digital

VAX-11 COBOL-74

Language

Reference Manual

Order No. AA-C985A-TE

VAX11

January 1979

This document is intended primarily for reference use. It describes the VAX-11 COBOL-74 language.

VAX-11 COBOL-74
Language
Reference Manual

Order No. AA-C985A-TE

OPERATING SYSTEM AND VERSION: VAX/VMS V01.5

SOFTWARE VERSION: VAX-11 COBOL-74 V04

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license, and may only be used or copied it in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-paid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	FOCAL
DECnet	IAS
DECsystem-10	MASSBUS
DECSYSTEM-20	PDP
DECtape	RSX
DECUS	UNIBUS
DIBOL	VAX
DIGITAL	VMS

Contents

Chapter 3 Environment Division

Chapter 4 Data Division

Chapter 5 Procedure Division

Figures

Tables

Commercial Engineering Publications typeset this manual using DIGITAL's TMS-11 System.

955ALL

Preface

This reference manual describes the COBOL language as implemented in *VAX-11 COBOL-74* for the VAX-11 system. It adheres to the 1974 ANSI standard. Furthermore, the text of this manual is based on *American National Standard Programming Language COBOL*, ANSI Document X3.23-1974.

You should have a working knowledge of the COBOL language before using this book, which is a reference document; it is not a tutorial guide for beginning COBOL programmers.

Chapter 1 contains an overview of the COBOL language. Chapters 2 through 5 detail the four COBOL divisions. A discussion of the Library module appears in Chapter 6. Appendixes A, B, and C contain the COBOL reserved word list, character set tables, and FILE STATUS codes.

Frequent references to the *VAX-11 COBOL-74 User's Guide* (User's Guide) appear in the text. The User's Guide and the *VAX-11 SORT User's Guide* contain additional information about the compiler, the runtime system, error messages, and utility programs.

Acknowledgments

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

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

The authors and copyright holders of the copyrighted material used herein are: FLOW-MATIC (trademark of Sperry Rand Corporation), programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

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

Chapter 1
Overview of the COBOL Language

This chapter contains general information about the language and structure of COBOL source programs. It describes the elements of COBOL and its meta-language, reference formats, and language organization. A sample format entry concludes the chapter and introduces the source language statements discussed in later chapters.

1.1 COBOL Language Elements

The COBOL language consists of the following components:

- Divisions
- Sections
- Paragraphs
- Sentences
- Clauses
- Statements
- Entries
- Words
- Characters

There are four divisions in COBOL programs: the Identification Division, the Environment Division, the Data Division, and the Procedure Division. Each division can contain sections, which in turn can contain paragraphs. Each paragraph can contain one or more sentences, clauses, statements, or entries.

The basic building blocks of these COBOL components include the COBOL character set, character-strings, COBOL words, separators/punctuation and literals.

1.1.1 COBOL Character Set

The basic and indivisible unit of the COBOL language is the character. Individual characters combine to form character-strings of one or more contiguous characters, and separators, which are punctuation character-strings. The character set for character-strings and separators includes the letters A through Z, digits, and special characters. The complete COBOL character set appears in Appendix B.

For nonnumeric literals, comment entries, and comment lines, the character set is expanded to include the entire computer character set except for some special characters (such as the carriage return) that control I/O devices. The computer character set and its subsets appear in Appendix B.

NOTE:

If special characters, other than commas and semicolons, appear in general formats, you must use them in your source program as well.

1.1.2 COBOL Words

A COBOL word is a character-string of not more than 30 ASCII characters. There are two classes of words: user-defined words and reserved words. A COBOL word can belong to one and only one of these classes.

1.1.2.1 User-Defined Words — COBOL words that you must supply to satisfy the format of a clause or statement. User-defined words consist of characters selected from the set A through Z, the digits 0 through 9, and the hyphen (-). A hyphen can neither begin nor end a user-defined word.

There are 12 types of user-defined words:

condition-name	paragraph-name
data-name	program-name
file-name	record-name
index-name	section-name
level-number	segment-number
mnemonic-name	text-name

Each of these types is defined in the glossary.

1.1.2.2 Reserved Words — A specific list of COBOL words that you can use only as specified in the general formats. Do not use a reserved word as a user-defined word. (See Appendix A for a complete list of COBOL reserved words.)

There are six types of reserved words:

1. *Key words* - Words that you must use in a particular format. Key words are upper case and underlined in general formats. Consider the following example.

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...
=   arithmetic-expression [; ON SIZE ERROR imperative-statement]
```

In this case, COMPUTE, ROUNDED, SIZE, and ERROR are key words.

2. *Optional Words* - Words you can use or omit without altering the semantics of the COBOL program. Optional words are upper case, but not underlined, in general formats. In the previous example, the word ON is an optional word.
3. *Connectives* - There are three types of connectives:
   a. *Qualifier connectives* - associate a data-name, a condition-name, or a text-name with its qualifiers: OF, IN. (See Section 5.4.8.1, Qualification.)
   b. *Series connectives* - link two or more consecutive operands: separator comma or separator semicolon.
   c. *Logical connectives* - express the following four conditions: AND, OR, AND NOT, OR NOT.
4. *Special Registers* - Compiler-generated storage areas, such as LINAGE-COUNTER, that are named and referred to by reserved words. (See Section 4.1.5, LINAGE Clause.)
5. *Figurative Constants* - Words that name and refer to specific constant values generated by the compiler. The singular and plural forms of figurative constants are equivalent, and you can use them interchangeably. Do not put quotation marks around figurative constants.

   Reserved words and their figurative constant values follow:

ZERO ZEROS ZEROES	Represents the value '0', or one or more of the character '0', depending on context.
SPACE SPACES	Represents one or more of the character space from the computer character set.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest ordinal position in the computer character set (hex 7F).
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest ordinal position in the computer character set (hex 00).
QUOTE QUOTES	Represents one or more of the character '"'.
ALL literal	Represents one or more repetitions of the string of characters comprising the literal. The literal must be either an alphanumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and serves only to enhance readability.

   When a figurative constant represents a string of one or more characters, the compiler determines the string's length from context according to the following rules:
   a. When a figurative constant is associated with another data item (for example, when the figurative constant is moved to or compared with another data item), the string of characters that the figurative constant represents is repeated character by character to the right (or truncated on the right in the case of ALL literal) until the size of the resultant string equals the size of the associated data item. This is done prior to and independent of the application of any JUSTIFIED clause specified for the data item. (See Section 4.3.3, JUSTIFIED Clause.)

b. When a figurative constant is not associated with another data item (for example, when the figurative constant appears in a DISPLAY, STRING, UNSTRING or STOP statement), the length of the string is one character.

You can use a figurative constant wherever a literal appears in a format. Whenever the literal is restricted to numeric characters, however, use only the ZERO (ZEROS, ZEROES) figurative constant.

6. *Special-Character Words* - The arithmetic operators + (addition), - (subtraction), * (multiplication), / (division), ** (exponentiation), and relation characters < (less than), > (greater than), and = (equal to). You must use these words where they appear in general formats even though they are not underlined.

1.1.3 Literals

A literal is a character-string whose value is determined by the ordered set of characters of which it is composed. There are two types of literals: numeric and alphanumeric (alphanumeric is sometimes referred to as "nonnumeric").

NOTE:

A figurative constant can also serve as a literal.

1.1.3.1 Numeric Literal — A character-string of 1 to 20 characters selected from the digits 0 through 9, the plus sign, the minus sign, and the decimal point.

The value of a numeric literal is the algebraic quantity represented by the characters in the literal. The size of the literal equals the number of digits specified, including leading zeros, if any. Every numeric literal is category numeric. (See Section 4.3.6, PICTURE Clause.)

The rules for forming numeric literals are:

1. A numeric literal must contain at least one digit and not more than 18 digits.
2. A numeric literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal has no sign, its value is positive.
3. A numeric literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point and can appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is an integer. (The word "integer" appearing in a general format represents a non-zero, positive numeric literal with no decimal point.)
4. The compiler treats a numeric literal enclosed in quotation marks as an alphanumeric literal.

1.1.3.2 Alphanumeric Literal — A character-string of 1 to 127 allowable characters from the computer character set. It is delimited on both ends by quotation marks.

The value of an alphanumeric literal in the object program is the value of the character-string itself, except that: (1) the delimiting quotation marks are excluded, and (2) each embedded pair of contiguous quotation marks represents a single quotation mark character; all other punctuation characters are part of the value of the alphanumeric literal and are not separators. Alphanumeric literals are category alphanumeric. (See Section 4.3.6, PICTURE Clause.)

The rules for forming alphanumeric literals are:

1. An alphanumeric literal must contain a space or left parenthesis immediately before the opening quotation mark.
2. An alphanumeric literal must contain a separator (space, comma, semicolon, or right parenthesis) or terminator (period) immediately after the closing quotation mark.
3. To represent a single quotation mark character within an alphanumeric literal, use two contiguous quotation marks.

1.1.4 Separators

A separator is a string of one or more of the punctuation characters described in this section. The rules for forming separators follow:

1.1.4.1 Space

1. Where a space is used as a separator, more than one space can be used.
2. A space can immediately precede any separator except the closing quotation mark. Before a closing quotation mark, the space is considered part of an alphanumeric literal rather than a separator.

NOTE:

Section 1.3, Source Reference Formats, describes the only exception to the first two rules.

3. A space can immediately follow any separator except the opening quotation mark. After an opening quotation mark, the space is considered part of an alphanumeric literal rather than a separator.

1.1.4.2 Comma and Semicolon — The comma and semicolon function as separators only when they are immediately followed by a space. Insert these separators only where explicitly permitted by the general formats, by format punctuation rules, by statement and sentence structure definitions, or by reference format rules.

1.1.4.3 Left and Right Parentheses — Left and right parentheses are separators only when used in balanced pairs.

1.1.4.4 Quotation Marks — Quotation marks used in balanced pairs delimit alphanumeric literals. (See Section 1.1.3.2, Alphanumeric Literals.)

1.1.4.5 Horizontal Tab — The horizontal tab character vertically aligns statements or clauses on successive lines of the source program listing. It adheres to the same rules that govern the space character. The compiler, upon encountering a tab character, generates one or more space characters consistent with the tab character position in the source line. (See Sections 1.3, Source Reference Formats.)

1.1.5 Format Punctuation

The comma, semicolon, and period appear in some formats. The comma and semicolon are optional and interchangeable. The period, however, is mandatory: Supply a period wherever one is shown in a general format. You must also specify a period to terminate a paragraph.

1.2 Meta-Language Elements

Meta-language elements describe the allowable use of language elements. They appear in formats but are not coded into source language statements.

1.2.1 Underline

Underlined, upper-case words denote reserved key words. The absence of an underline in an upper case word denotes an optional word.

1.2.2 Brackets and Braces

Brackets, [ ], enclose an optional portion of a general format. When they enclose vertically stacked entries, brackets indicate that you can, at your option, select one of the enclosed entries. Braces, { }, surrounding vertically stacked entries indicate that you must choose one of the enclosed entries.

In the following example, brackets indicate that the entire clause is optional. If you use the clause, you must select either SYNCHRONIZED or SYNC. You can select either LEFT or RIGHT (or neither).

```
[{SYNCHRONIZED}  [LEFT ]]
 {SYNC        }  [RIGHT]
```

NOTE:

In the general format for a clause, choices that are vertically stacked between brackets indicate that you have the option of overriding a default condition. The default condition is always described in the general rules for the clause.

1.2.3 The Ellipsis

The ellipsis (...) indicates that you can repeat the item preceding it. This item is usually enclosed in brackets or braces. Consider the following example.

```
[SAME [RECORD] AREA FOR file-name-1 {file-name-2} ...]...
```

The ellipsis following the outside brackets indicates that you can repeat the entire clause. The other ellipsis allows you to repeat the item in braces.

1.3 Source Reference Format

The compiler provides two formats for coding your source programs: conventional and terminal. The former is based on the traditional, 80-column punched card format. The latter is a DEC-specified format that shortens a source line by using horizontal tabs and carriage returns; the terminal format works well when you use a text editor from an on-line terminal.

NOTE:

> The compiler assumes terminal format as a default, but you can use either format. (The User's Guide discusses format selection.)

Use the reformatting program (REFORMAT) to change a terminal format program to conventional format for ease in transporting the source program to other COBOL compilers. (The User's Guide discusses the REFORMAT utility.)

NOTE:

> The rules for spacing presented in this discussion of reference formats take precedence over all other spacing rules.

1.3.1 Conventional Reference Format

The conventional reference format provides rules for coding your source program on 80-column punched cards. These rules are described in the following sections.

1.3.1.1 Reference Format Areas

1. *Sequence Number Area* - Character positions 1 through 6. Reserved for source line sequence numbers that enable you to locate and edit source lines in your program. The compiler ignores the contents of this field.
2. *Continuation/Comment Indicator Area* - Character position 7. Contains a character that directs the compiler to process the source line in one of the following ways:

Character	Source line processed as
blank ( )	Default - The compiler processes the line as normal COBOL text.
hyphen (-)	Continuation line - The compiler processes the line as a continuation of the previous source line. (See Section 1.3.1.2, Continuation of Lines.)
asterisk (*)	Comment line - The compiler transfers the contents of this line, as is, to the source listing and does not check syntax. (See Section 1.3.1.4, Comment Lines.)
slash (/)	Comment line - The compiler treats the line as if it were a comment line, except that it advances the source listing to the top of the next page before printing the line.

3. *Area A* - Character positions 8 through 11. Contains division headers, section headers, paragraph headers, paragraph-names, level-indicators, and certain level numbers.
4. *Area B* - Character positions 12 through 72. Contains all other COBOL text.
5. *Identification Field* - Character positions 73 through 80. Contains source program documentation that has no effect on compilation.

1.3.1.2 Continuation of Lines

1. Divide a multi-line sentence or entry by continuing in Area B of the next line.
2. Break a word or numeric literal from one line to the next by placing a hyphen (-) in character position 7 of the continuation line; the first non-blank character that you enter in Area B will become the next character of the continued word or numeric literal.
3. Break an alphanumeric literal from one line to the next by placing a hyphen in character position 7 of the continuation line. Put a quotation mark before the first character of the continuation literal. The literal can begin anywhere in Area B of the continuation line.

Consider the following example:

```
001010 01  CONTINUATION-NUMERIC.
001020       02  NUMERIC-LITERAL          PIC 9(18) VALUE IS 12345678912345
001030-      6789.
001040 01  CONTINUATION-ALPHANUMERIC.
001050       02  ALPHANUMERIC-LITERAL     PIC X(26) VALUE IS "ABCDEFGHIJKLM
001060-     "NOPQRSTUVWXYZ".
001070 PROCEDURE DIVISION.
001080 CONTINUATION-SENTENCE.
001090      IF NUMERIC-LITERAL NOT EQUAL TO ALPHANUMERIC-LITERAL
001100                  GO TO END-PROGRAM
001110             ELSE GO TO CONTINUATION-SENTENCE.
001120 END-PROGRAM.
001130           STOP RUN.
```

Source lines 001010 through 001030 show how to continue a numeric literal; lines 001040 through 001060 show line continuation for an alphanumeric literal. Finally, source lines 001090 through 001110 contain a sentence that continues for three lines.

1.3.1.3 Blank Lines — Include blank lines (character positions 7 through 72 blank) anywhere in a source program except immediately before a continuation line.

1.3.1.4 Comment Lines — Include comment lines (an asterisk in character position 7) anywhere in a source program except before the Identification Division. Successive comment lines must also contain asterisks in character position 7.

You can use any character from the computer character set to write a comment line. Begin your comments in Area A or Area B. The compiler reproduces comment lines on the source listing for documentation purposes.

NOTE:

The slash character (/) and asterisk (*) produce the same results, except that the slash directs the compiler to advance the source listing to the top of the next page before printing the comment entry.

1.3.1.5 Short Lines and Tab Characters — If you use a medium other than punched cards, you can shorten conventional format source lines: Either terminate the line with a carriage return, insert tab characters within the line to replace space characters, or use a combination of the two.

The compiler treats a carriage return character as a redefinition of character position 72. When you use a tab character, the compiler generates the required number of space characters consistent with the tab character position on the line. Tab stops are set in the compiler at character positions 7, 8, 12, 20, 28, 36, 44, 52, 60, 68, and 73.

Consider the following example, in which RET stands for the carriage return character and TAB stands for the tab character.

Shortened conventional source line

```
000130 01  TAB  FILE-A.  RET
000140  TAB  02  DATA-FIELD-A.  RET
000150  TAB  TAB  03 DESCRIPTION-A  TAB  PIC X(20).  RET
000160  TAB  TAB  03 DESCRIPTION-B  TAB  PIC X(20).  RET
000170  TAB  TAB  03 DESCRIPTION-C  TAB  PIC X(20).  RET
```

Source line as interpreted by the compiler

```
000130 01  FILE-A.
000140     02  DATA-FIELD-A.
000150             03 DESCRIPTION-A        PIC X(20).
000160             03 DESCRIPTION-B        PIC X(20).
000170             03 DESCRIPTION-C        PIC X(20).
```

1.3.2 Terminal Reference Format

Terminal reference format is the compiler's default format. It is easy to use with a computer terminal and is less time and space consuming than its conventional counterpart. This format eliminates the sequence number and identification fields and combines the indicator field with Area A.

The terminal reference format for a source line follows:

Character Position	Contents
1 through 4	Area A
5 through 65	Area B

NOTE:

Place continuation line (-), comment line (*), and skip-to-top-of-page (/) indicator characters in character position 1.

In terminal format, Area A and Area B contain the same kinds of source entries as their conventional format counterparts. (See Section 1.3.1.1) Similarly, tab characters cause the compiler to generate a number of spaces consistent with the tab character position on the line. Tab stops are set to character positions 5, 13, 21, 29, 37, 45, 53, 61, and 66.

1.4 Language Organization

Each division, section, and paragraph in a COBOL program contains headers followed by source text. The following sections describe both the organization of these headers and their reference format positions.

1.4.1 Division Header

A division header indicates the beginning of a division. It is a specific combination of words followed by a period. Division headers, in their order of appearance, are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

A division header must start in Area A. No non-comment text can appear between it and the following section header, paragraph header, or paragraph-name, except for the key word DECLARATIVES (followed by a period and a space), which can appear after the Procedure Division header.

1.4.2 Section Header

A section header indicates the beginning of a section in the Environment, Data, and Procedure Divisions. In the Environment and Data Divisions, a section header contains reserved words followed by the word SECTION (followed by a period and a space). In the Procedure Division, a section header contains a user-defined word followed by the word SECTION (and an optional segment-number) followed by a period and a space. The permissible section headers are:

In the Environment Division

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

In the Data Division

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

In the Procedure Division

