	7	G	W	g	w
8			(	8	H	X	h	x
9			)	9	I	Y	I	y
A			*	:	J	Z	j	z
B			+	;	K	[	k	{
C			-	<	L	\	l	|
D			-	=	M	]	m	}
E			.	>	N	^	n	~
F			/	?	O	_	o	

Internal Codes for JIS 8-Bit Code Characters

Low-order 4 Bits	**High-order 4 Bits**															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0			SP	0	@	P	`	p				*1	*1	*1		
1			!	1	A	Q	a	q			*1	*1	*1	*1		
2			“	2	B	R	b	r			*1	*1	*1	*1		
3			#	3	C	S	c	s			*1	*1	*1	*1		
4			$	4	D	T	d	t			*1	*1	*1	*1		
5			%	5	E	U	e	u			*1	*1	*1	*1		
6			&	6	F	V	f	v			*1	*1	*1	*1		
7			‘	7	G	W	g	w			*1	*1	*1	*1		
8			(	8	H	X	h	x			*1	*1	*1	*1		
9			)	9	I	Y	I	y			*1	*1	*1	*1		
A			*	:	J	Z	j	z			*1	*1	*1	*1		
B			+	;	K	[	k	{			*1	*1	*1	*1		
C			,	<	L	\	l	|			*1	*1	*1	*1		
D			-	=	M	]	m	}			*1	*1	*1	*1		
E			.	>	N	^	n	~			*1	*1	*1	*1		
F			/	?	O	_	o				*1	*1	*1	*1		

*1: Japanese character.

Appendix D. Intermediate Results

A temporary operation result is produced while the value of an arithmetic statement or arithmetic expression or a function value is determined. It is called "intermediate result." The attribute and precision of the intermediate result are determined according to the rules described in this appendix. These rules also apply to intermediate results of a data item, a function value, an intermediate result, and two intermediate results.

If the intermediate result value exceeds its precision at execution time, the intermediate result will be truncated. Therefore, if a longer data item is specified in the arithmetic statement or arithmetic expression operand or in the function argument, an unexpected truncation may occur during result evaluation.

Attribute and Precision of the Intermediate Result

There are two intermediate result attributes: fixed floating-point attribute and floating-point attribute.

An intermediate result having the fixed point attribute is handled as binary, packed decimal, or zoned decimal item. The precision of the fixed point attribute is represented using the total number of digits and the number of digits in the decimal-part.

The intermediate result having the floating-point attribute is handled as an internal floating-point data item. Precision of a floating-point attribute is either long precision or short precision. An intermediate result having the long precision is handled as a long precision floating-point data item. An intermediate result having the short precision is handled as a short precision floating-point data item.

The attribute and precision of intermediate results depend on the arithmetic operation and function types.

Intermediate Results of Arithmetic Operations

Intermediate results of arithmetic operations (addition, subtraction, multiplication, and division) have the following attributes:

- When both operands in an arithmetic operation are numeric data items, numeric literals, or intermediate results having the fixed point attribute, the intermediate result will have the fixed point attribute.
- When at least one operand of an arithmetic operation is a floating-point data item, floating-point literal, or intermediate result having the floating-point attribute, the intermediate result will have the floating-point attribute.

Precision of the Intermediate Results of Arithmetic Operations Having Fixed Point Attribute

Precision of an intermediate result of an arithmetic operation having the fixed point attribute is represented with the total number of digits and the number of digits in the decimal-part. The total number of digits of the intermediate result does not contain P.

Precision is determined using the procedures below.

1. The reference number of digits in the decimal-part is determined from the operand attributes in an arithmetic statement containing any arithmetic operation or in an arithmetic expression.
2. The temporary number of digits of an arithmetic operation is determined from the operands and number of digits in the arithmetic operations decimal-part.

3. The precision of the intermediate result of an arithmetic operation is determined by the temporary number of digits in an arithmetic operation and the reference number of digits in the decimal-part.

Determining the Reference Number of Digits in the Decimal-part

First, the reference number of digits in the decimal-part is determined. The "reference number of digits in the decimal-part" is the maximum number of digits in the decimal-part to assure an intermediate result.