```
user-name SECTION [segment-number].
```

The section header must start in Area A. No text can appear between it and the following paragraph header or paragraph-name except for the USE sentence in the Procedure Division.

1.4.3 Paragraph, Paragraph Header, Paragraph-Name

Paragraphs begin with paragraph headers (reserved words) or paragraph-names (user-defined words), depending on the division. In the Identification and Environment Divisions, a paragraph consists of a paragraph header (followed by a period) and zero, one, or more entries. In the Procedure Division, a paragraph consists of a paragraph-name (followed by a period) and zero, one, or more entries. Data Division entries follow a different format. (See Section 1.4.4, Data Division Entries.)

The permissible paragraph headers are:

In the Identification Division

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

In the Environment Division

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

A paragraph header or paragraph-name starts in Area A. The first sentence of a paragraph begins either on the same line or in Area B of the next non-blank line that is not a comment line. Successive sentences or entries begin either on the same line as the previous one or in Area B of the next non-blank line that is not a comment line. (See Section 1.3.1.2, Continuation of Lines.)

1.4.4 Data Division Entries

There are two types of Data Division entries: those that begin with a level-number (called "data-description-entries") and those that begin with a level indicator. The only level indicator is FD (File Description).

Following every level indicator or level-number are (in order): a space, its associated name, and a sequence of independent descriptive clauses. Each clause except the last ends with a separator semicolon or a separator space; the last clause ends with a period followed by a space.

Choose level-numbers from the set of values 1 through 49, 66, 77, and 88. Write the level-numbers 1 through 9 either as a single digit or as a zero followed by a significant digit. For level-numbers 01, 66, or 77, the entry begins in Area A with the level-number followed by a space; the entry ends in Area B with its associated record-name and descriptive information.

The FD level indicator entry begins in Area A with the level indicator followed by at least one space; the entry continues in Area B with a file-name and descriptive information.

You can maintain the same format for successive data-description-entries, or you can indent according to level-number. When you indent, begin each new level-number anywhere in Area A or Area B, and end anywhere within Area B. Indentation does not affect the magnitude of the level-number. Note that your output listing will be indented only if the input is indented.

1.4.5 Declaratives

The key words DECLARATIVES and END DECLARATIVES precede and follow, respectively, the declaratives portion of the Procedure Division. Each must appear on a line by itself, starting in Area A and ending with a terminator period.

1.5 Sample Format Entry Page

The following page is a model of the entries that comprise the bulk of this manual. Each COBOL division begins a new chapter, and each entry begins on a new page.

Entry-Name

n.n.n Entry-Name

Function

Describes the function or effect of the entry.

General Format

A general format shows the specific arrangement of elements in the entry. Formats are numbered if you can use more than one specific arrangement. You must write all clauses (mandatory and optional) in the sequence shown in these general formats. Only in certain cases can clauses appear in sequences other than those shown; these exceptions are stated explicitly in the rules that follow the general format.

Syntax Rules

Syntax rules tell you how to order words or elements to form larger elements, such as sentences, clauses, or statements. They also impose restrictions on individual words or elements.

General Rules

General rules define or clarify the meaning (or relationship of meanings) of an element or a set of elements. They define the semantics of the entry and the entry's effect on program execution or compilation.

Examples

(If required)

Division Name

Chapter 2
Identification Division

Function

The Identification Division marks the beginning of a COBOL program. It also identifies a program and its source listing.

General Format

```
        IDENTIFICATION DIVISION.

        PROGRAM-ID. program-name.

      * [AUTHOR. [comment-entry] ...]

      * [INSTALLATION. [comment-entry] ...]

      * [DATE-WRITTEN. [comment-entry] ...]

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

      * [SECURITY. [comment-entry] ...]
```

* These paragraphs are not described in individual entries; they follow the same format as the DATE-COMPILED paragraph and are for documentation only.

Syntax Rules

1. The Identification Division must be the first entry in a COBOL program.
2. The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.
3. The PROGRAM-ID paragraph must immediately follow the Identification Division header.

General Rules

1. The comment-entry can consist of any combination of characters from the computer character set.
2. Comment-entries can span several lines. However, do not continue a comment-entry by using a hyphen in the continuation indicator area.

2.1 PROGRAM-ID Paragraph

Function

The PROGRAM-ID paragraph identifies the program.

General Format

```
PROGRAM-ID. program-name.
```

Syntax Rule

The program-name must contain 1 to 15 characters from the set A through Z and 0 through 9. Do not use the hyphen.

General Rules

1. The PROGRAM-ID paragraph must be present in every program and must contain a program-name.
2. Program-name is a user-defined word that identifies a COBOL program.
3. The program-name identifies the object program entry point.
4. Program-names cannot exceed 15 characters in length.
5. The first eleven characters of the program-name must not duplicate the first eleven characters of the program-name in any other program in the linked image.

2.2 DATE-COMPILED Paragraph

Function

The DATE-COMPILED paragraph causes the compiler to display the compilation date on the source program listing in the Identification Division.

General Format

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

Syntax Rules

1. The comment-entry can consist of any combination of characters from the computer character set.
2. Comment-entries can span several lines. However, do not continue a comment-entry by using a hyphen in the continuation indicator area.

General Rules

1. During program compilation, the paragraph-name DATE-COMPILED causes the current date to be inserted on a subsequent line of the program listing. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

```
DATE-COMPILED. comment-entry.
    current-date
```

2. All listings produced during compilation contain the compilation date in the header line of each page regardless of the presence or absence of the DATE-COMPILED paragraph.

Chapter 3
Environment Division

Function

The Environment Division provides a standard method for describing the program's hardware environment. It enables you to specify both (1) the compiling and object computers, and (2) information about input-output control.

General Format

```
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
 [SOURCE-COMPUTER. source-computer-entry]
 [OBJECT-COMPUTER. object-computer-entry]
 [SPECIAL-NAMES. special-names-entry]]
[INPUT-OUTPUT SECTION.
 FILE-CONTROL. {file-control-entry} ...
 [I-O-CONTROL. input-output-control-entry]]
```

Syntax Rules

1. The Environment Division must follow the Identification Division in every COBOL program.
2. The Environment Division must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

SOURCE-COMPUTER

3.1 CONFIGURATION SECTION

The Configuration Section can consist of three paragraphs: SOURCE-COMPUTER, OBJECT-COMPUTER, AND SPECIAL-NAMES.

3.1.1 SOURCE-COMPUTER Paragraph

Function

The SOURCE-COMPUTER paragraph specifies the computer on which the source program is to be compiled.

General Format

```
SOURCE-COMPUTER.    VAX-11.
```

General Rules

This paragraph is for documentation purposes only.

3.1.2 OBJECT-COMPUTER Paragraph

Function

The OBJECT-COMPUTER paragraph specifies the computer on which the object program is to be executed.

General Format

```
OBJECT-COMPUTER.  VAX-11  [, MEMORY SIZE integer {WORDS     }]
                                                 {CHARACTERS}
                                                 {MODULES   }
    [, PROGRAM COLLATING SEQUENCE IS alphabet-name]
    [, SEGMENT-LIMIT IS segment-number].
```

General Rule

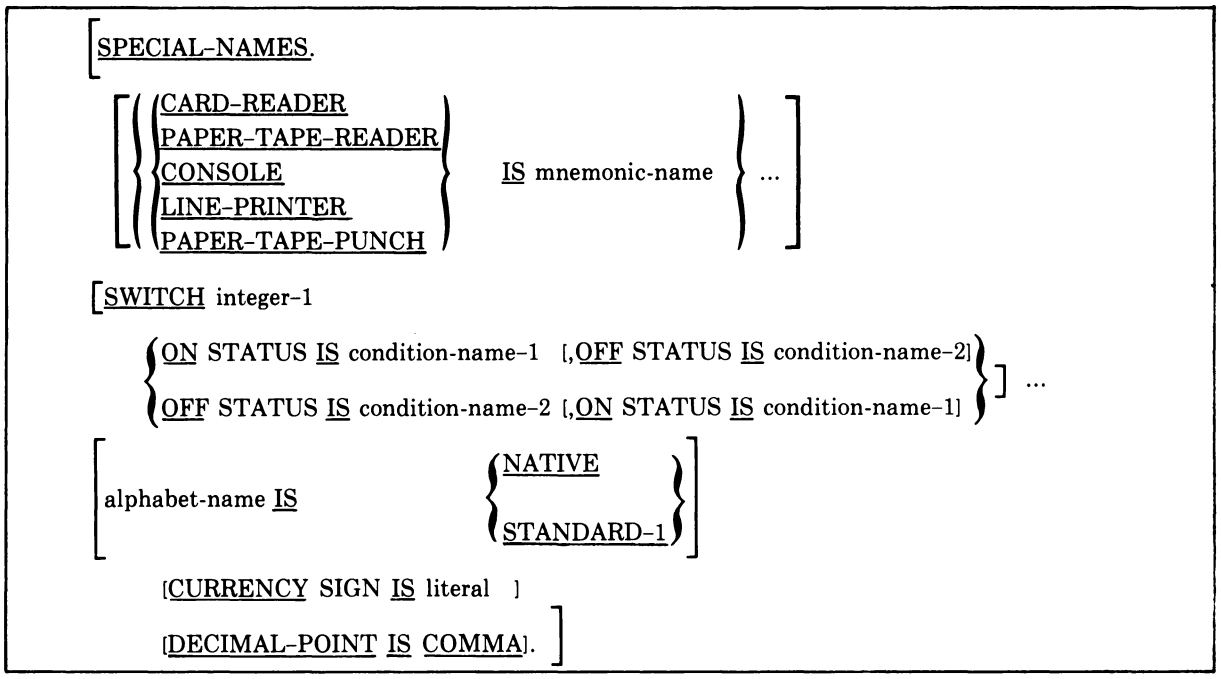
This paragraph is for documentation purposes only.

3.1.3 SPECIAL-NAMES Paragraph

Function

The SPECIAL-NAMES paragraph associates compiler features with user-specified mnemonic-names; it also associates alphabet-names (specified in the OBJECT-COMPUTER paragraph) with character sets and/or collating sequences.

General Format



Syntax Rules

1. You must use the SPECIAL-NAMES paragraph if your program includes mnemonic-names, condition-names, alphabet-names, the DECIMAL-POINT clause, or the CURRENCY SIGN clause.
2. Integer-1 represents any integer from 1 to 16.

General Rules

1. The names CARD-READER, PAPER-TAPE-READER, and CONSOLE refer to input devices. To transfer data from these devices, you can use the mnemonic-names assigned to them with the ACCEPT statement in the Procedure Division.
2. The names CONSOLE, LINE-PRINTER, and PAPER-TAPE-PUNCH refer to output devices. To transfer data to these devices, you can use the mnemonic-names assigned to them with the DISPLAY statement in the Procedure Division.

3. The name SWITCH refers to a logical switch to which the operator can assign a value at run-time. Chapter 2 of the User's Guide discusses the procedure for setting program switches.

4. The condition-name assigned to the ON or OFF STATUS of a switch can be used in a conditional expression. (See Section 5.6.8, Switch–Status Condition.)

5. The alphabet-name clause relates a name to a collating sequence and/or a character code set. An alphabet-name referenced in the PROGRAM COLLATING SEQUENCE clause specifies a collating sequence. An alphabet-name referenced in a CODE–SET clause in a file-description-entry specifies a character code set. (See Section 4.1.2, CODE–SET Clause.)

   a. If the STANDARD–1 phrase is specified, the character code set or collating sequence identified is that defined in the American National Standard Code for Information Interchange, X3.4–1968.

   b. Since the native character code set of your system is equivalent to the ASCII code, specification of the NATIVE phrase is equivalent to specification of the STANDARD–1 phrase.

6. The literal that appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. Use a single character for the literal. Do not use any of the following characters:

   a. Digits 0 through 9

   b. Alphabetic characters A,B,C,D,L,P,R,S,V,X,Z or the space

   c. Special characters *, +, –, , (comma), . (period), ; (semicolon), (,), ", /, or =

   If this clause is not present, you can only use the currency sign ($) in the PICTURE clause.

7. The DECIMAL–POINT IS COMMA clause exchanges the function of the comma and period in the PICTURE character-string and in numeric literals.

3.2 INPUT-OUTPUT SECTION

The Input-Output Section consists of two paragraphs that describe the information needed to control the transmission and handling of data between external media and the program. This section allows COBOL programs to access records stored in various file organizations.

The file organizations supported by the compiler, and the access methods available for processing them, are introduced below. Refer to the User's Guide for a more complete discussion of these topics.

File Organizations

The compiler supports three file organizations:

- Sequential
- Relative
- Indexed

Sequential files consist of records positioned one after the other in the order in which they were originally written. Each record (except the last) has another record following it. The location of a record is fixed in relation to the records that precede and succeed it. Sequential files can be processed only in a serial fashion. That is, to access a record in the middle of the file, the program must access all the records preceding it.

Relative files, restricted to disk storage devices, consist of successively numbered records. Each record is assigned a number relative to its position in the file. Thus, the first record in a file occupies the first position and receives a relative record number of 1, the second record occupies the second position and receives a relative record number of 2, and so on. An individual record in a relative file can be accessed directly (by specifying its relative record number) or serially, like sequential files.

Indexed files, like relative files, are restricted to disk storage devices. They consist of records and a primary key index (and optionally one or more alternate key indexes) used to process the records sequentially by key or randomly by key. A key is a data item in each record of the file.

Access Modes

File organization determines the access modes that can be used to retrieve and store records in the file. Though file organization is fixed when the file is created (and cannot be changed later), the access mode is not fixed (except for sequential files) until a program opens the file. Therefore, different programs can use different access methods for the same file.

The compiler supports three access modes:

- Sequential
- Random
- Dynamic

In the sequential access mode, the program accesses records serially. The first record must be accessed before the second, the second before the third, and so on.

In the random access mode, the program accesses records individually by a random record number or a data key.

Dynamic access allows you to choose at will between sequential or random access.

Table 3-1 lists the allowable combinations of file organizations and access modes.

Table 3-1: Access Modes and File Organizations

File Organization	Access Mode		
	Sequential	**Random**	**Dynamic**
Sequential	Yes	No	No
Relative	Yes	Yes	Yes
Indexed	Yes	Yes	Yes

The User's Guide further discusses the access modes and file organizations.

3.2.1 FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph names each file and specifies other file-related information.

General Format

```
FILE-CONTROL. {file-control-entry} ...

Format 1 - Sequential File-Control-Entry

SELECT [OPTIONAL] file-name
    ASSIGN TO literal-1
    [ , RESERVE integer-1 [ AREA  ] ]
                          [ AREAS ]
    [, ORGANIZATION IS SEQUENTIAL ]
    [, ACCESS MODE IS  SEQUENTIAL]
    [, FILE STATUS IS data-name-4] .

Format 2 - Relative File-Control-Entry

SELECT file-name
    ASSIGN TO literal-1
    [ ; RESERVE integer-1 [ AREA  ] ]
                          [ AREAS ]
    ; ORGANIZATION IS  RELATIVE
    [ ; ACCESS MODE IS { SEQUENTIAL [, RELATIVE KEY IS data-name-1] } ]
                       { {RANDOM }                                    }
                       { {DYNAMIC} RELATIVE KEY IS data-name-1        }

    [; FILE STATUS IS data-name-4 ] .
```

(continued on next page)

Format 3 – Indexed File-Control-Entry

```
SELECT file-name

    ASSIGN TO literal-1

[ ; RESERVE integer-1 [ AREA  ] ]
                      [ AREAS ]

    ; ORGANIZATION IS INDEXED

[ ; ACCESS MODE IS { SEQUENTIAL } ]
                   { RANDOM     }
                   { DYNAMIC    }

    ; RECORD KEY IS data-name-2

[ ; ALTERNATE RECORD KEY IS data-name-3 [WITH DUPLICATES] ] ...

[ ; FILE STATUS IS data-name-4 ] .
```

Syntax Rules

All Formats

1. Specify the SELECT clause first in the file control entry. Clauses following the SELECT clause can appear in any order.
2. You must name each file described in a Data Division file-description-entry once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the file control paragraph must have a corresponding file-description-entry in the Data Division.
3. Literal-1 must be an alphanumeric literal.
4. If you do not specify the ACCESS MODE IS clause, the compiler assumes sequential access as a default.
5. You can qualify data-name-1, data-name-2, data-name-3, and data-name-4.
6. Data-name-4 must be defined in the Working-Storage Section of the Data Division as a 2-character alphanumeric data item.

Format 1

7. Specify the OPTIONAL phrase only for input files that need not be present whenever the object program is executed.
8. If you do not specify the ORGANIZATION IS SEQUENTIAL clause, the compiler assumes sequential organization as a default.

Format 2

9. Specify the RELATIVE KEY phrase for a file if it will be referenced in a START statement.

10. Data-name–1 must not be defined in a record-description-entry associated with file-name.

11. The data item referenced by data-name–1 must be defined as an unsigned integer.

Format 3

12. The data items referenced by data-name–2 and data-name–3 must each be defined as alphanumeric data items in a record-description-entry associated with that file-name.

13. Neither data-name–2 nor data-name–3 can describe a variable-sized item.

14. Data-name–3 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name–2 or by any other data-name–3 associated with this file.

General Rules

All Formats

1. The ASSIGN clause specifies the default file specification of the file referenced by file-name. Literal–1 must be a file specification in command-string format. (See Section 4.1.7, VALUE OF ID Clause).

2. The ORGANIZATION clause specifies the logical organization of data in a file. The file organization is established at the time a file is created. Once established, the file organization cannot be changed.

3. If you specify the FILE STATUS clause, a value is placed into the 2-character data item (data-name–4) during the execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, or WRITE statement and before the execution of any applicable USE procedure. This value indicates the result of any input-output operation.

   The leftmost character position of the FILE STATUS data item is known as Status Key 1. It is set to one of the following values upon completion of an input-output operation:

   0 = Successful Completion
   1 = At End
   2 = Invalid Key
   3 = Permanent Error
   9 = DEC–Defined

The rightmost character position is known as Status Key 2. It further describes the results of the input-output operation. This character will contain one of the following values:

0 = No Further Information
1 = Sequence Error
2 = Duplicate Key
3 = No Record Found
4 = Boundary Violation
5 = Allocation Failure
6 = Buffer Failure
7 = No File Found
8 = Close Error
9 = Close Reel Error

Possible combinations of Status Keys 1 and 2 are shown in Table 3-2. Appendix C contains a complete listing of the File Status Keys and a description of each.

Table 3-2: Possible Combinations of Status Keys 1 and 2

Status Key 1	Status Key 2									
	No Further Info. (0)	Sequence Error (1)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)	Allocation Failure (5)	Buffer Failure (6)	No File Found (7)	CLOSE Error (8)	CLOSE REEL Error (9)
Successful Completion (0)	X		X(***)							
At End (1)	X									
Invalid Key (2)		X(***)	X(**)	X(**)	X(**)					
Permanent Error (3)	X				X(*)					
DEC-Defined (9)		X(!)	X(!!)	X(!!!)	X	X	X	X	X	

* Valid for sequentially organized files only.
** Valid for indexed and relative files only.
*** Valid for indexed files only.
! File locked by another process.
!! Record locked by another process.
!!! No sequential READ previous to a REWRITE or DELETE operation.

Format 1

4. The RESERVE clause specifies the number of input-output areas allocated for sequential files. This number equals the value of integer–1, which cannot be greater than 127. If the RESERVE clause is not specified, the number of input-output areas is determined by the Record Management Services (RMS) default.

5. Sequential files are accessed by predecessor/successor record relationships established by the execution of WRITE statements when the file is created or extended.

Format 2

6. The RESERVE clause specifies the number of input-output areas allocated for relative files. This number equals the value of integer–1, which cannot be greater than 127. If the RESERVE clause is not specified, the number of input-output areas is determined by the Record Management Services (RMS) default.

7. When the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization. This sequence follows the order of ascending relative record numbers of existing records in the file.

8. If the access mode is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

9. When the access mode is dynamic, records in the file can be accessed sequentially and/or randomly.

10. Relative record numbers uniquely identify all records stored in a relative file. The relative record number of a given record specifies the logical ordinal position of the record in the file. The first logical record has a relative record number of one (1), and subsequent logical records have relative record numbers of 2, 3, 4, ...

11. The data item specified by data-name–1 is used to communicate a relative record number between the program and Record Management Services.

Format 3

12 The RESERVE clause specifies the number of input-output areas allocated for indexed files. This number equals the value of integer–1, which must be greater than 1 and not greater than 127. If the RESERVE clause is omitted, the number of input-output areas is determined by the Record Management Services (RMS) default.

13. When the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization. This sequence follows the order of ascending record key values in a given key of reference.

14. If the access mode is random, the value of the record key data item specifies the record to be accessed.

15. When the access mode is dynamic, records in the file can be accessed sequentially and/or randomly.

16. The RECORD KEY clause specifies the prime record key for the file and provides an access path to records in an indexed file. The values of the prime record key must be unique among file records.

17. An ALTERNATE RECORD KEY clause specifies an alternate record key for the file. It provides an alternate access path to records in an indexed file.

18. Retain the same data descriptions of data-name-2 and data-name-3, and their same relative locations in a record, as those used when the file was created. Retain the same alternate key specifications as well.

19. The DUPLICATES phrase specifies that the value of the associated alternate record key can be duplicated in any of the file records. If you do not specify the DUPLICATES phrase, the value of the associated alternate record key must not be duplicated in any of the records in the file.

3.2.2 I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph specifies the memory area to be shared by different files and the location of sequential files on a multiple-file tape.

General Format

```
I-O-CONTROL.
[, SAME [RECORD] AREA FOR file-name-1 {, file-name-2} ...] ...
[, MULTIPLE FILE TAPE CONTAINS file-name-3 [POSITION integer-1]
    [, file-name-4 [POSITION integer-2]] ... ] ...
[, APPLY PRINT-CONTROL ON file-name-5 [,file-name-6]...]... .
```

Syntax Rules

1. You can include more than one SAME clause in a program.
2. A file-name must not appear in more than one SAME AREA clause or in more than one SAME RECORD AREA clause.
3. If one or more file-names in a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in the SAME RECORD AREA clause. However, additional file-names can also appear in that SAME RECORD AREA clause (even if they do not appear in the SAME AREA clause).
4. The files referenced in a SAME AREA clause or a SAME RECORD AREA clause (or both) need not have the same organization or access mode.

General Rules

1. The SAME AREA clause specifies that two or more files are to use the same memory area during processing. Therefore, it is not valid to have more than one of the files open at the same time.
2. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing the current logical record. More than one (or all) of the files can be open at the same time. A logical record in the shared area is considered as a logical record (1) of each opened, output file whose file-name appears in this SAME RECORD AREA clause, and (2) of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area, i.e., records are aligned on the leftmost character position.

3. The one-file-open rule for the SAME AREA clause takes precedence over the multiple-files-open rule for the SAME RECORD AREA clause.

4. The MULTIPLE FILE clause is for documentation purposes only. It is used when more than one file shares the same physical reel of tape. Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION clause need not be given. If any file in the sequence is not listed, the position relative to the beginning of the tape must be given. Not more than one file on the same tape reel can be open at one time.

5. The compiler uses default techniques when the APPLY clause is not present; hence, the clause is always optional, as the following explanation makes clear.

   If the FD entry does not specify a LINAGE clause, you can specify the APPLY PRINT-CONTROL clause for a printable file. The APPLY PRINT-CONTROL clause supplies a default LINAGE clause.

   If you specify neither APPLY PRINT-CONTROL nor LINAGE for a sequential file, a WRITE statement with the ADVANCING option will include formatting information in the record.

Chapter 4
Data Division

Function

The Data Division describes the data that the object program receives as input, manipulates, creates, and produces as output.

General Format

```
DATA DIVISION.
[FILE SECTION.
     [file-description-entry  {record-description-entry}...] ...]
[WORKING-STORAGE SECTION.
     [77-level-description-entry ]
     [record-description-entry   ] ... ]
[LINKAGE SECTION.
     [77-level-description-entry ]
     [record-description-entry   ] ... ]
```

Syntax Rules

1. The Data Division must follow the Environment Division in every COBOL program.
2. The Data Division must begin with the reserved words Data Division followed by a period and a space.

General Rules

File Section

The File Section describes the program's files. It begins with a section header followed by file-description-entries and record-description-entries.

1. **File-Description-Entry**

   The file-description-entry consists of a level indicator (FD), a file-name, and a series of independent clauses. These clauses describe the size of physical and logical records, the presence or absence of label records, and the names of the data records that are described for the file. The entry itself is terminated by a period.

2. **Record-Description-Entry**

   A record-description-entry is a set of data-description-entries that describe the characteristics of a particular record. Each data-description-entry consists of a level-number followed, as required, by a data-name and a series of independent clauses.

   A record description has a hierarchical structure; therefore, the clauses used in an entry can vary considerably, depending upon whether or not the entry is followed by subordinate entries.

Working-Storage Section

The Working-Storage Section begins with the section header followed by entries that describe records and noncontiguous data items. Each Working-Storage Section record name (and each data-name for noncontiguous data items) must be unique.

1. **Noncontiguous Working-Storage**

   Noncontiguous elementary items are data items in Working-Storage that bear no hierarchical relationship to one another and are not grouped into records. Each of these items is defined in a separate data-description-entry.

2. **Working-Storage Records**

   Working-Storage records are data elements and constants in Working-Storage that bear a definite hierarchical relationship to one another and are grouped into records according to the rules for forming record descriptions.

3. **Initial Values**

   Specify the initial value of any item in the Working-Storage Section, except an index data item, by using the VALUE clause in the data-description-entry (see Section 4.3.12, VALUE Clause). The initial value of any index data item is unpredictable.

Linkage Section

The Linkage Section in a program is meaningful only if: (1) the object program is to function under the control of a CALL statement (see Section 5.11), and (2) the USING phrase in the Procedure Division header is not empty (see Section 5.2).

The Linkage Section begins with the section header followed by record-description-entries. These entries describe data available through the calling program but to be referred to in both the calling and the called program. No space is allocated in the program for data items defined in the Linkage Section. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. In the case of index-names, no such correspondence is established; index-names in the calling and called programs always refer to separate indexes.

Data items defined in the Linkage Section of the called program can be referenced in the Procedure Division of that program if and only if they are:

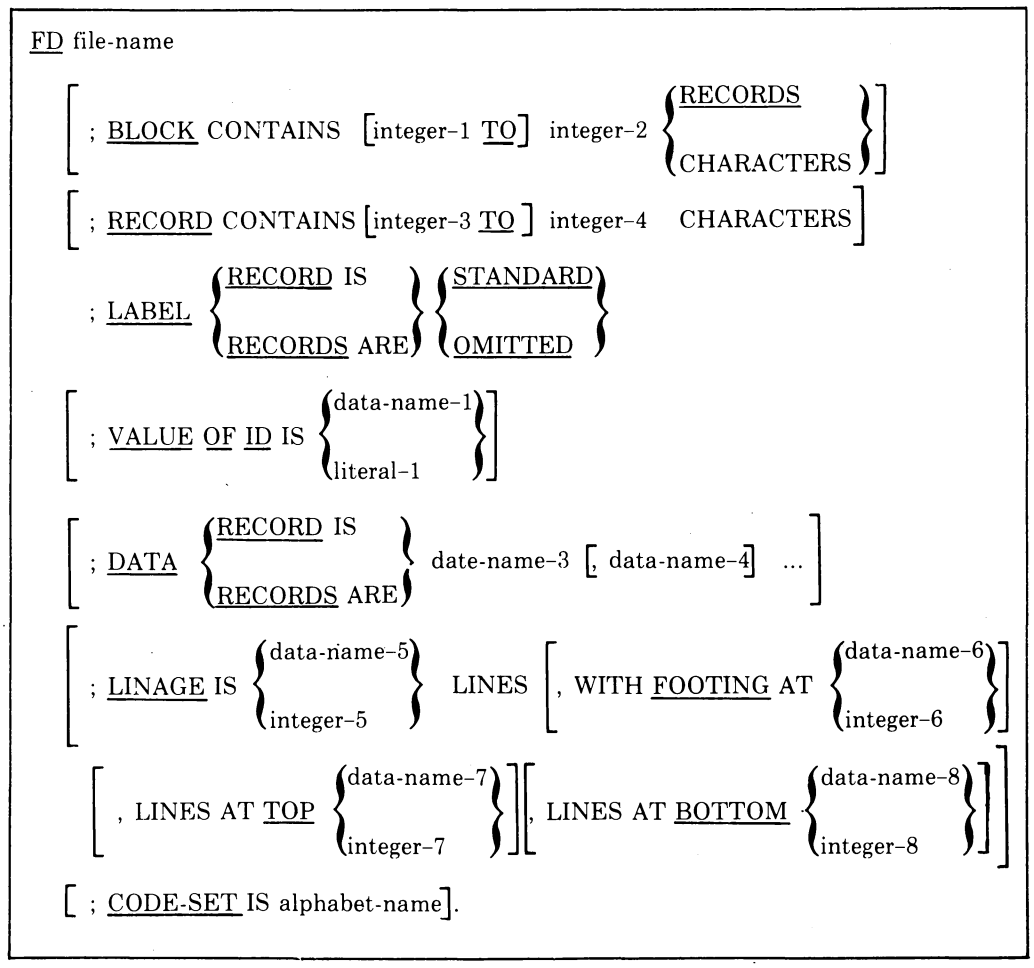
1. Operands of the USING phrase of the Procedure Division header.
2. Subordinate to operands of the USING phrase of the Procedure Division header.
3. Defined with a REDEFINES or RENAMES clause, the object of which is an operand of the USING phrase of the Procedure Division header.
4. Items subordinate to any of the items defined in number 3 above.
5. Condition-names and index-names associated with data items that meet any of the above conditions.

4.1 File Description – Complete Entry Skeleton

Function

The file description gives information about the physical structure, identification, and record names of a file.

General Format



Syntax Rules

1. The level indicator FD identifies the beginning of a file description and must appear before the file-name.
2. Clauses that follow the file-name are frequently optional, and their order of appearance is immaterial.
3. One or more record-description-entries must follow the file-description-entry.

4.1.1 BLOCK CONTAINS Clause

Function

The BLOCK CONTAINS clause specifies the mapping of a logical record into physical blocks recorded on the storage medium.

General Format

```
BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS    }
                                        { CHARACTERS }
```

Syntax Rules

The reserved word RECORD does not appear in this clause; therefore, if integer-2 has the value 1, write the clause as BLOCK CONTAINS 1 RECORDS.

General Rules

1. Integer-1, if present, is ignored.
2. Block size can be stated in terms of RECORDS.
   a. For a file of fixed-length records that is assigned to magnetic tape, each block except the last will contain integer-2 records. Integer-1, if present, is ignored. (See Section 4.1.6, RECORD CONTAINS Clause.)
   b. For a file of variable-length records that is assigned to magnetic tape, the compiler calculates the buffer size by multiplying the largest record size, plus four bytes, by the value of integer-2. (See Section 4.1.6, RECORD CONTAINS Clause.)
   c. For a sequential file assigned to a disk device, there are no unused bytes in any block and the records can span block boundaries.
   d. For files with relative or indexed organization assigned to a directory device, the compiler uses the value of integer-2 to calculate the size of the block. Because of overhead bytes, this size may or may not be equal to the record size times integer-2. (The User's Guide fully describes the compiler's algorithms for computing block size.)
3. Block size can be stated in terms of CHARACTERS.
   a. For files assigned to magnetic tape, the size of the block is the maximum of either:
      (1) Integer-2 bytes, or
      (2) The size of the largest record (add four overhead bytes for variable-length records).

b. For files with sequential organization assigned to a disk device, records are packed together in each physical block. There are no unused bytes in any block, and the records can span block boundaries.

c. For files with relative or indexed organization, the block size is integer-2 bytes. Integer-2 must be at least as large as the largest record, plus any overhead bytes, and should be a multiple of 512 bytes. (The User's Guide further discusses block size computation.)

4. When you do not specify a BLOCK CONTAINS clause, block size is calculated as follows:

   a. For files assigned to magnetic tape, the block size is the size of the largest record plus any overhead bytes.

   b. For files with sequential organization assigned to a disk device, the records are packed together in each physical block. There are no unused bytes in any block, and the records can span block boundaries.

   c. For files with relative or indexed organization, the block size is the smallest number of physical blocks that can contain one record, plus any overhead bytes.

4.1.2 CODE-SET Clause

Function

The CODE-SET clause specifies the character code set used to represent data on an external medium.

General Format

```
CODE-SET IS alphabet-name
```

Syntax Rules

1. When you specify the CODE-SET clause for a file, all data in that file must be described as USAGE IS DISPLAY; any signed numeric data must be described with the SIGN IS SEPARATE clause.
2. You can specify the CODE-SET clause only for files with sequential organization.

General Rules

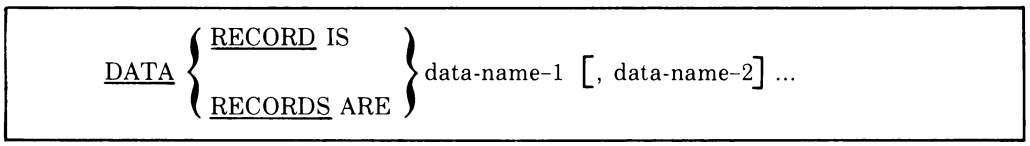
1. Alphabet-name specifies the character code convention used to represent data on the external medium. It also specifies the algorithm for converting character codes on the external medium from/to native character codes. This conversion occurs during the execution of an input or output operation. (See Section 3.1.3, SPECIAL-NAMES Paragraph.)
2. If you do not specify the CODE-SET clause, the compiler assumes the native character code set as the default.

4.1.3 DATA RECORDS Clause

Function

The DATA RECORDS clause serves only as documentation for the names of a file's data records.

General Format



Syntax Rule

Data-name-1 and data-name-2 are the names of data records associated with the file.

General Rules

1. This optional clause is for documentation purposes only. The compiler does not check the names of the records against the names appearing in the 01 record descriptions that follow the file description.
2. Conceptually, all data records of a file share the same area, even if there are multiple data record descriptions.

4.1.4 LABEL RECORDS Clause

Function

The LABEL RECORDS clause specifies the presence or absence of labels.

General Format

```
LABEL { RECORD IS   } { STANDARD }
      { RECORDS ARE } { OMITTED  }
```

Syntax Rule

This clause is required in every file-description-entry.

General Rules

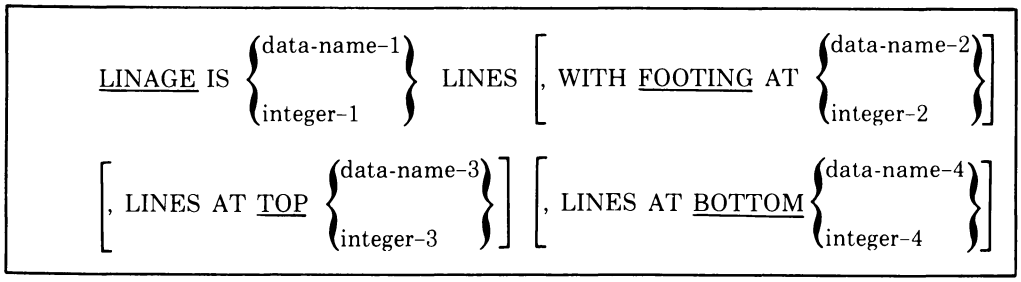
1. STANDARD specifies that labels meeting file system label specifications exist for the file or for the device to which the file is assigned.
2. OMITTED specifies that no explicit labels exist for the file or for the device to which the file is assigned.
3. Specify STANDARD for all files assigned to directory devices.
4. Specify OMITTED only for files assigned to non-directory devices.

4.1.5 LINAGE Clause

Function

The LINAGE clause specifies the number of lines on a logical page for sequential output files. It also specifies the size of the top and bottom margins on the logical page, and the logical line number at which the footing area begins.

General Format



Syntax Rules

1. Data-name-1, data-name-2, data-name-3, and data-name-4 must reference elementary, unsigned, numeric, integer data items.
2. The value of integer-1 must be greater than zero.
3. The value of integer-2 must not be greater than that of integer-1.
4. The value of integer-3 and integer-4 can be zero.

General Rules

1. The LINAGE clause pertains to sequential output files only.
2. The LINAGE clause specifies the number of lines on a logical page. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If you do not specify the LINES AT TOP or LINES AT BOTTOM phrases, the values for these functions are zero. If you do not specify the FOOTING phrase, the assumed value equals either integer-1 or the contents of the data item referenced by data-name-1, whichever is specified.

   There is no necessary relationship between the size of a logical page and the size of a physical page.
3. The value of integer-1, or the data item referenced by data-name-1, specifies the number of lines that can be written and/or spaced in the page body. This value must be greater than zero.
4. The value of integer-3, or the data item referenced by data-name-3, specifies the number of lines in the top margin of a logical page. This value can be zero.

5. The value of integer-4, or the data item referenced by data-name-4, specifies the number of lines in the bottom margin of a logical page. This value can be zero.

6. The value of integer-2, or the data item referenced by data-name-2, specifies the line number in the page body at which the footing area begins. This value must be greater than zero and less than or equal to the value of integer-1 or the data item referenced by data-name-1.

   The footing area is that area of the logical page between the line represented by the value integer-2 (or the data item referenced by data-name-2) and the line represented by the value integer-1 (or the data item referenced by data-name-1), inclusive.

7. During the execution of an OPEN statement with the OUTPUT phrase specified, the values of integer-1, integer-3, and integer-4 are used to specify the number of lines in the sections of a logical page. The value of integer-2 is used at that time to define the footing area. These values are used for all logical pages written during a given execution of the program.

8. The values of the data items referenced by data-name-1, data-name-3, and data-name-4 are used as follows:

   a. When an OPEN statement with the OUTPUT phrase is executed for the file, the data item values are used to specify the number of lines to be in each indicated section of the first logical page.

   b. When a WRITE statement with the ADVANCING PAGE phrase is executed or page overflow condition occurs, the data item values are used to specify the number of lines to be in each indicated section of the next logical page. (See Section 5.42, WRITE Statement.)

9. When an OPEN statement with the OUTPUT phrase is executed for the file, the data item value referenced by data-name-2 is used to define the footing area for the first logical page. When a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, the value will be used to define the footing area for the next logical page.

10. The presence of a LINAGE clause generates a LINAGE-COUNTER. At any given time, the LINAGE-COUNTER value represents the current line position in the page body. The rules governing the LINAGE-COUNTER are as follows:

    a. The compiler supplies a separate LINAGE-COUNTER for each file whose file-description-entry contains a LINAGE clause.

b. Procedure Division statements can reference (but not modify) LINAGE-COUNTER. Because more than one LINAGE-COUNTER can exist in a program, you must qualify LINAGE-COUNTER by file-name when necessary. LINAGE-COUNTER is implicitly defined as a one-word COMPUTATIONAL item.

c. During the execution of a WRITE statement to a file, LINAGE-COUNTER is automatically modified according to the following rules:

   (1) When you specify the ADVANCING PAGE phrase of the WRITE statement, the LINAGE-COUNTER is automatically reset to one.

   (2) When you specify the ADVANCING identifier-2 or integer phrase of the WRITE statement, the LINAGE-COUNTER is incremented by the integer or by the value of the data item referenced by identifier-2.

   (3) When you do not specify the ADVANCING phrase of the WRITE statement, the LINAGE-COUNTER is incremented by the value one. (See Section 5.42, WRITE Statement.)

   (4) The value of LINAGE-COUNTER is automatically reset to one when each successive logical page begins. (See Section 5.42, WRITE Statement.)

d. The value of LINAGE-COUNTER is automatically set to one when an OPEN statement is executed for the associated file.

11. Each logical page immediately follows the one before with no additional spacing.

4.1.6 RECORD CONTAINS Clause

Function

The RECORD CONTAINS clause specifies the size of data records.

General Format