For Arithmetic Statement or Arithmetic Expression

All operands in arithmetic statements and arithmetic expressions (including destination items) are used. Use the maximum number of digits in the decimal-part operand as the reference number of digits in the decimal-part. However, when using arithmetic statements and expression, any operands used as an exponent in an exponent or division divisor and as function arguments should be excluded. When ROUNDED is specified in the destination, increase the number of digits in the decimal-part of the destination item and use that result for the destination item. If no operand is used to determine the reference number of digits in the decimal-part, 0 is set as the reference number of digits in the decimal-part.

For Relation Condition or EVALUATE Statement

Use any operand in an optional body or selectable pair in a relation condition or EVALUATE statement. Of these operands, the maximum number of digits in the decimal-part is used as the reference number of digits in the decimal-part. However, in the relation condition or EVALUATE statement, any operands used as an exponent in the square or division divisor and as function arguments should be excluded. If no operand is used to determine the reference number of digits in the decimal-part, 0 is set as the reference number of digits in the decimal-part.

Determining the Temporary Number of Arithmetic Operation Digits

After determining the reference number of digits in the decimal-part , the temporary number of arithmetic operation digits is determined. The "temporary number of arithmetic operation digits" is the maximum number of digits of the arithmetic operation. This number is used to obtain the number of final digits of the intermediate result.

The temporary number of arithmetic operation digits is determined by using the actual values shown below. The temporary number of digits for subtraction can be determined similar to addition procedures.

When the Operand is a Data Item

Except for divisor, use the data item PICTURE symbol maximum value . For example, for "PIC S9(5)V99", use 99999.99.

For the divisor, use the positive values minimum value in the data item PICTURE symbol . For example, for "PIC S9(5)V99," use 0.01.

When the Operand is a Literal

The absolute value of the literal value is used.

When the Operand is an Intermediate Result

Operation result is determined together with the precision of the intermediate result. This result is used in the following:

- When the temporary number of arithmetic operation digits is 30 or less, and the temporary number is calculated.
- When the temporary number of arithmetic operation digits is over 30, and the maximum value having the precision of the intermediate result (30 digits 99 .... 99) is used.

Number of Digits in the Decimal-part in the Temporary Number of Arithmetic Operation Digits

The table below lists the number of digits in the decimal-part in the temporary number of arithmetic operation digits.

Arithmetic Operation	**Number of Digits in Decimal-part**
Addition (OP1 + OP2) Subtraction (OP1 - OP2)	MAX(OP1d, OP2d)
Multiplication (OP1 * OP2)	OP1d+OP2d
Division (OP1/OP2)	MAX(Bd, OP1d-OP2d)

OP1d and OP2d indicate the number of digits in the decimal-part of OP1 and OP2, respectively. Bd indicates the reference number of digits in the decimal-part.

Example of Determining the Temporary Number of Arithmetic Operation Digits (Example 1)

```
S PIC S9V9.
T PIC S9V99.
U PIC S9V999.
V PIC S9V9999.
X PIC S9999V9999.

COMPUTE X = S + 1.00 + U * V.
```

a. Precision of the intermediate result of S + 1.00 is determined as detailed below:

$$\begin{array}{llll} & S & \text{PIC} & 9V9 \\ +) & & & 1\,.\,00 \\ \hline & S\ +\ 1\,.\,00 & & 10\,.\,90 \quad \ldots \textcircled{1} \end{array}$$

$$\underbrace{10}_{\text{The number of digits in the integer-part is 2.}}\,.\,\underbrace{90}_{\text{The number of digits in the decimal-part is 2.}}$$

b. Precision of the intermediate result of U * V is determined as shown below:

$$\begin{array}{llll} & U & \text{PIC} & 9V999 \\ \times) & V & \text{PIC} & 9V9999 \\ \hline & U \times V & & 99\,.\,9890001 \quad \ldots \textcircled{2} \end{array}$$

$$\underbrace{99}_{\text{The number of digits in the integer-part is 2.}}\,.\,\underbrace{9890001}_{\text{The number of digits in the decimal-part is 7.}}$$

c. Precision of the intermediate result of S + 1.00 plus intermediate result of U * V is determined as detailed below:

$$\begin{array}{lll} & \textcircled{1} & 10\,.\,90 \\ +) & \textcircled{2} & 99\,.\,9890001 \\ \hline & \textcircled{1}\ +\ \textcircled{2} & 110\,.\,8890001 \end{array}$$