```
RECORD CONTAINS [integer-1 TO ] integer-2 CHARACTERS
```

General Rules

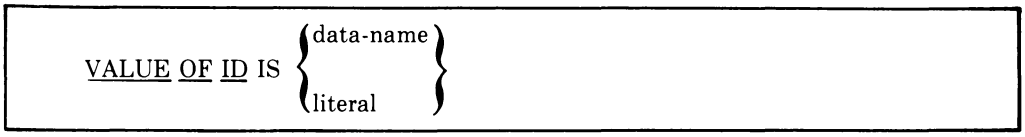
1. The record-description-entry completely defines the size of each data record; therefore, the RECORD CONTAINS clause cannot change the memory storage allocated to the records.
2. For a relative file, record size on the storage medium is fixed and equal to a value that is large enough to hold the largest record described in the file. This rule is not affected by the RECORD CONTAINS clause.
3. For a sequential or indexed file, record size on the storage medium can be fixed or variable. If the record descriptions for a file yield variable record sizes, the record storage areas allocated on the storage medium will vary in size and will be preceded by a byte count word supplied automatically by Record Management Services.
4. If the record descriptions for a file all yield record sizes that are the same size, the record storage areas allocated on the storage medium will be fixed in size and will not be preceded by a byte count word. However, you can force a variable size record format, with a byte count word prefix on each record, by using a RECORD CONTAINS clause with the "integer-1 TO" phrase.
5. Do not use integer-2 by itself unless all data records in the file have the same size; in this case, integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 both appear, they refer to the number of characters in the smallest size data record and the number in the largest size data record, respectively.
6. Record size is specified in terms of the number of character positions needed to store the logical record, regardless of the character types used to represent the items in the logical record. It is determined by summing the number of characters in all elementary items, plus any characters required by implicit or explicit synchronization.
7. Except for forcing a variable record storage size on the medium, the RECORD CONTAINS clause is for documentation only.

4.1.7 VALUE OF ID Clause

Function

The VALUE OF ID clause particularizes the description of an item in the label records associated with a file.

General Format



Syntax Rules

1. Do not use the VALUE OF ID clause when you specify LABEL RECORDS ARE OMITTED.
2. Data-name cannot be subscripted or indexed, nor can it be described with the USAGE IS INDEX clause.
3. Data-name must be in the Working-Storage Section.
4. Data-name must be an alphanumeric elementary item.
5. Literal must be a alphanumeric literal.

General Rules

1. For an existing file, the VALUE OF ID literal or data-name supplies information that enables Record Management Services to locate and identify the desired file.
2. For an output file, the VALUE OF ID literal or data-name supplies information that enables Record Management Services to create the desired file.
3. The VALUE OF ID literal or data-name is interpreted as a file specification in command string format. File specifications are discussed in the User's Guide.

4.2 Data Description Concepts

This section presents basic data description concepts. Actual COBOL clauses used to describe data appear in Section 4.3.

COBOL makes data as computer-independent as possible. Therefore, data used in COBOL programs is described using a standard rather than equipment-oriented format. This standard data format uses the decimal system to represent numbers and characters from the computer character set to describe alphanumeric data items.

4.2.1 Physical Aspects of a File

The physical aspects of a file describe data as it appears on the input or output medium. Physical aspects include such features as:

1. The mapping of logical records into the physical structure of the file medium, and
2. The ways of identifying a file.

4.2.2 Record Concepts

It is important to distinguish between a logical record and a physical record. A COBOL logical record is a uniquely identifiable group of related information that is treated as a unit. A physical record is a physical unit of information; its size and recording mode are hardware dependent and bear no direct relationship to the size of the file contained on a device.

One or more logical records can be contained in a single physical unit; or, in the case of formatted storage media, a logical record can require more than one physical unit to contain it. Source language methods describe the relationships of logical records to physical units. Using these predefined relationships, COBOL input-output statements then allow access to logical records through the facilities of the hardware-software system.

NOTE:

In this manual, the term "record" refers to a logical record, unless the term "physical record" is specifically used.

The concept of a logical record is not restricted to file data, but is carried over into the definition of Working Storage. Thus, Working Storage can be grouped into logical records and defined by a series of record-description-entries.

4.2.3 Record Description

A record description consists of a set of data-description-entries that describe the characteristics of a record. Each entry consists of a level-number followed by a data-name, if required, and a series of independent clauses, as required.

Levels

Logical records are described as hierarchical structures. The level concept arises from the need to specify subdivisions of a record, and to even further subdivide records to permit progressively more detailed data definition.

The basic and indivisible subdivision of a record is the elementary item. A record either consists of a sequence of elementary items or is itself an elementary item.

A group item is a set of elementary items. Each group item consists of a named sequence of one or more elementary items. Group items, in turn, can combine to form group items containing one or more group items, etc. Thus, an elementary item can belong to more than one group item.

Level-Numbers

A system of level-numbers shows the hierarchical organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (though not necessarily successive) level-numbers not greater in value than 49; special level-numbers -- 66, 77, and 88 -- are exceptions to this rule. Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items that are immediately subordinate to a given group item must be described using identical level-numbers; these level-numbers must be greater than the level-number used to describe that group item.

Three types of entries exist for which there is no true concept of level. These are:

1. Entries that identify RENAMES items,
2. Entries that specify noncontiguous Working Storage data items, and
3. Entries that specify condition-names.

Entries that specify RENAMES items have been assigned the special level-number 66. They can be used only as described in Format 2 of the Data Description Entry Skeleton. (See Section 4.3.)

Entries that specify noncontiguous data items have been assigned the special level-number 77. They are not subdivisions of other items and cannot themselves be subdivided.

Entries that specify condition-names to be associated with particular values of a conditional variable have been assigned the special level-number 88.

4.2.4 Classes of Data

The five categories of data items (alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited) are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric.

Every elementary item except an index data item belongs to one of the classes and, further, to one of the categories. The class of a group item is treated as alphanumeric regardless of the class of elementary items subordinate to it. Table 4–1 shows the relationship of the classes and categories of data items.

Table 4–1: Classes and Categories of Elementary and Group Data Items

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited
		Alphanumeric Edited
		Alphanumeric
Non-Elementary (Group)	Alphanumeric	Alphabetic
		Numeric
		Numeric Edited
		Alphanumeric Edited
		Alphanumeric

4.2.5 Selection of Numeric Character Representation

The value of a numeric item can be represented in binary, decimal, or packed-decimal form. The form can be selected by using the USAGE clause of the data-description-entry.

4.2.6 Algebraic Signs

Algebraic signs fall into two categories: operational signs and editing signs. The former are associated with signed numeric data items to indicate their algebraic properties; the latter appear in edited items to identify their sign.

The SIGN clause permits you to explicitly state the location of the operational sign. The clause is optional; if it is not used, operational signs are represented by a default.

Editing signs are inserted into a data item by using the sign control symbols of the PICTURE clause.

4.2.7 Standard Alignment Rules

The standard rules for positioning data within an elementary item when moving data depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:

   a. The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end, as required.

   b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character and is aligned as in paragraph 1.a above.

2. If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end, as required, within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in the JUSTIFIED Clause. (See Section 4.3.3, JUSTIFIED Clause.)

4.2.8 Item Alignment for Increased Object-Code Efficiency

All binary items are automatically aligned and SYNCHRONIZED RIGHT. (See Section 4.3.11, USAGE Clause.) The SYNCHRONIZED clause can be used to control word alignment of DISPLAY or packed-decimal data.

All index data items are automatically SYNCHRONIZED RIGHT and occupy two bytes.

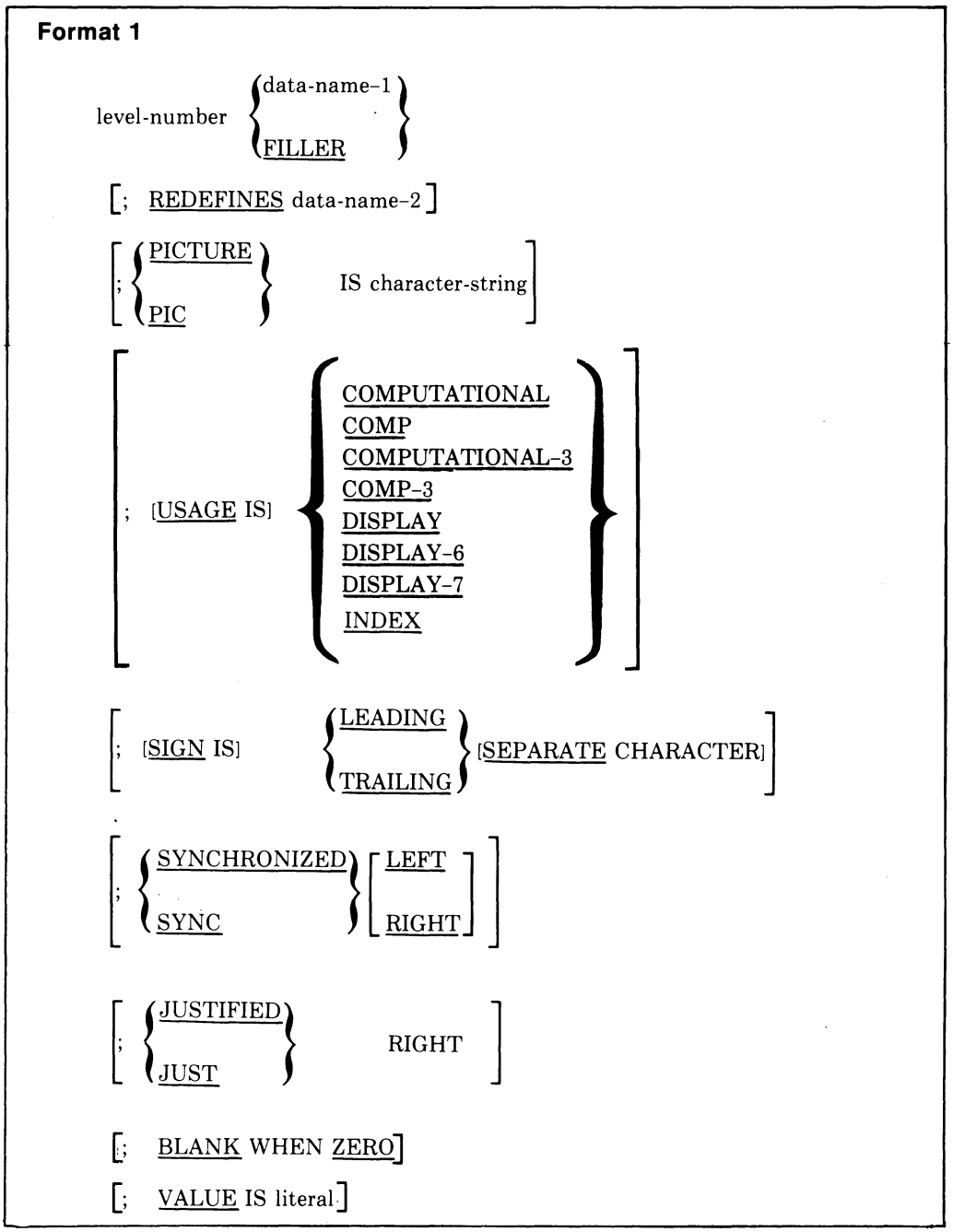
4.3 Data Description – Complete Entry Skeleton

Function

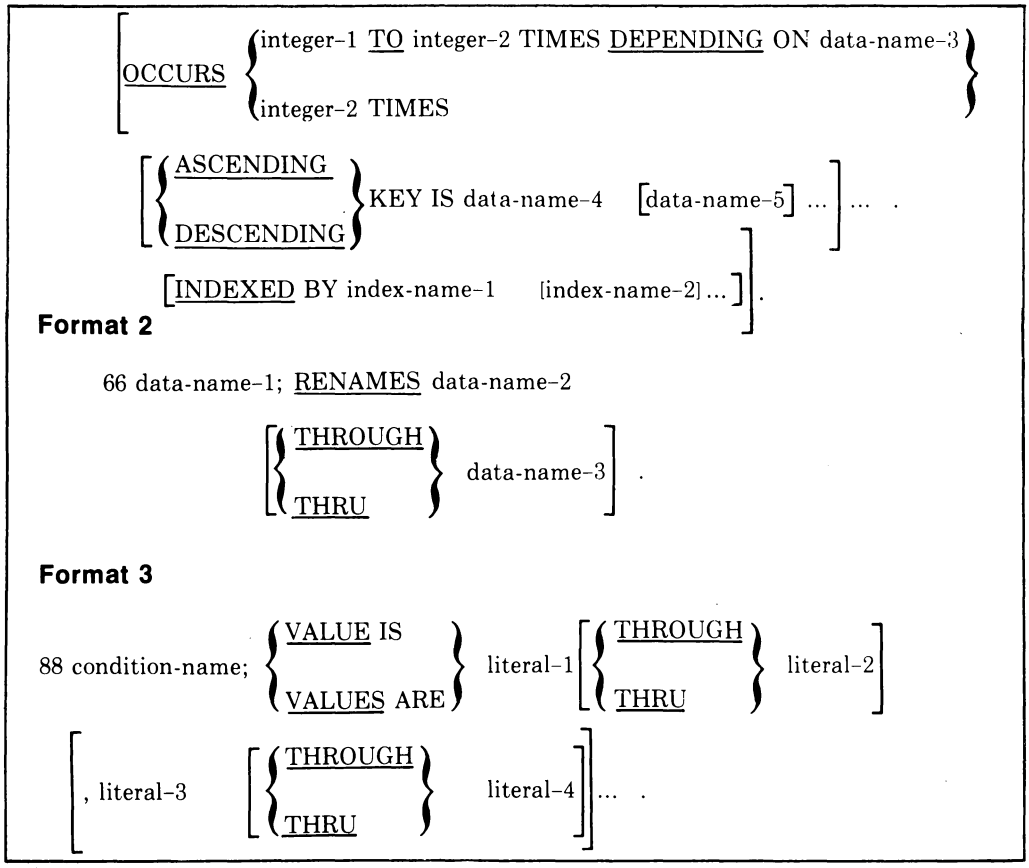
A data-description-entry specifies the characteristics of a particular item of data.

General Format

Format 1



(continued on next page)



Syntax Rules

1. The level-number in Format 1 can be any number from 01–49 or 77.
2. You can write the clauses in any order, with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.
3. The PICTURE clause must be specified for all elementary items except index data items; for these items, the PICTURE clause is not permitted.
4. The words THRU and THROUGH are equivalent.

General Rules

1. A data-name is a user-defined word that names a data item. When used in the general formats, data-name represents a word that can neither be subscripted nor indexed unless specifically permitted by the rules of that format. A data-name must contain at least one alphabetic character; however, it need not begin with an alphabetic character; the alphabetic characters can be positioned anywhere in the data-name. Qualification is sometimes permitted; therefore, data-names need not always be unique.

2. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses can be specified only for an elementary data item.

3. Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. The entry contains the name of the condition and the value(s) or range of values associated with the condition-name. Condition-name entries for a conditional variable must follow the defining entry for the associated item. A condition-name can be associated with any data-description-entry containing a level-number, except the following:

   a. Another condition-name,

   b. A group containing items with descriptions, including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY),

   c. An index data item, and

   d. A level 66 item.

4.3.1 BLANK WHEN ZERO Clause

Function

The BLANK WHEN ZERO clause causes an item to be filled entirely with spaces when its value is zero.

General Format

```
BLANK WHEN ZERO
```

Syntax Rule

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric edited or numeric. (See Section 4.3.6, PICTURE Clause.)

General Rules

1. When the BLANK WHEN ZERO clause applies to an item that is used as a receiving field for a numeric value, the item will contain nothing but spaces if the value being stored is 0.
2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

4.3.2 Data–Name or FILLER Clause

Function

A data-name specifies the name of the data being described. The word FILLER specifies an elementary item of the logical record that is not to be referenced explicitly.

General Format

```
{ data-name }
{ FILLER    }
```

Syntax Rule

In the File, Working–Storage and Linkage Sections, a data-name or the key word FILLER must be the first word following the level-number in each data-description-entry.

General Rule

The key word FILLER can be used to name an elementary item in a record. Under no circumstances can you refer explicitly to a FILLER item. However, you can use the key word FILLER as a conditional variable; such use does not require explicit reference to the FILLER item.

4.3.3 JUSTIFIED Clause

Function

The JUSTIFIED clause specifies non-standard positioning of data within a receiving data item.

General Format

Syntax Rules

1. The JUSTIFIED clause can be specified only at the elementary item level.
2. JUST is an abbreviation for JUSTIFIED.
3. The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.

General Rules

1. The leftmost characters of a receiving item are truncated when: (a) the receiving data item is described with the JUSTIFIED clause, and (b) the sending data item is larger than the receiving item. Data is aligned at the rightmost character position in the item with space fill for the leftmost character positions when: (a) the receiving data item is described with the JUSTIFIED clause, and (b) the receiving item is larger than the sending data item.
2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. (See Section 4.2.7, Standard Alignment Rules.)

4.3.4 Level–Number

Function

The level-number defines the hierarchy of data within a logical record. It also identifies entries for non-contiguous working storage items, condition-names, and the RENAMES clause.

General Format

```
level-number
```

Syntax Rules

1. A level-number must be the first element in each data-description-entry.
2. Data-description-entries subordinate to an FD entry must have level-numbers with the values 01–49, 66, or 88.
3. Data-description-entries in the Working–Storage Section and Linkage Section must have level-numbers with the values 01–49, 66, 77, or 88.

General Rules

1. The level-number 01 identifies the first entry in each record description.
2. Special level-numbers have been assigned to certain entries where no real concept of level applies:
   a. Level-number 77 identifies noncontiguous working storage data items. It can be used only as described by Format 1 of the data description skeleton. (See Section 4.3.)
   b. Level-number 66 applies to entries that define RENAMES items. It can be used only as described in Format 2 of the data description skeleton. (See Section 4.3.)
   c. Level-number 88 applies to entries that define condition-names associated with a conditional variable. It can be used only as described in Format 3 of the data description skeleton. (See Section 4.3.)
3. Multiple level 01 entries subordinate to an FD level indicator are implicit redefinitions of the same area.

4.3.5 OCCURS Clause

Function

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indexes.

General Format

Format 1

```
OCCURS integer-2 TIMES

      [ { ASCENDING  } KEY IS data-name-2 [,data-name-3]... ]...
        { DESCENDING }

      [INDEXED BY index-name-1 [, index-name-2] ... ]
```

Format 2

```
OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1

      [ { ASCENDING  } KEY IS data-name-2 [,data-name-3]... ]...
        { DESCENDING }

      [INDEXED BY index-name-1 [,index-name-2]...]
```

Syntax Rules

1. Where both integer-1 and integer-2 are used, the value of integer-1 must be less than the value of integer-2. Integer-1 must be greater than or equal to 1.
2. The data description of data-name-1 must describe a positive integer.
3. Data-name-1, data-name-2, data-name-3, ... can be qualified.
4. Data-name-2 must be either the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.
5. Data-name-3, etc., must be the name of an entry subordinate to the group item that is the subject of this entry.
6. An INDEXED BY phrase is required if the subject of this entry (or an entry subordinate to this entry) is to be referred to by indexing. The index-name identified by this clause cannot be defined elsewhere, because its storage allocation and format are hardware-dependent and are not associated with any data hierarchy.

7. A data-description-entry that contains Format 2 of the OCCURS clause can be followed in its record description only by subordinate data-description-entries.
8. The OCCURS clause cannot be specified in a data-description-entry that:
   a. Has a 01, 77, or an 88 level-number, or
   b. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.
9. In Format 2, the data item defined by data-name-1 must not occupy a character position with in the range of: (a) the first character position defined by the data-description-entry containing the OCCURS clause, and (b) the last character position defined by the record-description-entry containing that OCCURS clause.
10. If data-name-2 is not the subject of this entry, then:
    a. All the items identified by the data-names in the KEY IS phrase must be in the group item that is the subject of this entry.
    b. Items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause.
    c. No entry containing an OCCURS clause can appear between the items identified by the data-names in the KEY IS phrase and the subject of this entry.
11. Index-name-1, index-name-2, ... must be unique words in the program.

General Rules

1. The OCCURS clause is used to define tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name that is the subject of this entry must either be subscripted or indexed whenever it appears in a Procedure Division statement other than SEARCH. Further, if the subject of this entry is the name of a group item, then all data-names subordinate to the group entry must be subscripted or indexed whenever they are used as operands, except as the object of a REDEFINES clause.
2. Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

3. The number of occurrences of the subject entry is defined as follows:

   a. In Format 1, the value of integer-2 specifies the number of occurrences.

   b. In Format 2, the current value of the data item referenced by data-name-1 represents the number of occurrences.

      Format 2 specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences, while the value of integer-1 represents the minimum. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

      The value of the data item referenced by data-name-1 must fall within the range integer-1 through integer-2. Reducing the value of the data item referenced by data-name-1 means that the contents of data items whose occurrence numbers now exceed the value of the data item referenced by data-name-1 cannot be referenced.

4. When you reference a group item having subordinate to it an entry specifying Format 2 of the OCCURS clause, only that part of the table area specified by the value of data-name-1 is used in the operation.

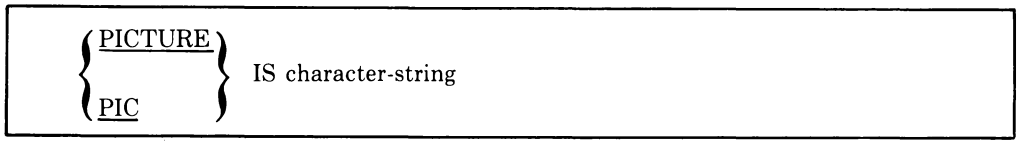
5. The KEY IS phrase indicates that repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. The specific order is determined according to the rules for comparison of operands. (See Section 5.6.3, Comparison of Numeric Operands, and Section 5.6.4, Comparison of Alphanumeric Operands.) The data-names are listed in their descending order of significance.

4.3.6 PICTURE Clause

Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

General Format



Syntax Rules

1. A PICTURE clause can be specified only at the elementary item level.
2. The PICTURE clause must be specified for all elementary items except index data items; for these items, the clause is not allowed.
3. A character-string consists of symbols that are allowable combinations of characters in the COBOL character set. The allowable combinations determine the category of the elementary item.
4. The maximum number of characters allowed in the character-string is 30.
5. PIC is an abbreviation for PICTURE.
6. When the asterisk is used as the zero suppression symbol, it cannot appear in the same entry as the BLANK WHEN ZERO clause.

General Rules

1. Five categories of data can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.
2. To define an item as alphabetic:
   a. Its PICTURE character-string can contain only the symbols A and B, and
   b. Its contents, when represented in standard data format, can be any combination of the 26 letters of the alphabet (A–Z) and the space.
3. To define an item as numeric:
   a. Its PICTURE character-string can contain only the symbols 9, P, S, and V. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive, regardless of sign.

b. If unsigned, its contents, when represented in standard data format, must be a combination of the numerals 0 through 9; if signed, the item can also contain a +, –, or other representation of an operational sign. (See Section 4.3.9, SIGN Clause.)

4. To define an item as alphanumeric:

   a. Its PICTURE character-string is restricted to certain combinations of the symbols A, X, 9, and the item is treated as if the character-string contained all Xs. A PICTURE character-string that contains all As or all 9s does not define an alphanumeric item.

   b. Its contents, when represented in standard data format, are allowable characters in the computer character set.

5. To define an item as alphanumeric edited:

   a. Its PICTURE character-string is restricted to certain combinations of the following symbols: A, X, 9, B, 0, and /. The character-string must contain at least one of the following combinations:

      1. B and X
      2. 0 and X
      3. / and X
      4. 0 and A
      5. / and A

   b. When represented in standard data format, the contents are allowable characters in the computer character set.

6. To define an item as numeric edited:

   a. Its PICTURE character-string is restricted to certain combinations of the following symbols: B, /, P, V, Z, 0, 9, ,(comma), .(period), *, +, –, CR, DB, and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and from the editing rules.

      1. The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18, inclusive.

      2. The character-string must contain at least one 0, B, /, Z, *, +, ,(comma), .(period), –, CR, DB, or currency symbol.

   b. The contents of the character positions of those symbols that are allowed to represent a digit in standard data format must be numerals (0–9).

7. The number of character positions occupied by an elementary item in standard data format (its size) is determined by the number of allowable symbols that represent character positions. An integer enclosed in parentheses following the symbols A, ,(comma), X, 9, P, Z, *, B, /, 0, +, –, or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols can appear only once in a given PICTURE: S, V, .(period), CR, and DB.

8. The functions of the symbols used to describe an elementary item are as follows:

   A Each A in the character-string represents a character position that can contain only a letter of the alphabet or a space.

   B Each B represents a character position into which a space character will be inserted.

   P Each P indicates an assumed decimal scaling position. It specifies the location of an assumed decimal point when the point is not in the number that appears in the data item.

   The scaling position character P is not counted in determining the size of the data item. Scaling position characters are counted, however, in determining the maximum number of digit positions (18) in numeric edited items or numeric items.

   P can appear only as the leftmost or rightmost part of a PICTURE description as a continuous string of Ps; because P implies an assumed decimal point (to the left of Ps if Ps are leftmost PICTURE characters and to the right if Ps are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description. Furthermore, the character P and the insertion character . (decimal point) cannot both occur in the same PICTURE character-string.

   In any operation involving conversion of data from one form of internal representation to another, if the data item being converted is described with the PICTURE character P, each digit position described by a P is considered to contain the value 0, and the size of the data item is considered to include the digit positions so described.

   S The S indicates the presence, but neither the representation nor, necessarily, the position of an operational sign; it must be written as the leftmost character in the PICTURE. The S is not counted in determining the size of the elementary item unless the entry is subject to a SIGN clause that specifies the optional SEPARATE CHARACTER phrase. (See Section 4.3.9, SIGN Clause.)

V — The V indicates the location of an assumed decimal point. It can appear only once in a character-string. The V does not represent a character position and, therefore, is not counted in determining the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.

X — Each X in the character-string represents a character position that contains any allowable character from the computer character set.

Z — Each Z can be used only to represent the leftmost leading numeric character positions that will be replaced by a space character when the content of that character position is 0. Each Z is counted in determining the size of the item.

9 — Each 9 represents a character position that contains a numeral; it is counted in determining the size of the item.

0 — Each 0 (zero) represents a character position into which the numeral 0 will be inserted. The 0 is counted in determining the size of the item.

/ — Each / (stroke) represents a character position into which the stroke character will be inserted. The / is counted in determining the size of the item.

, — Each , (comma) represents a character position into which the comma character will be inserted. This character position is counted in determining the size of the item. The insertion character , (comma) must not be the last character in the PICTURE character-string.

For a given program, the functions of the period and comma are exchanged if the DECIMAL-POINT IS COMMA clause appears in the SPECIAL-NAMES paragraph: the rules for the period will apply to the comma (and vice versa) whenever these symbols appear in a PICTURE clause.

. — The . (period) is an editing symbol that represents the decimal point for alignment purposes. It also represents a character position into which the character .(period) will be inserted. The insertion character .(period) must not be the last character in the PICTURE character-string. It is counted in determining the size of the item.

+, −, CR, DB — These editing sign-control symbols represent the character position into which these symbols will be placed. The symbols are mutually exclusive in any one character-string, and each character used in the symbol is counted in determining the size of the data item.

* Each * (asterisk) represents a leading numeric character position into which an * will be placed when the content of that position is 0. Each * is counted in determining the size of the item.

cs The currency symbol represents a character position into which a currency symbol is to be placed. It is represented either by the default currency sign ($) or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in determining the size of the item.

Editing Rules

1. Editing in the PICTURE clause can be performed either by insertion or by suppression and replacement. There are four types of insertion editing available. They are:

   a. Simple insertion
   b. Special insertion
   c. Fixed insertion
   d. Floating insertion

   There are two types of suppression and replacement editing:

   a. Zero suppression and replacement with spaces

   b. Zero suppression and replacement with asterisks

2. The type of editing that can be performed depends on the data-item category. The following table specifies the allowable types of editing by category:

Table 4-2: Types of Editing by Data Category

Category	Type of Editing
Alphabetic	Simple insertion B only
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion 0, B and /
Numeric Edited	All, subject to Editing Rule 3

3. Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement can be used with zero suppression.

4. *Simple Insertion Editing.* The , (comma), B (space), 0 (zero), and / (stroke) are used as the insertion characters. They are counted in determining the size of the item and represent the position in the item into which the character will be inserted.

5. *Special Insertion Editing.* The . (period) is used as the insertion character. In addition, it also represents the decimal point for alignment purposes. The insertion character (used for the actual decimal point) is counted in determining the size of the item. The use of the assumed decimal point, ('V') and the actual decimal point (represented by the insertion character) in the same PICTURE character-string is not allowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

6. *Fixed Insertion Editing.* The currency symbol and the editing sign control symbols +, -, CR, and DB are the insertion characters. Only one currency symbol and one of the editing sign-control symbols can be used in a given PICTURE character-string.

   When the symbols CR or DB are used, they represent two character positions in determining the size of the item; they must represent the rightmost character positions that are counted in determining the size of the item. The symbols + or - must be either the leftmost or rightmost character position to be counted in determining the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a + or a - symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it did in the PICTURE character-string.

   Editing sign-control symbols produce the following results, depending upon the value of the data item:

Table 4-3: Editing with Sign-Control Symbols

Editing Symbol In Picture Character-String	Result	
	Data Item Positive or Zero	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

7. *Floating Insertion Editing.* The currency symbol and editing sign-control symbols + or – are the floating insertion characters. They are mutually exclusive in a given PICTURE character-string.

   Floating insertion editing is indicated by using a string of at least two of the floating insertion characters. This string can contain any of the fixed insertion symbols or have fixed insertion characters immediately to its right. The simple insertion characters are part of the floating string.

   The leftmost (rightmost) character of the floating insertion string represents the leftmost (rightmost) limit of the floating symbol in the data item.

   The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Non-zero, numeric data can replace all characters at or to the right of this limit.

   There are only two ways of representing floating insertion editing: (1) the insertion character can represent any or all of the leading numeric character positions on the left of the decimal point, or (2) the insertion character can represent all of the numeric character positions in the PICTURE character-string.

   a. If the insertion characters are only to the left of the decimal point, a single floating insertion character will be placed into the character position immediately preceding either the decimal point or the first non-zero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions to the left of the insertion character are replaced with spaces.

   b. If the insertion character represents all of the numeric character positions, the result depends upon the value of the data. If the value is 0, the entire data item will contain spaces. If the value is not 0, the result is the same as in (a).

   To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of non-floating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

8. *Zero Suppression Editing.* The suppression of leading 0s in numeric character positions is indicated by the use of the Z or the * (asterisk) as suppression symbols. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If Z is used, the replacement character will be the space, and if the asterisk is used, the replacement character will be *.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions that are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression: (1) suppression symbols can represent any or all of the leading numeric character positions to the left of the decimal point, or (2) suppression symbols can represent all of the numeric character positions in the PICTURE character-string.

a. If the suppression symbols appear only to the left of the decimal point, any leading 0 in the data that corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

b. If the suppression symbols represent all numeric character positions and the value of the data is not 0, the result is the same as if the suppression characters were not specified. If the value is 0, the entire data item will be spaces if the suppression symbol is Z or all asterisks (except for the actual decimal point) if the suppression symbol is *.

9. The symbols +, –, *, Z, and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Precedence Rules

The following table shows the order of precedence when using characters as symbols in a character-string. An X at an intersection indicates that the symbol(s) at the top of the column can precede the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol "cs".

At least one of the symbols A, X, Z, 9 or *, or at least two of the symbols +, –, or cs, must be present in a PICTURE string.

The non-floating insertion symbols + and –, the floating insertion symbols Z, *, +, –, and cs, and other symbol P appear twice in the following table. The leftmost column and uppermost row for each symbol represent its use to the left of the decimal point position. The second appearance of the symbol represents its use to the right of the decimal point position.

Table 4-4: PICTURE Character Precedence Table

First Symbol / Second Symbol		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ −}	{+ −}	{CR DB}	cs	{Z *}	{Z *}	{+ −}	{+ −}	cs	cs	9	{A X}	S	V	P	P
Non-Floating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x
	.	x	x	x	x		x			x	x		x		x		x					
	{+ −}																					
	{+ −}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	cs						x															
Floating Insertion Symbols	{Z *}	x	x	x	x		x			x	x											
	{Z *}	x	x	x	x	x	x			x	x	x								x		x
	{+ −}	x	x	x	x					x			x									
	{+ −}	x	x	x	x	x				x			x	x						x		x
	cs	x	x	x	x		x								x							
	cs	x	x	x	x	x	x								x	x				x		x
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x
	{A X}	x	x	x													x	x				
	S																					
	V	x	x	x	x		x			x	x		x		x		x		x		x	
	P	x	x	x	x		x			x	x		x		x		x		x		x	
	P						x			x									x	x		x

4.3.7 REDEFINES Clause

Function

The REDEFINES clause allows different data-description-entries to describe the same computer storage area.

General Format

```
level-number data-name-1; REDEFINES data-name-2
```

NOTE:

Level-number, data-name-1, and the semicolon are shown in the above format to improve clarity. They are not part of the REDEFINES clause.

Syntax Rules

1. The REDEFINES clause must immediately follow data-name-1.

2. The level-numbers of data-name-1 and data-name-2 must be identical; they must not be 66 or 88. (Level 77 items can be redefined.)

3. This clause must not be used in level 01 entries in the File Section.

4. The data-description-entry for data-name-2 cannot contain a REDEFINES clause; however, data-name-2 can be subordinate to an item whose data-description-entry contains a REDEFINES clause. The data-description-entry for data-name-2 cannot contain an OCCURS clause. However, data-name-2 can be subordinate to an item whose data-description-entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause cannot be subscripted or indexed. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS Clause. (See Section 4.3.5, OCCURS Clause.)

5. No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 can occur between the data-description-entries of data-name-2 and data-name-1.

General Rules

1. Redefinition starts at the area allocated to data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

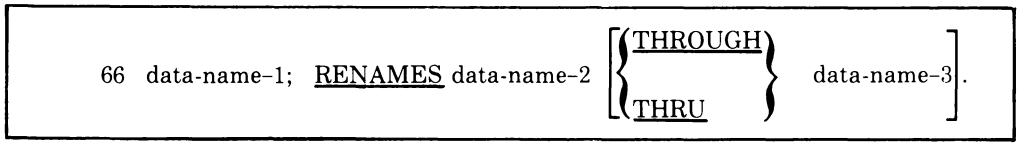
2. When the level-number of data-name–1 is other than 01, it must specify the same number of character positions contained in the data item referenced by data-name–2. Note that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.
3. Multiple redefinitions of the same character positions are permitted. The entries giving the new descriptions of the character positions must follow the entries defining the area being redefined, without intervening entries that define new character positions. Multiple redefinitions of the same character positions must all use the data-name of the entry that originally defined the area.
4. The entries giving the new description of the character positions must not contain any VALUE clauses, except in condition-name entries.
5. Multiple level 01 entries subordinate to an FD level indicator represent implicit redefinitions of the same area.

4.3.8 RENAMES Clause

Function

The RENAMES clause permits alternative, possibly overlapping, groupings of elementary items.

General Format



NOTE:

Level-number 66, data-name-1 and the semicolon are shown in the above format to improve clarity. They are not part of the RENAMES clause.

Syntax Rules

1. All RENAMES entries referring to data items within a given logical record must immediately follow the last data-description-entry of the associated record-description-entry.
2. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same logical record and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry, nor can it rename a 77, 88, or 01 level entry.
3. Data-name-1 cannot be used as a qualifier and can only be qualified by the names of the associated level 01 or FD entries. Neither data-name-2 nor data-name-3 can have an OCCURS clause in its data-description-entry; nor can either be subordinate to an item that has an OCCURS clause in its data-description-entry.
4. The beginning of the area described by data-name-3 must not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must be to the right of the end of the area described by data-name-2. Data-name-3, therefore, cannot be subordinate to data-name-2.
5. Data-name-2 and data-name-3 can be qualified.
6. The words THRU and THROUGH are equivalent.
7. No item within the range including data-name-2 and data-name-3 can have a variable size as defined in the OCCURS clause. (See Section 4.3.5, OCCURS Clause.)

General Rules

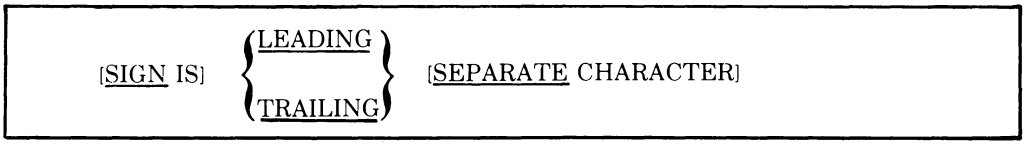
1. One or more RENAMES entries can be written for a logical record.
2. When data-name-3 is specified, data-name-1 is a group item that includes all elementary items: (1) starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item) and (2) concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).
3. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item; when data-name-2 is a group (elementary) item, data-name-1 is treated as a group (elementary) item.

4.3.9 SIGN Clause

Function

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to explicitly describe these properties.

General Format



Syntax Rules

1. The SIGN clause can be specified only for a numeric data-description-entry whose PICTURE contains the character S, or a group item containing at least one such numeric data-description-entry.
2. The numeric data-description-entries to which the SIGN clause applies must be described as USAGE IS DISPLAY.
3. At most one SIGN clause can apply to any given numeric data-description-entry.

General Rules

1. The SIGN clause specifies the position and the mode of representation of the operational sign for the numeric data-description-entry to which it applies, or for each numeric data-description-entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data-description-entries whose PICTURE contains the character S; the S indicates the presence of the operational sign (though not its representation or, necessarily, its position).
2. A numeric data-description-entry whose PICTURE contains the S, but to which no SIGN clause applies, has an operational sign. In this default case, the sign is a part of the right-most, or trailing, digit in the item (much like an overpunch).
3. If the SEPARATE CHARACTER phrase is not present, then:
   a. The operational sign is associated with the leading (or trailing) digit position of the elementary numeric data item.
   b. The letter S in the PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

c. The digit position containing the operational sign holds a character whose value represents both a numeric digit and the algebraic sign of the item. The allowable characters for all combinations of the numeric digits, and the positive and negative sign values, are:

		DIGIT VALUES									
		1	2	3	4	5	6	7	8	9	0
SIGN	POSITIVE	A	B	C	D	E	F	G	H	I	{
	NEGATIVE	J	K	L	M	N	O	P	Q	R	}

4. If the SEPARATE CHARACTER phrase is present, then:

   a. The operational sign is the leading (or trailing) character position of the elementary numeric data item; this character position is not a digit position.

   b. The letter S in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

   c. The operational signs for positive and negative are the standard data format characters + and –, respectively.

5. Every numeric data-description-entry whose PICTURE contains the character S is a signed numeric data-description-entry. If a SIGN clause applies to such an entry, and conversion is necessary for computation or comparison purposes, conversion takes place automatically.

4.3.10 SYNCHRONIZED Clause

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on a word boundary in computer memory. (See Section 4.2.8, Item Alignment for Increased Object-Code Efficiency.)

General Format