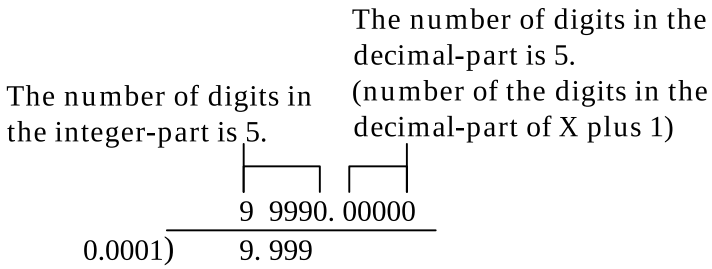
$$\underbrace{110}_{\text{The number of digits in the integer-part is 3.}}\,.\,\underbrace{8890001}_{\text{The number of digits in the decimal-part is 7.}}$$

Example of Determining the Temporary Number of Arithmetic Operation Digits (Example 2)

```
DIVIDE  U  BY  V  GIVING  X  ROUNDED .
```

Assume the same operand attribute as in example 1.

Precision of the intermediate result of the arithmetic operation U / V is determined as shown below:



The number of digits in the integer-part is 5.

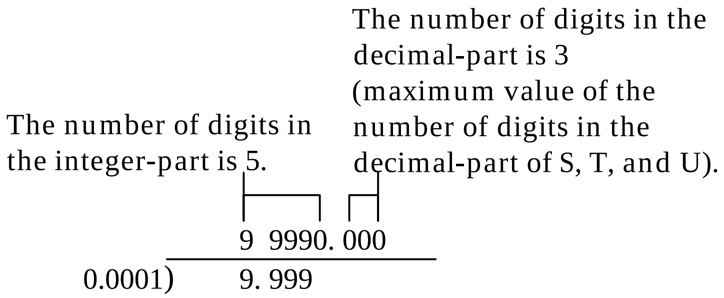
The number of digits in the decimal-part is 5. (number of the digits in the decimal-part of X plus 1)

Example of Determining the Temporary Number of Arithmetic Operation Digits (Example 3)

```
COMPUTE  S = S + T + ( S * U ) / ( U  /  V )
```

Assume the same operand attribute as in example 1.

Precision of the intermediate result of the arithmetic operation U / V is determined as shown below:



The number of digits in the decimal-part is 3 (maximum value of the number of digits in the decimal-part of S, T, and U).

The number of digits in the integer-part is 5.

Determining Precision of the Intermediate Result of Arithmetic Operations

After the temporary number of arithmetic operation digits is determined, precision of the arithmetic operation is determined by the following:

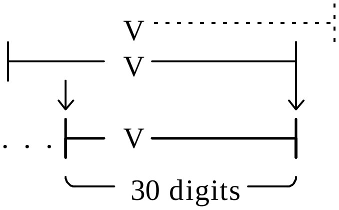
- When the temporary number of arithmetic operation digits is 30 or less, use the same value as precision of the intermediate result.
- When the temporary number of arithmetic operation digits is over 30, truncate the number of digits to 30 digits and use it as precision of the intermediate result.

Rules for truncation when the temporary number of arithmetic operation digits exceeds 30 digits are the following:

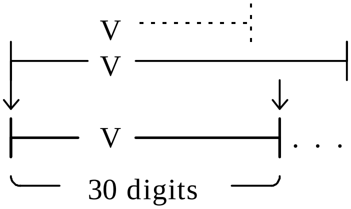
Symbols used throughout the description mean the following:

V	:	Decimal point location
V········⋮	:	Reference number of digits in the decimal part
├──── V ──────┤	:	Temporary number of all arithmetic operation digits (over 30 digits)
├──── V ───┤	:	All of digits of the intermediate result (30 digits)
. . .	:	Digits to be truncated

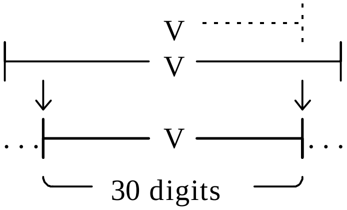
1. When the number of digits in the decimal-part is less than the reference number of digits in the decimal-part, then the number of digits in the decimal-part determined above is used, and the higher digits of the integer part is truncated.



2. When the number of digits in the decimal-part is greater than the reference number of digits in the decimal-part and the sum of the number of digits in the integer-part and the reference number of digits in the decimal-part is 30 digits or less, then the number of digits in the integer-part determined above is used, and the lower digits of the decimal-part is truncated.



3. When the number of digits in the decimal-part is greater than the reference number of digits in the decimal-part and the sum of the number of digits in the integer-part and the reference number of digits in the decimal-part exceeds 30 digits, then the reference number of digits in the decimal-part is used as the number of digits in the decimal-part, and the lower digits in the decimal-part and higher digits in the integer-part are truncated.



Intermediate Result in Arithmetic Operations Having the Floating-point Attribute

Precision of the intermediate result of arithmetic operations having the floating-point attribute is determined by the following:

- When at least one operand of an arithmetic operation is long precision, the intermediate result will have long precision.
- When both operands in an arithmetic operation are short precision, the intermediate result will have short precision.

Intermediate Result of the Exponent

The intermediate result of the exponent has the following attributes:

- When the exponent is an integer, the intermediate result of the exponent has the fixed point attribute.
- When the exponent is not an integer, the intermediate result of the exponent has the floating-point attribute.

Intermediate Result of the Exponent Having Fixed-point Attribute

Precision of the intermediate result of the exponent having the fixed point attribute depends on whether the exponent is a literal or variable.

When the Exponent is a Literal

When the exponent is a literal (integer), the exponent (A ** B) is expanded to the formulas shown below. At this point, the intermediate result of each multiplication (T1, T2, ..., Tn) and the intermediate result of division when B is a negative value, precision of the intermediate result of the multiplication and division follows.

[When B is a positive value]

$$\left.\begin{array}{llllll} \mathrm{A} & * & \mathrm{A} & \Rightarrow & \mathrm{T1} \\ \mathrm{T1} & * & \mathrm{A} & \Rightarrow & \mathrm{T2} \\ \mathrm{T2} & * & \mathrm{A} & \Rightarrow & \mathrm{T3} \\ & & : & & \\ \mathrm{Tm} & * & \mathrm{A} & \Rightarrow & \mathrm{Tn} \end{array}\right\} \begin{array}{l} \text{Multiplication is} \\ \text{repeated n times.} \\ \text{(n=B-1, m=n-1)} \end{array}$$

[When B is a negative value]

$$\left.\begin{array}{llllll} \mathrm{A} & * & \mathrm{A} & \Rightarrow & \mathrm{T1} \\ \mathrm{T1} & * & \mathrm{A} & \Rightarrow & \mathrm{T2} \\ \mathrm{T2} & * & \mathrm{A} & \Rightarrow & \mathrm{T3} \\ & & : & & \\ \mathrm{Tm} & * & \mathrm{A} & \Rightarrow & \mathrm{Tn} \end{array}\right\} \begin{array}{l} \text{Multiplication is repeated} \\ \text{n times.} \\ \text{(n=absolute value of B-1,} \\ \text{m=n-1)} \end{array}$$

$$1 \;/\; \mathrm{Tn}$$

When the Exponent is a Variable

When the exponent is a variable (integer), the number of all digits in the intermediate result of the exponent is 30 and the number of digits in the decimal-part is the reference number of digits in the decimal-part. The reference number of digits in the decimal-part used is the value described in Appendix D.

Intermediate Result of the Exponent Having Floating-point Attribute

Precision of the intermediate result of the exponent having the floating-point attribute is long precision.

Attributes and Accuracy of Function Values

The following table lists the attribute and precision of numeric and integer functions.

Function Name	Function Type	Attribute and Precision of Function Value
ACOS	Numeric	Long precision floating-point
ANNUITY	Numeric	Follow the intermediate result in the expression which determines the function
ASIN	Numeric	Long precision floating-point
ATAN	Numeric	Long precision floating-point
COS	Numeric	Long precision floating-point
DATE-OF-INTEGER	Integer	8-digit fixed point
DAY-OF-INTEGER	Integer	7-digit fixed point
FACTORIAL	Integer	Follow the intermediate result in the expression which determines the function
INTEGER	Integer	Fixed point with number of digits in the integer-part of argument plus 1
INTEGER-OF-DATE	Integer	8-digit fixed point
INTEGER-OF-DAY	Integer	7-digit fixed point
INTEGER-PART	Integer	Fixed point with number of digits in the integer-part of argument
LENG	Integer	10-digit fixed point