```
{SYNCHRONIZED} [LEFT ]
{SYNC        } [RIGHT]
```

Syntax Rules

1. This clause can appear only with an elementary item.
2. SYNC is an abbreviation for SYNCHRONIZED.

General Rules

1. Use of this clause aligns the subject data item on memory word boundaries such that no other data item occupies any of the words delimiting the data item. A memory word contains two character positions. If the number of character positions required to store the data item is odd, the unused character is not used for any other data item. These unused character positions, however, are included in:
   a. The size of any group item(s) to which the elementary item belongs.
   b. The character positions redefined when this data item is the object of a REDEFINES clause.
2. SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be synchronized left.
3. SYNCHRONIZED LEFT specifies that the elementary item is to begin at the even byte address of the memory word.
4. SYNCHRONIZED RIGHT specifies that the elementary item is to terminate on the odd byte address of the memory word.
5. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used to determine any action that depends on size, such as justification, truncation or overflow.
6. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

7. All binary items and all INDEX items are automatically SYNCHRONIZED and occupy an integral number of words. (See Section 4.3.11, USAGE Clause.)

8. When the SYNCHRONIZED clause is specified for an item in the scope of an OCCURS clause, each occurrence is SYNCHRONIZED.

9. A fill byte is added to the end of each occurrence of a group item if all of the following conditions are met:

   a. One or more items within the group item are SYNCHRONIZED (implicitly or explicitly).

   b. The data description of the group item contains an OCCURS clause.

   c. The size of the group is odd after synchronization of the items within it.

   The group size then becomes even, causing each occurrence of an item within the group to align on memory boundaries in the same manner as the first occurrence of the item.

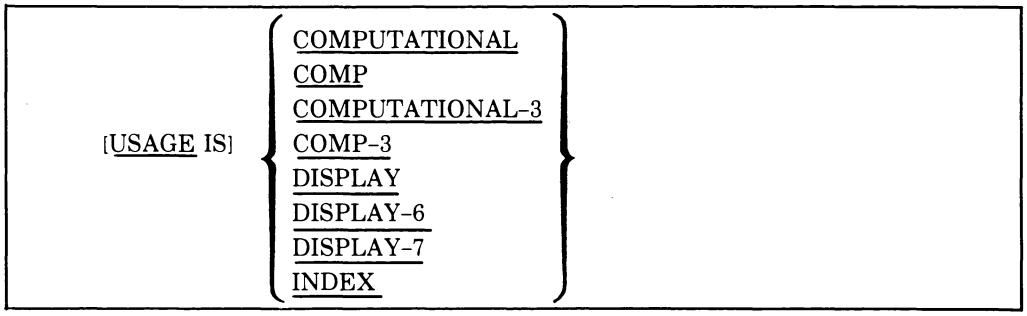
10. All record descriptions in both the File Section and Working-Storage Section, and all noncontiguous data items in the Working-Storage Section, are automatically SYNCHRONIZED.

4.3.11 USAGE Clause

Function

The USAGE clause specifies the format of a data item in the computer's storage.

General Format



Syntax Rules

1. COMP is an abbreviation for COMPUTATIONAL.
2. COMP-3 is an abbreviation for COMPUTATIONAL-3.
3. The PICTURE character-string of a COMP or COMP-3 item can contain only 9s, the operational sign character S, the implied decimal point character V, and one or more Ps. (See Section 4.3.6, PICTURE Clause.)
4. DISPLAY, DISPLAY-6, and DISPLAY-7 are equivalent.
5. An index data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.
6. The SYNCHRONIZED, JUSTIFIED, PICTURE, SIGN, VALUE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

General Rules

1. The USAGE clause can be written at any level. If it is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.
2. This clause specifies the manner in which a data item is represented in the computer's storage. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division can restrict the USAGE clause of the referenced operands.

3. A COMP or COMP-3 item can represent a value to be used in computations and must be numeric. If a group item is described as COMP or COMP-3, the specification applies to the elementary items in the group, but not to the group itself; the group item cannot be used in computation.

4. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value that must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. If a group item is described with a USAGE IS INDEX clause, the elementary items in the group are all index data items. However, the group item itself is not an index data item and cannot be used in the SEARCH or SET statement or in a relation condition.

5. An index data item can be part of a group that is referred to in a MOVE or input-output statement, in which case conversion does not occur.

6. If the USAGE clause is not specified for an elementary item or for any group to which the item belongs, the USAGE is implicitly DISPLAY.

7. A COMP item is a binary value with an assumed decimal point that is automatically SYNCHRONIZED and stored in memory (in one, two, or four words) as follows:

PICTURE Range	Storage	
S9 TO S9(4)	1 word	(2 bytes)
S9(5) TO S9(9)	1 longword	(4 bytes)
S9(10) TO S9(18)	1 quadword	(8 bytes)

8. The representation of the binary value is independent of the presence of V or one or more Ps in its PICTURE character-string. The binary value of a COMP item represents the exact decimal quantity whose description is given by the PICTURE character-string as if it contained no V or P characters. However, the decimal point indicated by these characters is remembered and used to adjust the binary value before using it in arithmetic operations. Thus, the binary value represents the decimal value as though it were an integer, and decimal accuracy is achieved, although representation is binary. The internal representation of COMP items is discussed in the User's Guide.

9. A COMP-3 item is a signed packed decimal value with an assumed decimal point that is stored internally as two decimal digits per byte (byte-aligned). The maximum size of a COMP-3 item is 18 decimal digits. Its PICTURE character-string must contain an S. The item can begin in the even address byte or the odd address byte subject to the implicit or explicit synchronization. (See Section 4.3.10, SYNCHRONIZED Clause.) The internal format of COMP-3 items is fully discussed in the User's Guide.

10. A DISPLAY item is a string of bytes stored in memory as two bytes per word. The item can begin in the even address byte or the odd address byte subject to the implicit or explicit synchronization. (See Section 4.3.10, SYNCHRONIZED Clause.)

11. Index data items are stored as one-word COMP items with PIC 9(4).

    Their value is always positive.

    Index data items are implicitly SYNCHRONIZED. Thus, when they are described in record descriptions, they may cause automatic fill bytes to be supplied.

4.3.12 VALUE Clause

Function

The VALUE clause defines the initial value of Working-Storage items and the values associated with a condition-name.

General Format

Format 1

```
VALUE IS literal
```

Format 2

```
{VALUE IS  }          [{THROUGH}          ]
{VALUES ARE} literal-1 [{THRU   } literal-2]

   [, literal-3 [{THROUGH} literal-4 ]]...
                {THRU   }
```

Syntax Rules

1. The words THRU and THROUGH are equivalent.
2. A signed numeric literal must have an associated signed numeric PICTURE character-string.
3. All numeric literals in a VALUE clause of an item must have a value in the range indicated by the PICTURE clause and must not have a value that would require truncation of non-zero digits. Alphanumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

General Rules

1. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description in the hierarchy of the item. The following rules apply:

   a. If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a Working-Storage item, it is aligned in the data item according to the standard alignment rules. (See Section 4.2.7, Standard Alignment Rules.)

b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be alphanumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. (See Section 4.2.7, Standard Alignment Rules.) Editing characters in the PICTURE clause are included in determining the size of the data item, but they have no effect on its initialization.

c. Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that is specified.

2. A figurative constant can be substituted in Format 1 and Format 2 wherever a literal is specified.

Condition–Name Rules

1. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

2. Format 2 can be used only in connection with condition-names. (See Section 1.1.2.1, User–Defined Words.) Wherever the THRU phrase is used, literal–1 must be less than literal–2, literal–3 less than literal–4, etc.

Data–Description–Entries Other Than Condition–Names

1. Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

   a. In the File Section and the Linkage Section, the VALUE clause can be used only in condition-name entries.

   b. In the Working–Storage Section, the VALUE clause must be used in condition-name entries. The VALUE clause can also be used to specify the initial value of any other data item except an index data item, in which case the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item description, the initial value is undefined.

2. The VALUE clause must not be stated in a data-description-entry that either contains an OCCURS clause or is subordinate to one that contains an OCCURS clause. This rule does not apply to condition-name entries. (See Section 4.3.5, OCCURS Clause.)

3. The VALUE clause must not be stated in a data-description-entry that either contains a REDEFINES clause or is subordinate to one that contains a REDEFINES clause. This rule does not apply to condition-name entries.

4. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a alphanumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained in this group. The VALUE clause cannot be stated at the subordinate levels in this group.

5. The VALUE clause must not be written for a group containing items with descriptions that include JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

Chapter 5
Procedure Division

5.1 General Description

The Procedure Division must be included in every COBOL source program. It specifies the processing to be performed on the files and file data described in the Environment and Data Divisions. This division contains declaratives and procedures.

5.1.1 Declaratives

Declarative sections must be grouped at the beginning of the Procedure Division. They are preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES. Declarative sections detail the procedures to be followed whenever an I–O error occurs on a particular file. (See Section 5.41, USE statement.)

5.1.2 Procedures

A procedure consists of a paragraph, a group of successive paragraphs, a section, or a group of successive sections. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program. It consists of a paragraph-name or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further text appears.

A section consists of a section header followed by zero or more successive paragraphs. A section ends immediately before the next section or at the end of the Procedure Division. In the declaratives portion of the Procedure Division, the section ends at the key words END DECLARATIVES.

A paragraph consists of a paragraph-name followed by a period and a space and by zero or more successive sentences. A paragraph ends immediately before the next paragraph-name or section name or at the end of the Procedure Division. In the declaratives portion of the Procedure Division, a paragraph ends at the key words END DECLARATIVES.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

An identifier is the word or words necessary to make unique reference to a data item. (See Section 5.4.8, Uniqueness of Reference.)

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

5.2 Procedure Division Header

The Procedure Division is identified by and must begin with the following header:

PROCEDURE DIVISION [USING [data-name-1] [,data-name-2] ...] .

The USING phrase is present if, and only if, the object program is to function under the control of a CALL statement. A COBOL program which is to function under the control of a CALL statement, but which has no arguments passed to it, is specified by a USING phrase that contains no data-names (an empty USING phrase).

Each of the operands in the USING phrase of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

Within a called program, Linkage Section data items are processed according to their data descriptions given in the called program.

When the USING phrase is present, the object program operates as if data-name-1 of the Procedure Division header in the called program and data-name-1 in the USING phrase of the CALL statement in the calling program refer to a single set of data that is equally available to both the called and calling programs. Except that they must define an equal number of character positions, their descriptions need not be the same. In like manner, there is an equivalent relationship between data-name-2, ..., in the USING phrase of the called program and data-name-2, ..., in the USING phrase of the CALL statement in the calling program. A data-name must not appear more than once in the USING phrase in the Procedure Division header of the called

program; however, a given data-name can appear more than once in the USING phrase of a CALL statement.

Data items defined in the Linkage Section of the called program can be referenced within the Procedure Division of the called program if and only if they are:

1. Operands of the USING phrase of the Procedure Division header.
2. Subordinate to operands of the USING phrase of the Procedure Division header.
3. Defined with a REDEFINES or RENAMES clause, the object of which is an operand of the USING phrase of the Procedure Division header.
4. Items subordinate to any of the items defined in number 3 above.
5. Condition-names and index-names associated with data items that meet any of the above conditions.

5.3 Procedure Division Body

The body of the Procedure Division must conform to one of the following formats:

Format 1

```
[DECLARATIVES.

{section-name SECTION    [segment-number].    declarative-sentence

 [paragraph-name. [sentence] ...] ...}...

 END DECLARATIVES.]

{section-name SECTION [segment-number].

 [paragraph-name. [sentence] ...] ...}...
```

Format 2

```
{paragraph-name. [sentence] ...}...
```

5.4 Statements and Sentences

There are three types of statements: conditional, compiler-directing, and imperative.

There are three types of sentences: conditional, compiler-directing, and imperative.

5.4.1 Conditional Statement

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

A conditional statement is one of the following:

a. An IF statement or a SEARCH statement.

b. A READ statement that specifies the AT END or INVALID KEY phrase.

c. A WRITE statement that specifies the INVALID KEY or END-OF-PAGE phrase.

d. A REWRITE or DELETE statement that specifies the INVALID KEY phrase.

e. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.

f. A STRING or UNSTRING statement that specifies the ON OVERFLOW phrase.

g. A GO TO ... DEPENDING ... statement.

5.4.2 Conditional Sentence

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by a period, and followed by a space.

5.4.3 Compiler-Directing Statement

A compiler-directing statement causes the compiler to take a specific action during compilation. It consists of a compiler-directing verb (USE or COPY) and its operands.

5.4.4 Compiler-Directing Sentence

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

5.4.5 Imperative Statement

An imperative statement indicates a specific, unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement nor a compiler-directing statement. An imperative statement can consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT		
ADD(1)	GO(5)	SET
ALTER		
	INSPECT	START(2)
		STOP
CLOSE	MOVE	STRING(4)
COMPUTE(1)		
	MULTIPLY (1)	SUBTRACT (1)
DELETE (2)	OPEN	
	PERFORM	
DISPLAY	READ (3)	UNSTRING(4)
DIVIDE (1)		WRITE (2)
EXIT	REWRITE (2)	

(1) Without the optional SIZE ERROR phrase.
(2) Without the optional INVALID KEY phrase.
(3) Without the optional AT END phrase or INVALID KEY phrase.
(4) Without the optional ON OVERFLOW phrase.
(5) Without the optional DEPENDING phrase.

When imperative-statement appears in the general format of statements, it refers to that sequence of consecutive imperative statements that must be ended by a period, an ELSE phrase associated with a previous IF statement, or a WHEN phrase associated with the previous SEARCH statement.

5.4.6 Imperative Sentence

An imperative sentence is an imperative statement terminated by a period, and followed by a space.

5.4.7 Statement Categories

COBOL statements are categorized by verb type and format:

Category	Verbs
Arithmetic	ADD COMPUTE DIVIDE INSPECT (TALLYING) MULTIPLY SUBTRACT

Compiler-Directing	COPY USE
Conditional	ADD (SIZE ERROR) COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DIVIDE (SIZE ERROR) GO (DEPENDING) IF MULTIPLY (SIZE ERROR) READ (END or INVALID KEY) REWRITE (INVALID KEY) SEARCH START (INVALID KEY) STRING (OVERFLOW) SUBTRACT (SIZE ERROR) UNSTRING (OVERFLOW) WRITE (INVALID KEY or END-OF-PAGE)
Data Movement	ACCEPT (DATE, DAY, or TIME) INSPECT (REPLACING) MOVE STRING UNSTRING
Ending	STOP
Input-Output	ACCEPT CLOSE DELETE DISPLAY OPEN READ REWRITE START STOP (literal) WRITE
Procedure Branching	ALTER CALL EXIT GO TO PERFORM
Table-handling	SEARCH SET

IF is used as a verb in the COBOL language although it is not a verb in the English language.

5.4.8 Uniqueness of Reference

Uniqueness of reference in a COBOL program is accomplished by using qualifiers, subscripts, indexes, unique identifiers, and condition-names.

5.4.8.1 Qualification — Every user-specified name that defines an element in a COBOL source program must be unique, either because no other name has the identical spelling and hyphenation or because the name exists within a hierarchy of names such that references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers, and the process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. Within the Data Division, all data-names used for qualification must be associated with a level indicator or a level-number. Therefore, two identical data-names must not appear as entries subordinate to a group item unless they are capable of being made unique through qualification. In the Procedure Division two identical paragraph-names must not appear in the same section.

In the hierarchy of qualification, names associated with a level indicator are the most significant, followed, in order, by names associated with level-number 01 and names associated with level-numbers 02 through 49. A section-name is the highest (and the only) qualifier available for a paragraph-name. Thus, the most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, can be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, a name cannot be both a data-name and a procedure-name.

Qualification is performed by following a data-name, a condition-name, a paragraph-name, or a text-name by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

The general formats for qualification are:

Format 1

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

Format 2

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

The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.
2. The same name must not appear at two levels in a hierarchy.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except in the REDEFINES clause, in which qualification must not be used).

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when it is referenced from within the same section.

5. A data-name cannot be subscripted when it is being used as a qualifier.

6. A name can be qualified even though it does not need qualifications; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used. The complete set of qualifiers for a data-name must not be the same as any partial set of qualifiers for another data-name. Qualified data-names can have up to 48 qualifiers.

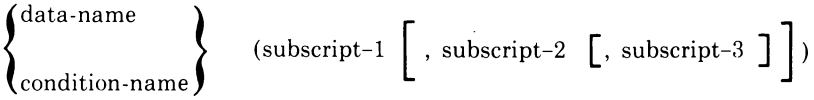
5.4.8.2 Subscripting — Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names. (See Section 4.3.5, OCCURS Clause.)

The subscript can be represented either by a numeric literal that is an integer or by a data-name. The data-name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name can be qualified but not subscripted.

The subscript can be signed and, if signed, it must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ... The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript or set of subscripts that identify the table element are delimited by a balanced pair of separators, the left and right parentheses, following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript is required, they are written in the order of successively less-inclusive dimensions of the data organization.

The format is:

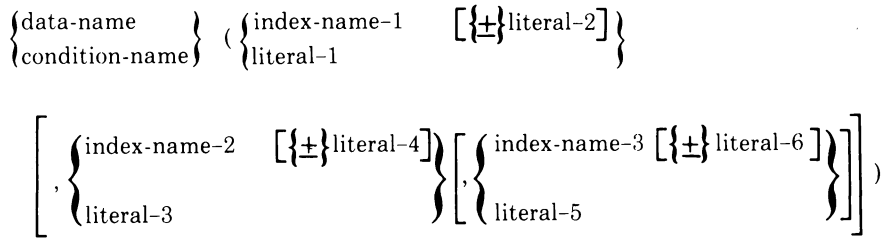


5.4.8.3 Indexing — Indexing allows references to be made to individual elements within a table of like elements. An index is assigned to a level of the table by using the INDEXED BY phrase in the table's definition. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name must be initialized before it is used as a table reference. An index-name can be given an initial value by a SET, SEARCH ALL, or Format 4 PERFORM statement.

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed, in order, by: (1) the operator + or −, and (2) an unsigned integer numeric literal, all three delimited by a balanced pair of separators, the left and right parentheses, following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (where the operator − is used) by the value of the literal, the occurrence number represented by the value of the index. When more than one index-name is required, they are written in the order of successively less-inclusive dimensions of the data organization.

At the time of execution of a statement that refers to an indexed table element, the value contained in the index-name associated with the table element must neither correspond to a value less than one (1) nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resultant from relative indexing.

The general format for indexing is:



5.4.8.4 Internal Formats of Subscripts, Index–Names and Index Data Items

1. Subscripts are stored as either binary or DISPLAY numeric integers with a size that can vary from 1 to 18 digits. They can contain an operational sign, although at the time of their use as a subscript the value must be positive.

2. Index-names are stored as two-part items consisting of a binary occurrence number and a binary index value. Both values are always positive.

3. Index data items are stored as 1 word COMP items consisting of a binary occurrence number with an implicit PIC 9(4) description. Their value is always positive.

   Index data items are implicitly SYNCHRONIZED; thus, when they are described within record descriptions they can cause automatic fill bytes to be supplied.

5.4.8.5 Identifier — An identifier is a term used to indicate that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or indexes necessary to ensure uniqueness.

The general formats for identifiers follow:

Format 1

```
data-name-1 [ { OF } data-name-2 ] ...  [ (subscript-1 [ , subscript-2
                { IN }
      [ , subscript-3 ] ) ]
```

Format 2

```
data-name-1 [ { OF } data-name-2 ]  [ ( { index-name-1 [ { + } literal-2 ] }
                { IN }                { literal-1      [ { - }           ] }
  { index-name-2 [ { + } literal-4 ] }    [ , { index-name-3 [ { + } literal-6 ] } ] ) ]
  { literal-3    [ { - }           ] }      [ , { literal-5    [ { - }           ] } ]
```

The following are restrictions on qualification, subscripting and indexing:

1. A data-name must not itself be subscripted or indexed when it is being used as an index, subscript or qualifier.
2. Indexing is not permitted where subscripting is not permitted.
3. An index name can be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data in a form called index data items.
4. Literal-1, literal-3, literal-5 in the above format must be positive numeric integers. Literal-2, literal-4, and literal-6 must be unsigned numeric integers.

5.4.8.6 Condition-Name — Each condition-name must be unique or be made unique through qualification and/or indexing or subscripting.

If qualification is used to make a condition-name unique, the associated conditional variable can be used as the first qualifier. If qualification is used, the hierarchy of names associated with the conditional variable or the conditional variable itself must be used to make the condition-name unique.

If references to a conditional variable require indexing or subscripting, then references to any of its condition-names also require the same combination of indexing or subscripting.

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names are exactly those pertaining to "identifier", except that data-name-1 is replaced by condition-name-1.

In the general formats, "condition-name" refers to a condition-name qualified, indexed or subscripted, as necessary.

5.4.9 Explicit and Implicit Specifications

There are three types of explicit and implicit specifications that occur in COBOL source programs:

1. Explicit and Implicit Procedure Division References
2. Explicit and Implicit Transfers of Control
3. Explicit and Implicit Attributes

5.4.9.1 Explicit and Implicit Procedure Division References — A COBOL source program can reference data items either explicitly or implicitly in Procedure Division statements. An explicit reference occurs when the name of the referenced item is written in a Procedure Division statement or when the name of the referenced item is copied into the Procedure Division by the processing of a COPY statement. An implicit reference occurs when the item is referenced by a Procedure Division statement without the name of the referenced item being written in the source statement. Such an implicit reference occurs if, and only if, the data item contributes to the execution of the statement.

5.4.9.2 Explicit and Implicit Transfers of Control — In a COBOL program, each statement is executed in the sequence in which it was written in the source program unless an explicit transfer of control overrides this sequence. The transfer of control from statement to statement occurs without writing an explicit Procedure Division statement and, therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without

the execution of a procedure branching statement. COBOL provides the following types of implicit control flow alterations that override the statement-to-statement transfers of control:

1. If a paragraph is being executed under control of another COBOL statement (for example, PERFORM, USE) and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs following the last statement in the paragraph to the control mechanism of the last executed controlling statement.

2. When any COBOL statement is executed that results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Note that another implicit transfer of control occurs after execution of the declarative section, as described in number 1 above.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. (See Section 5.4, Statements and Sentences.) An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The execution of the procedure branching statement ALTER does not in itself constitute an explicit transfer of control, but affects the explicit transfer of control that occurs when the associated GO TO statement is executed.

In this document, the term "next executable statement" is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules associated with each language element in the Procedure Division.

5.4.9.3 Explicit and Implicit Attributes — Attributes can be implicitly or explicitly specified. An attribute that has been explicitly specified is called an explicit attribute. If an attribute has not been specified explicitly, then the attribute takes on the default and is known as an implicit attribute.

For example, the usage of a data item need not be specified, in which case data item usage is DISPLAY.

5.5 Arithmetic Expressions

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operator and parentheses are given in Table 5–1, Combination of Symbols in Arithmetic Expressions, Section 5.5.2.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

NOTE:

Arithmetic expressions must not contain non-integer exponents.

5.5.1 Arithmetic Operators

An arithmetic operator is a single character or a fixed 2-character combination.

There are five binary arithmetic operators and two unary arithmetic operators that can be used in arithmetic expressions. They are represented by specific characters that must be preceded by and followed by a space.

Binary Arithmetic Operators	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operators	Meaning
+	The effect of multiplication by numeric literal +1.
-	The effect of multiplication by numeric literal -1.

5.5.2 Formation And Evaluation Rules

1. Parentheses can be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first; and within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   1st - Unary plus and minus
   2nd - Exponentiation
   3rd - Multiplication and division
   4th - Addition and subtraction

2. Parentheses are also used either (a) to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or (b) to modify the normal hierarchical sequence of execution in expressions where it is necessary to deviate from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which operators, variables, and parentheses can be combined in an arithmetic expression are summarized in Table 5–1, where:

   a. The letter P indicates a permissible pair of symbols.

   b. The character – indicates an invalid pair of symbols.

   c. The term variable indicates an identifier or literal.

Table 5–1: Symbol Combinations in Arithmetic Expressions

First Symbol	Second Symbol				
	Variable	* / ** – +	Unary + or –	(	)
Variable	–	P	–	–	P
* / ** + –	P	–	P	P	–
Unary + or –	P	–	–	P	–
(	P	–	P	P	–
)	–	P	–	–	P

4. An arithmetic expression can begin only with an open parenthesis, a plus sign, a minus sign, or a variable and can end only with a close parenthesis or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression; each left parenthesis is to the left of its corresponding right parenthesis.

5. Arithmetic expressions allow you to combine arithmetic operations without restrictions on composite of operands and/or receiving data items.

5.6 Conditional Expressions

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF, SEARCH, and PERFORM statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions.

5.6.1 Simple Conditions

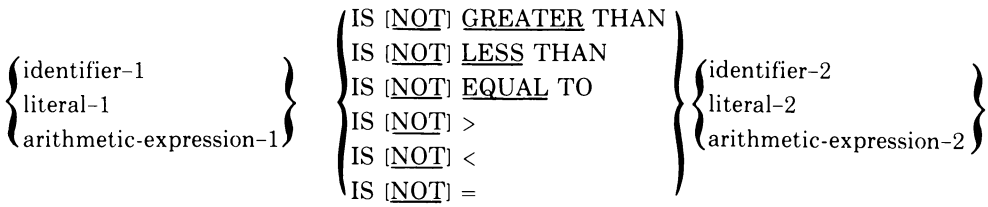
The simple conditions are the relation, class, condition-name, switch-status, and sign conditions. A simple condition has a truth value of true or false.

5.6.2 Relation Condition

A relation condition causes a comparison of two operands, each of which can be the data item referenced by an identifier or a literal or the value resulting from an arithmetic expression. A relation condition has a truth value of true if the relation exists between the operands.

Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the non-numeric comparison rules apply.

The general format of a relation condition is as follows:



NOTE:

The required relational characters >, <, and = are not underlined to avoid confusion with other symbols such as greater-than-or-equal-to.

The first operand (identifier-1, literal-1, or arithmetic-expression-1) is called the subject of the condition; the second operand (identifier-2, literal-2, or arithmetic-expression-2) is called the object of the condition. The subject and the object cannot both be literals.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, NOT and the next key word or relation character are one relational operator that defines the comparison to

be executed for truth value: e.g., NOT EQUAL is a truth test for an unequal comparison; NOT GREATER is a truth test for an equal or less comparison. The meaning of the relational operators is as follows:

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

NOTE:

The required relational characters >, <, and = are not underlined to avoid confusion with other symbols such as greater-than-or-equal-to.

5.6.3 Comparison of Numeric Operands

For operands whose class is numeric (see Section 4.2.4, Classes of Data), a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic-expression operands, in terms of number of digits, is not significant. Zero is considered a unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

5.6.4 Comparison of Alphanumeric Operands

For nonnumeric operands, or numeric and nonnumeric operand combinations, a comparison is made with respect to a specified collating sequence of characters (See Section 3.1.2, OBJECT-COMPUTER paragraph). If one of the operands is specified as numeric, it must be an integer data item (USAGE DISPLAY) or an integer literal:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this alphanumeric data item were then compared to the nonnumeric operand. (See Section 5.22, MOVE Statement and Section 4.3.6, PICTURE Clause.)

2. If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item were then compared to the nonnumeric operand. (See Section 5.22, MOVE Statement and Section 4.3.6, PICTURE Clause.)
3. A non-integer numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters it contains. Numeric and nonnumeric operands can be compared only when their usage is the same.

Comparisons can be made between operands of equal size and operands of unequal size.

1. Operands of equal size.

   Comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

   The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand containing the character that is positioned higher in the collating sequence is considered to be the greater operand.

2. Operands of unequal size.

   Comparison proceeds as though the shorter operand were extended on the right by sufficient space characters to make the operands of equal size.

5.6.5 Comparisons Involving Index–Names and/or Index Data Items

Relation tests can be made between:

1. Two index-names. The result is the same as if the corresponding occurrence numbers were compared.
2. An index-name and a data item (including an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.
3. An index data item and an index data item. The actual values are compared.

Index data items cannot be compared with literals or other data items that are not index data items.

5.6.6 Class Condition

The class condition determines whether the operand is numeric or alphabetic. Numeric consists entirely of the characters 0 through 9, with or without the operational sign. Alphabetic consists entirely of the characters A through Z and space. The general format for the class condition is as follows:

```
identifier IS [NOT]  { NUMERIC    }
                     { ALPHABETIC }
```

The usage of the operand being tested must be described as DISPLAY.

When used, NOT and the next key word specify one class condition that defines the class test to be executed for truth value, that is, NOT NUMERIC is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present. Valid operational signs for data items described with the SIGN IS SEPARATE clause are the standard data format characters, + and -. (See Section 4.3.9, SIGN Clause, for the format of valid operational signs when the SIGN IS SEPARATE clause is not present.)

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

5.6.7 Condition–Name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name. The general format for the condition-name condition is as follows:

```
condition-name
```

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

5.6.8 Switch–Status Condition

A switch-status condition determines the ON or OFF status of a numbered switch. The switch number and the ON or OFF value associated with the condition must be named in the SPECIAL-NAMES paragraph of the Environment Division. The general format for the switch-status condition is as follows:

condition-name

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

5.6.9 Sign Condition

The sign condition determines whether or not the algebraic value of a data item is less than, greater than, or equal to 0. The general format for a sign condition is as follows:

arithmetic-expression IS [NOT] { POSITIVE / NEGATIVE / ZERO }

When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value; for example, NOT ZERO is a truth test for a nonzero (positive or negative) value.

An operand is positive if its value is greater than 0, negative if its value is less than 0, and 0 if its value is equal to 0.

5.6.10 Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions and/or complex conditions with logical connectors (logical operators AND and OR) or negating these conditions with logical negation (the logical operator NOT). The truth value of a complex condition, whether parenthesized or not, is that truth value which results from the interaction of all stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically connected or logically negated.

The logical operators and their meanings are:

Logical Operator	Meaning
AND	Logical conjunction; the truth value is true if both of the conjoined conditions are true; false if one or both of the conjoined conditions are false.
OR	Logical inclusive OR; the truth value is true if one or both of the included conditions is true; false if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is true if the condition is false; false if the condition is true.

The logical operators must be preceded by a space and followed by a space.

5.6.11 Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT. The truth value for the negated simple condition is the opposite of that for the simple condition. Thus, the truth value of a negated simple condition is true if, and only if, the truth value of the simple condition is false; the truth value of a negated simple condition is false if, and only if, the truth value of the simple condition is true. The inclusion in parentheses of a negated simple condition does not change the truth value.

The general format for a negated simple condition is:

NOT simple-condition

5.6.12 Combined and Negated Combined Conditions

A combined condition results from connecting conditions with one of the logical operators AND or OR.

The general format of a combined condition is:

condition $\left\{ \begin{Bmatrix} \text{AND} \\ \text{OR} \end{Bmatrix} \text{ condition} \right\} \ldots$

where condition can be one of the following:

1. A simple condition.
2. A negated simple condition.
3. A combined condition.

4. A negated combined condition, that is, the NOT logical operator followed by a combined condition enclosed within parentheses.

5. Combinations of the above, specified according to the rules summarized in Table 5-2, Combinations of Conditions, Logical Operators, and Parentheses.

Although parentheses need not be used when either AND or OR (but not both) is used exclusively in a combined condition, they can be used to effect a final truth value when a mixture of AND, OR and NOT is used.

Table 5-2 indicates the ways in which conditions and logical operators can be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses, and each left parenthesis must be to the left of its corresponding right parenthesis.

Table 5-2: Combinations of Conditions, Logical Operators, and Parentheses

Given the following element	Location in conditional expression		In a left-to-right sequence of elements:	
	First	Last	Element, when not first, can be immediately preceded only by:	Element when not last, can be immediately followed only by:
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	NO	NO	simple-condition,)	simple-condition, NOT, (
NOT	Yes	NO	OR, AND, (	simple-condition, (
(	Yes	NO	OR, NOT, AND, (	simple-condition, NOT, (
)	NO	Yes	simple-condition,)	OR, AND,)

Thus, the element pair OR NOT is permissible, while the pair NOT OR is not permissible; NOT (is permissible, while NOT NOT is not permissible.

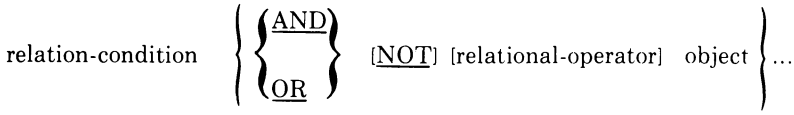
5.6.13 Abbreviated Combined Condition Relations

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that are common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first can be abbreviated by the omission of one of the following:

1. The subject of the relation condition, or

2. The subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:



Within a sequence of relation conditions both of the above forms of abbreviation can be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules of Table 5–2, Combinations of Conditions, Logical Operators, and Parentheses. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The interpretation applied to the use of the word NOT in an abbreviated combined relation condition is as follows:

1. If the word immediately following NOT is GREATER or >, LESS or <, or EQUAL or =, then the NOT participates as part of the relational operator; otherwise,

2. The NOT is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Some examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

Condition	Expanded Equivalent
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))))

5.6.14 Condition Evaluation Rules

Parentheses can be used to specify the order in which individual conditions of complex conditions are to be evaluated when it is necessary to depart from the implied evaluation precedence. Conditions within parentheses are evaluated

first, and, within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. When parentheses are not used or parenthesized conditions are at the same level of inclusiveness, the following hierarchical order of logical evaluation is implied until the final truth value is determined:

1. Values are established for arithmetic expressions. (See Formation and Evaluation Rules, Section 5.5.2.)
2. Truth values for simple conditions are established in the following order:
   a. Relation condition (following the expansion of any abbreviated relation condition)
   b. Class condition
   c. Condition-name condition
   d. Switch-status condition
   e. Sign condition
3. Truth values for negated simple conditions are established.
4. Truth values for combined conditions are established (AND logical operators, followed by OR logical operators).
5. Truth values for negated combined conditions are established.
6. When the sequence of evaluation is not completely specified by parentheses, the order of evaluation of consecutive operations of the same hierarchical level is from left to right.

5.7 Common Phrases and General Rules for Statement Formats

In the statement descriptions that follow, several phrases appear frequently: the ROUNDED phrase, the SIZE ERROR phrase, and the CORRESPONDING phrase.

In the discussion below, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

5.7.1 ROUNDED Phrase

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the low-order integer positions in a resultant-identifier are represented by the character P in the PICTURE clause for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

5.7.2 SIZE ERROR Phrase

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by 0 always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements. Then the size error condition applies to the intermediate results as well. If the ROUNDED phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR phrase is specified.

1. If the SIZE ERROR phrase is not specified and a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

2. If the SIZE ERROR phrase is specified and a size error condition occurs, then the value of the resultant-identifier(s) affected by the size errors is not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

   For the ADD statement with the CORRESPONDING phrase and the SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, the imperative statement in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

5.7.3 CORRESPONDING Phrase

If group-1 and group-2 are identifiers that refer to group items, a pair of data items, one from group-1 and one from group-2, correspond if the following conditions exist:

1. A data item in group-1 and a data item in group-2 are not designated by the key word FILLER and have the same data-name and the same qualifiers up to, but not including, group-1 and group-2.

2. In the case of a MOVE statement with the CORRESPONDING phrase, at least one of the data items is an elementary data item; in the case of the ADD statement with the CORRESPONDING phrase or the SUBTRACT statement with the CORRESPONDING phrase, both of the data items are elementary numeric data items.

3. The description of group-1 and group-2 must not contain level-number 66, 77, or 88 or the USAGE IS INDEX clause.

4. A data item that is subordinate to group-1 or group-2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as are those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, group-1 and group-2 can have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses. (See Section 4.3.5, OCCURS Clause.)

5.7.4 Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

2. The maximum size of each operand is 18 decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points, must not contain more than 18 decimal digits.

5.7.5 Multiple Results In Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements can have multiple results. Such statements behave as though they had been written in the following way:

1. A statement that performs all arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence in which the multiple results are listed.

The result of the statement