Function Name	Function Type	Attribute and Precision of Function Value
LENGTH	Integer	Fixed point with number of digits accommodating the length of argument
LOG	Numeric	Long precision floating-point
LOG10	Numeric	Long precision floating-point
MAX	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
MEAN	Numeric	Follow the intermediate result in the expression which determines the function
MEDIAN	Numeric	Follow the intermediate result in the expression which determines the function
MIDRANGE	Numeric	Follow the intermediate result in the expression which determines the function
MIN	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
MOD	Integer	Follow the intermediate result in the expression which determines the function
NUMVAL	Numeric	Fixed point with the number of digits of argument
NUMVAL-C	Numeric	Fixed point with the number of digits of argument
ORD	Integer	3-digit fixed point
ORD-MAX	Integer	9-digit fixed point
ORD-MIN	Integer	9-digit fixed point
PRESENT-VALUE	Numeric	Long precision floating-point
RANDOM	Numeric	Follow the intermediate result in the expression which determines the function
RANGE	Depending on the type of argument	Follow the intermediate result in the expression which determines the function

Function Name	Function Type	Attribute and Precision of Function Value
REM	Numeric	Follow the intermediate result in the expression which determines the function
SIN	Numeric	Long precision floating-point
SQRT	Numeric	Long precision floating-point
STANDARD-DEVIATION	Numeric	Long precision floating-point
SUM	Depending on the type of argument	Follow the intermediate result in the expression which determines the function
TAN	Numeric	Long precision floating-point
VARIANCE	Numeric	Follow the intermediate result in the expression which determines the function

Appendix E. Functional Differences

This appendix specifies the functional differences for each system.

Win

Handling of Numeric Values

Windows handles COMP-5 differently from other systems. For further details see the section titled “USAGE clause”.

Presentation Files

win

Destination

TRM and APL cannot be specified as the destination. For further details see the section titled “SYMBOLIC DESTINATION clause (presentation file)” for details.

Combining Specifiable Clauses and Destinations

Allowable combinations of destinations and clauses depends on the system. For further details, see the presentation file module table and file-control entry in the section titled “File-control paragraph”.

DS

USE FOR DEAD-LOCK Statement

The USE FOR DEAD-LOCK statement can be specified.

Win

Inter-Program Communication

Declare the WITH phrase in the procedure division header PROCEDURE DIVISION, CALL or ENTRY statement to specify the linkage type.

See the section titled “Procedure Division Header”, the section titled “CALL statement (inter-program communication)”, and the section titled “ENTRY statement (inter-program communication)”, individually.

Statements

HP

WRITE Statement (Sequential File)

The ADVANCING phrase can be specified in the WRITE statement for a print file or line sequence file. However, the ADVANCING phrase can be specified only in the WRITE statement for a print file in other systems.

For further details , see the section titled “WRITE statement (sequential file)” .

Win]

Database

The database function can be specified.

[Win16]

Communication Database

Linkage using PowerAIM as a communication database function can be specified.

[Win16]

Quantitative System Limits

Part of the quantitative system limits in the nucleus data sections differ from other systems. For details, see the section titled "Nucleus Data Section" in Appendix B, "Quantitative System Limits".

Appendix F. Control Record Formats

This appendix explains the I control record and the S control record. Certain fields are functionally meaningless. For details, see “COBOL85 User's Guide” for individual system.

I Control Record

The figures below show the formats of the I control record.

[Format-1]

REC-ID	M	FOVL	R	C	FCB	FORMAT-ID	CMOD	S	T	FORM	RSV
(2)	(1)	(4)	(3)	(3)	(4)	(8)	(4)	(3)	(1)	(4)	(24)

Remarks: The number enclosed in parentheses in each field indicates the number of characters(bytes).

[Format-2]

REC-ID	M	FOVL	R	C	FCB	FORMAT-ID	CMOD	S	T	FORM
(2)	(1)	(4)	(3)	(3)	(4)	(8)	(4)	(3)	(1)	(4)

XTB1	XTB2	XTB3	XTB4	LOAD	OFF-STK	PRT-FORM	SIZE	HOPPER	STACKER	SIDE	POSIT
(4)	(4)	(4)	(4)	(1)	(1)	(2)	(3)	(2)	(2)	(1)	(1)

PRT -AREA	BIND Portrait Front	BIND Portrait Back	BIND Landscape Front	BIND Landscape Back	WIDTH
(1)	(1)	(1)	(1)	(1)	(4)

OFFSET	OFFSET	OFFSET	OFFSET	RSV
Portrait	Portrait	Landscape	Landscape	
X	Y	X	Y	
(4)	(4)	(4)	(4)	(9)

Remarks: The number enclosed in parentheses in each field indicates the number of characters(bytes).

The data attributes and meanings in I control record fields are as follows:

REC-ID: control record ID; PICTURE X (2)

The control record identifier must specify I1 for an I control record.

M: format; PICTURE X

A 0 indicates an I control record in Format-1.

A 1 indicates an I control record in Format-2.

FOVL: form overlay module name; PICTURE X (4)

Specifies the form module name overlay to use. If this field is blank, the file open specification takes effect.

The file open specification is a system print environment when opened.

R: form overlay print count; PICTURE 9 (3)

Specifies the number of page copies subject to form overlay printing specified by FOVL. The R range is form 0 to 255. If FOVL specifies 0, the print count takes the same value as C (number of copies), and prints all copies. If the FOVL field is left blank, this field is meaningless.

C: number of copies; PICTURE 9 (3)

Specifies the number of copies of each page. The C range is from 0 to 255. If 0 is specified, the file open specification takes effect.

FCB/LPCI: FCB name / LPCI name; PICTURE X (4)

Specifies the FCB/LPCI name applied to the page. If the field is blank, the file open specification takes effect. A program cannot specify an FCB/LPCI name concurrently as a format definition.

FORMAT-ID: format definition name; PICTURE X (8)

Specifies the format definition name applied to the page, making the page a fixed-format page. If the field is blank, a non-fixed-format page results. A program cannot specify a format definition name and an FCB/LPCI name concurrently.

CMOD: copy modification module name; PICTURE (4)

Specifies the copy modification module name applied to the page. If the CMOD field is blank, the file OPEN specification takes effect.

S: copy modification start number; PICTURE 9 (3)

Specifies the segment number in the copy modification module to start copied data modification. The range is from 1 to 255. If the CMOD field is left blank, the S field is meaningless.

T: copy modification character array table number; PICTURE 9

Specifies the character array table number to be used. Use numbers 0 to 3, corresponding XTB1 to 4. If CMOD is blank, the T field is meaningless. The file open specification takes effect.

FORM: form identifier name; PICTURE X(4)

If the printer requires forms changes, you can specify the form identifier name with any name desired. If the field is blank, the program defaults to the file open forms already in use.

XTB1 through XTB4: character array table or additional character set; PICTURE X (4)

Specifies a character array table or additional character set identifier name. If this field is blank, the file open specification takes effect. Character array table specification for a printer file using a specified FORMAT clause is not available.

LOAD: dynamic load; PICTURE X

Specifies dynamic load is performed if unprintable characters are detected during printout. If field D is assigned, the program downloads the characters dynamically. If the field is blank, the file open specification takes effect.

OFF-STK: offset stack; PICTURE X

Specifies offset stacking. An O specifies offset stacking. A blank specifies no offset stacking. Offset stacking refers to the physical offsetting of the printed output, in whatever units are desired, on output from the stacker. When the back surface is the printing surface, offset stacking must not be specified for two-sided printing.

PRT-FORM: print format; PICTURE X (2)

Specifies the print format. The formats are: P(portrait mode), L(landscape mode), PZ(condensed print portrait mode), LZ(condensed print landscape mode), and LP(line printer mode). If the field is blank, SIZE must also be blank, and the file open specification takes effect.

SIZE: paper size; PICTURE X (3)

Specifies the paper size. The following sizes can be specified: A3,A4,A5, B4,B5,and LTR. If this field is blank, PRT-FORM must also be blank, and the file open specification takes effect. For condensed printing, paper size A3 must not be specified. For line printer mode, paper size A4 must be specified. The table below shows the allowable SIZE and PRT-FORM combinations.

PRT-FORM	SIZE					
	A3	A4	A5	B4	B5	LTR
L	o	o	o	o	o	o
P	o	o	o	o	o	o
LZ	x	o	o	o	o	o
PZ	x	o	o	o	o	o
LP	x	o	x	x	x	o

o : Legal combination
x : Illegal combination

HOPPER: paper supply hopper; PICTURE X(2)

Specifies the paper supply to use. The following values can be specified: P1 (primary supply hopper 1), P9 (primary supply hopper 2), 10 (primary supply hopper 10), 99 (primary supply hopper 99), or S (secondary supply hopper). If P is specified, any hopper may supply paper. If the field is blank, the file open specification takes effect. When the HOPPER field is specified, PRT-FORM and SIZE also must be specified.

STACKER: paper ejection port; PICTURE X(2)

Specifies the paper ejection port for printer output. The ports are: P1 (primary ejection port 1), P2 (primary ejection port 2), or S (secondary ejection port). If P is specified, any port can eject paper. If the STACKER field is blank, the file open specification takes effect.

SIDE: print surface specification; PICTURE X

Specifies the surface to be printed. An F specifies single-sided printing. A B specifies two-sided printing. If the SIDE field is blank, the file open specification takes effect.

POSIT: print position; PICTURE X

Specifies whether to position the printing surface on the front surface or back surface for two-sided printing. An F specifies printing on the front surface. A B specifies printing on the back surface. If the POSIT field is blank, the file open specification takes effect. If SIDE is blank, the POSIT field also must be blank. If the SIDE field specifies single-sided printing, the printing must not be on the back surface.

PRT-AREA: print prohibit area; PICTURE X

Specifies prohibited printing area. An L allows printing. An N prohibits printing. If the PRT-AREA field is blank, the file open specification takes effect.

BIND: binding direction; PICTURE X

Specifies the desired direction to bind continuous multipage output. The binding format consists of four characters which specify a print format and printed sides combination. The four available fields are, in sequence, portrait mode front, portrait mode back, landscape mode front, and landscape mode back. The binding directions and characters specified for binding are: L (bind left), R (bind right), U (bind up), D (bind down). The binding direction is only for multipage output. Therefore, the binding direction for the preceding page remains in effect if the BIND field is blank. The possible binding combinations are the following:

LLUU, LLUD, LLDU, LLDD,
LRUU, LRUD, LRDU, LRDD,
RLUU, RLUD, RLDU, RLDD,
RRUU, RRUD, RRDU, RRDD,
UULL, UULR, UURL, UURR,
UDLL, UDLR, UDRL, UDRR,
DULL, DULR, DURL, DURR,
DDLL, DDLR, DDRL, DDRR

WIDTH: binding width; PICTURE 9(4)

Specifies the binding width. The range is 0 to 9999, in units of 1/1440inch. If 9999 is specified, 9999 must be specified in all the OFFSET fields. If the WIDTH field is blank, all the OFFSET fields must be blank. The file open specification then starts.

OFFSET: printing origin position; PICTURE 9(4)

Specifies the printing origin position. The range is 0 to 9999 in units of 1/1440 inch. If the printing format is only portrait or landscape mode, blanks are specified in the remaining printing format fields. If 9999 is specified in all the OFFSET fields, the WIDTH field must also specify 9999. If all the OFFSET fields are blank, the WIDTH field must also be blank. The file open specification then takes effect.

RSV: used by the system; PICTURE X(24) or X(9)

Must be blank.

S Control Records

The format for the S control record is shown below.

REC -ID	M	RSV	N	FOVL-1	FOVL-2	....	FOVL-n
(2)	(1)	(3)	(3)	(4)	(4)		(4)

Remarks: The number enclosed in parentheses in each field indicates the number of character(bytes).

The data attributes and the S control record field meanings are as follows:

REC-ID: control record ID; PICTURE X(2)

Specifies S1 to indicate an S control record.

M: used by the system; PICTURE X

Must be 0.

RSV: used by the system; PICTURE X(3)

Must be blank.

N: overlay sequence count; PICTURE 9(3)

Specifies the number of form overlay module names FOVL-n.

FOVL-n: the name of a module in the form overlay sequence group; PICTURE X(4)

Specifies the number of form overlay module names specified in n. Form overlay printing is performed in the assigned order for the number of copies specified in the I control record C field.

Index