```
ADD A, B, C TO C, D (C), E
```

is equivalent to

```
ADD A, B, C GIVING temp
ADD temp TO C
ADD temp TO D (C)
ADD temp TO E
```

where temp is an intermediate result item defined as follows:

> The number of integer places in temp is the maximum of the integer places of all operands in the statement. The number of decimal places is the maximum of all the operands in the statement. If the sum of the number of integer places and decimal places is greater than 18, then the number of integer places will be reduced until the sum equals 18. Therefore, high-order truncation could occur in some receiving operands, depending on the resulting value of the arithmetic statement.

5.7.6 Overlapping Operands

When a sending and a receiving item in an arithmetic statement or INSPECT, MOVE, SET, STRING, or UNSTRING statement share a part of their storage areas, the result of the execution of such a statement is undefined. The compiler does not detect overlapping or potentially overlapping operands.

5.7.7 Incompatible Data

Except for the class condition (see Section 5.6.6, Class Condition), when the contents of a data item are referenced in the Procedure Division and the contents of that data item are not compatible with the class specified for that data item by its PICTURE clause, then the result of such a reference is undefined.

5.8 ACCEPT Statement

Function

The ACCEPT statement makes low-volume data available to the specified data item.

General Format

Format 1

```
ACCEPT identifier [FROM mnemonic-name]
```

Format 2

```
ACCEPT identifier FROM { DATE }
                       { DAY  }
                       { TIME }
```

Syntax Rule

The mnemonic-name in Format 1 must be specified in the SPECIAL-NAMES paragraph of the Environment Division and must be associated with a hardware device.

General Rules

Format 1

1. The ACCEPT statement causes the transfer of data from the hardware device. This data replaces the contents of the data item named by the identifier.
2. The ACCEPT statement causes a stream of bytes to be transferred with no editing or conversion to the data item specified by the identifier. The data item is treated as alphanumeric regardless of its class. The data is aligned at the leftmost character position of the data item with space fill or truncation to the right.
3. If the FROM mnemonic-name phrase is not specified, the hardware device is the default system input device.

Format 2

4. The ACCEPT statement causes the information requested to be transferred to the data item specified by the identifier according to the rules of the MOVE statement. DATE, DAY, and TIME are conceptual data items and, therefore, are not described in the COBOL program. Their usage is DISPLAY.

5. DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes is from high order to low order (left to right), that is, year of century, month of year, and day of month. Thus, July 4, 1976 is expressed as 760704. DATE, when accessed by a COBOL program, behaves as if it had been described in the COBOL program as an unsigned elementary numeric integer data item six digits long.

6. DAY is composed of the data elements year of century and day of year. The sequence of the data element codes is from high order to low order (left to right). That is, year of century, day of year. Thus, July 4, 1976 is expressed as 76186. DAY, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned, elementary, numeric integer data item five digits long.

7. TIME consists of the data elements hours, minutes, seconds and hundredths of a second. TIME is based on elapsed time after midnight on a 24-hour clock basis; thus, 2:41 p.m. would be expressed as 14410000. TIME, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned, elementary, numeric integer data item eight digits long. The minimum value of TIME is 00000000; the maximum value is 23595999.

5.9 ADD Statement

Function

The ADD statement adds two or more numeric operands together and stores the result.

General Format

Format 1

```
ADD { identifier-1 } [, identifier-2 ] ... TO identifier-3 [ROUNDED]
    { literal-1    } [, literal-2    ]

    [, identifier-4 [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 2

```
ADD { identifier-1 } { identifier-2 } [, identifier-3 ]
    { literal-1    } { literal-2    } [, literal-3    ] ...

    GIVING identifier-4 [ROUNDED] [, identifier-5 [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 3

```
ADD { CORRESPONDING } identifier-1 TO identifier-2 [ROUNDED]
    { CORR          }

    [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that, in Format 2, identifier-4, following the word GIVING, must refer to either an elementary numeric item or an elementary numeric edited item. In Format 3, each identifier must refer to a group item.
2. Each literal must be a numeric literal.
3. The composite of operands must not contain more than 18 digits (see Section 5.7.4, Arithmetic Statements).

   a. In Format 1, the composite of operands is determined by using all of the operands in a given statement.

b. In Format 2, the composite of operands is determined by using all of the operands in a given statement excluding the data items that follow the word GIVING.

c. In Format 3, the composite of operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value of identifier-3, and the result is stored into identifier-3. This process is repeated for each operand following identifier-3.
2. If Format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of each identifier-4, ,identifier-5, ...,.
3. If Format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.
4. The compiler ensures that enough places are carried (unless an intermediate result exceeds the 18-digit limitation) to avoid losing significant digits during execution.

5.10 ALTER Statement

Function

The ALTER statement modifies the destination of a GO TO statement.

General Format

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2

    [, procedure-name-3 TO [PROCEED TO] procedure-name-4] ...
```

Syntax Rules

1. Each procedure-name-1, procedure-name-3,..., is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.
2. Each procedure-name-2, procedure-name-4, ..., is the name of a paragraph or section in the Procedure Division.

General Rule

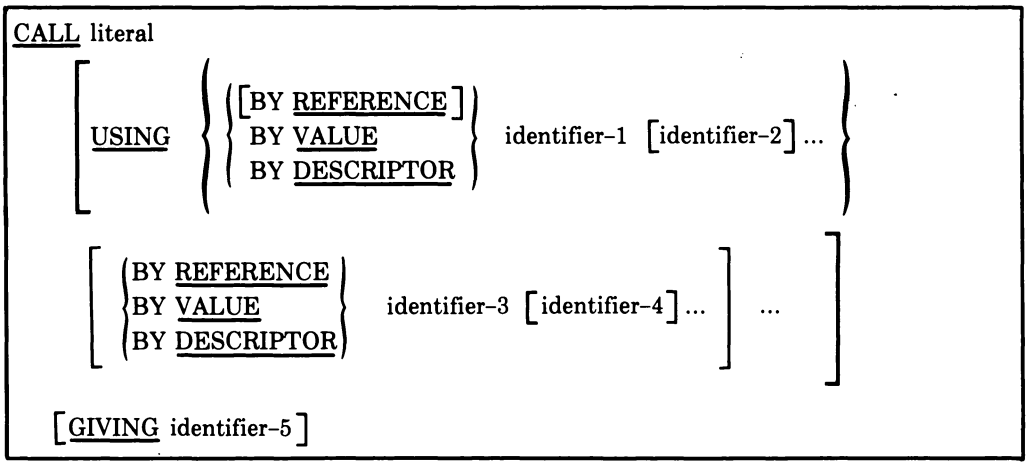
Execution of the ALTER statement modifies the GO TO statement in the paragraph named with procedure-name-1 and procedure-name-3 so that subsequent executions of the modified GO TO statements cause transfers of control to procedure-name-2, procedure-name-4, ..., respectively.

5.11 CALL Statement

Function

The CALL statement transfers control from one program to another within the executable image.

General Format



Syntax Rules

1. Literal must be a nonnumeric literal, one to 15 characters long, consisting of the characters 0–9, A–Z, $ (dollar sign), and _ (underscore). Literal is the entry point in the called subprogram. For COBOL subprograms, literal is the called program's PROGRAM-ID.
2. The same identifier can be referenced more than once in the USING phrase.
3. If an initial mechanism (REFERENCE, VALUE, or DESCRIPTOR) is not specified, BY REFERENCE is the default.
4. A mechanism applies to all identifiers following it until a new mechanism (if any) is specified.

General Rules

1. The program whose name is specified by the value of literal is the called program; the program in which the CALL statement appears is the calling program.
2. The execution of a CALL statement transfers control to the called program.
3. The CALL statement can appear anywhere in the Procedure Division of a program, regardless of its segmentation structure.

4. A called program is in its initial state the first time it is called within an image.

   On all later entries into the called program, the state of the program remains unchanged from its state when last exited. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

5. Called programs can contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

6. The USING phrase is included in the CALL statement only if there is a nonempty USING phrase in the Procedure Division header of the called COBOL program or a nonempty argument list in the header of the called non-COBOL program. The number of operands in corresponding USING phrases (or argument lists) must be identical.

7. The method by which the CALL statement makes data available to the called program is known as the mechanism. The mechanisms are:

   a. REFERENCE - The address of (pointer to) the data item is passed to the called program. This is the default mechanism; that is, arguments are passed by REFERENCE if a mechanism is not specified.

   b. VALUE - The value contained in the data item is passed to the called program. The data item must be a longword COMPUTATIONAL item with no scaling or implied decimal point; that is, the picture of the data item must be in the range S9(5) TO S9(9).

   c. DESCRIPTOR - The address of (pointer to) the descriptor of the data item is passed to the called program. The usage of the data item cannot be COMPUTATIONAL.

8. Only the REFERENCE mechanism can be used to call COBOL subprograms. Identifiers in the PROCEDURE DIVISION USING phrase of a called COBOL program are interpreted to be BY REFERENCE.

9. The order of appearance of identifiers in the USING phrase is critical. Corresponding identifiers refer to a single set of data that is available to the calling and called program. The correspondence is positional, not by name. For index-names, no such correspondence is established; therefore, index-names in the called and calling program always refer to separate indexes.

   For non-COBOL called programs, the mechanism for each identifier in the using phrase must be identical to the mechanism for each argument in the called program's argument list.

10. Identifier-5 must be defined as a COMPUTATIONAL integer with a picture in the range S9(5) to S9(9). If the called program returns a single longword (4-byte) function result, identifier-5 contains the value on return from the called program.

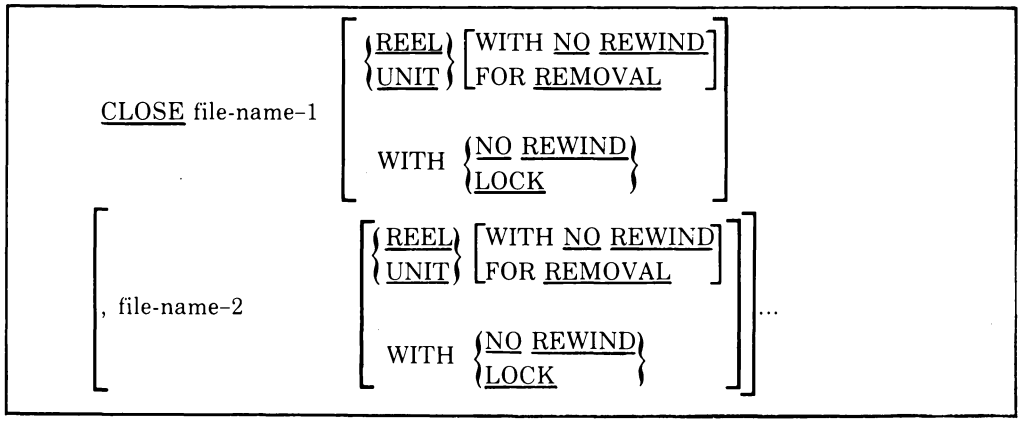
    COBOL programs cannot return a function result.

5.12 CLOSE Statement (Sequential)

Function

The CLOSE statement terminates the processing of reels/units and files with optional rewind and/or lock or removal, where applicable.

General Format



Syntax Rules

1. The REEL/UNIT phrase must be used only for sequential files.
2. The files referenced in the CLOSE statement need not all have the same organization or access.

General Rules

Except where otherwise stated in the general rules below, the terms REEL and UNIT are synonymous and completely interchangeable in the CLOSE statement. Treatment of sequential mass storage files is logically equivalent to the treatment of a file on tape or an analogous sequential medium.

1. A CLOSE statement can be executed for a file only when the file is open.
2. For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:
   a. Non-reel/unit. A file whose input or output medium is such that the concept of rewind and reels/units has no meaning.
   b. Sequential single-reel/unit. A sequential file that is entirely contained on one reel/unit.
   c. Sequential multi-reel/unit. A sequential file that is contained on more than one reel/unit.

3. The results of executing each type of CLOSE for each category of file are summarized in Table 5-3.

Table 5-3: Relationship of CLOSE Statement Formats to File Categories*

CLOSE Statement Format	File Category		
	Non-Reel/Unit	Sequential Single-Reel/Unit	Sequential Multi-Reel/Unit
CLOSE	C	C,G	C,G,A
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A
CLOSE WITH NO REWIND	X	C,B	C,B,A
CLOSE REEL/UNIT	X	X	F,G
CLOSE REEL/UNIT FOR REMOVAL	X	X	F,D,G
CLOSE REEL/UNIT WITH NO REWIND	X	X	F,B

*The definitions of the symbols in the table are given below. Where the definition depends on whether the file is an input, output, or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A Previous Reels/Units Unaffected

Input Files and Input-Output Files:

All reels/units in the file prior to the current reel/unit are processed according to the standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement. If the current reel/unit is not the last in the file, the reels/units in the file following the current one are not processed.

Output Files:

All reels/units in the file prior to the current reel/unit are processed according to the standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement.

B No Rewind of Current Reel

The current reel/unit is left in its current position.

C Close File

Input Files and Input-Output Files:

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the Record Management Services. Closing operations specified by the Record Management Services are executed. If the file is positioned at its end and label records are not specified for the file, label processing does not take place, but other closing operations specified by the Record Management Services are executed. If the file is positioned other than at its end, the closing operations specified by the Record Management Services are executed, but there is no ending label processing.

Output Files:

If label records are specified for the file, the labels are processed according to the standard label convention. Closing operations specified by the Record Management Services are executed. If label records are not specified for the file, label processing does not take place, but other closing operations specified by the Record Management Services are executed.

D Reel/Unit Removal

A Record Management Services defined technique is supplied to ensure that the current reel or unit is rewound when applicable, and that the operating system is notified that the reel or unit is logically removed from this run unit; however, the reel or unit can be accessed again, in its proper order of reels or units within the file, if a CLOSE statement without the REEL or UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

E File Lock

The file cannot be opened again during this execution of the run unit.

F Close Reel/Unit

Input Files:

The following operations take place:

(1) A reel/unit swap.

(2) The standard beginning reel/unit label procedure is executed.

The next executed READ statement for that file makes available the next data record on the new reel/unit.

Output Files and Input–Output Files:

The following operations take place:

(1) (For output files only) The standard ending reel/unit label procedure is executed.

(2) A reel/unit swap.

(3) The standard beginning reel/unit label procedure is executed.

For input-output files, the next executed READ statement that references that file makes the next logical data record on the next mass storage unit available. For output files, the next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

G Rewind

The current reel or analogous device is positioned at its physical beginning.

X Illegal

This is an illegal combination of a CLOSE option and a file category. The object program execution is terminated.

4. If the file is open when a STOP RUN statement is executed or when program execution terminates prematurely on an error condition, the file is closed automatically.

5. If the OPTIONAL phrase has been specified for the file in the FILE- CONTROL paragraph of the Environment Division and the file is not present, the standard end-of-file processing is not performed for that file.

6. If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

7. The WITH NO REWIND and FOR REMOVAL phrases will have no effect at object time if they do not apply to the storage medium on which the file resides.

8. Following the successful execution of a CLOSE statement without the REEL or UNIT phrase, the record area associated with a file-name is no longer available.

CLOSE (Sequential)

Continued

9. If an error occurs during the execution of a CLOSE statement issued without the UNIT or REEL phrase specified, the CLOSE will not occur. The value 98 is placed in the FILE STATUS data item (if one was specified) associated with the file.

10. If an error occurs during the execution of a CLOSE statement issued with the UNIT or REEL phrase specified, the CLOSE will not occur. The value 99 is placed in the FILE STATUS data item (if one was specified) associated with the file.

5.13 CLOSE Statement (Indexed and Relative)

Function

The CLOSE statement terminates the processing of files with optional lock.

General Format

```
CLOSE file-name-1 [WITH LOCK] [, file-name-2 [WITH LOCK]] ...
```

Syntax Rule

The files referenced in the CLOSE statement need not all have the same organization or access.

General Rules

1. A CLOSE statement can only be executed for a file in an open mode.
2. After the CLOSE ... WITH LOCK statement is executed, the file cannot be opened again during the current execution.
3. If a file is open when a STOP RUN statement is executed or when program execution terminates prematurely on an error condition, the file is closed automatically.
4. If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.
5. Following the successful execution of a CLOSE statement, the record area associated with file-name is no longer available.
6. If an error occurs during the execution of a CLOSE statement, the CLOSE will not occur. The value 98 is placed in the FILE STATUS data item (if one was specified) associated with the file.

5.14 COMPUTE Statement

Function

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items .

General Format

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...
        = arithmetic-expression [; ON SIZE ERROR imperative-statement]
```

Syntax Rule

Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric edited item.

General Rules

1. An arithmetic expression, consisting of a single identifier or literal, provides a method of setting the values of identifier–1, identifier–2, etc., equal to the value of the single identifier or literal. (See Section 5.5, Arithmetic Expressions.)
2. If more than one identifier is specified for the result of the operation that is preceding = (equal sign), the value of the arithmetic expression is computed, and then this value is stored as the new value of each of identifier–1, identifier–2, etc., in turn.
3. The COMPUTE statement allows you to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.
4. Arithmetic expressions must not contain non-integer exponents.

5.15 DELETE Statement (Indexed and Relative)

Function

The DELETE statement logically removes a record from a file on a directory device.

General Format

```
DELETE file-name RECORD [; INVALID KEY imperative-statement]
```

Syntax Rules

1. The INVALID KEY phrase must not be specified for a DELETE statement that references a file in sequential access mode.
2. The INVALID KEY phrase must be specified for a DELETE statement that references a file that is not in sequential access mode and for which an applicable USE procedure is not specified.

General Rules

1. The associated file must be open in the I-O mode when the DELETE statement is executed.
2. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The record that was accessed by that READ statement is logically removed from the file. If the last input-output statement executed for the associated file was not a successfully executed READ statement, the DELETE statement is not attempted, and the value of 93 is placed in the File Status data item, if any, associated with the file to indicate an unsuccessful DELETE operation.
3. When the INVALID KEY condition is recognized, actions are taken in the following order:

   a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition.

   b. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

   c. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

   When the INVALID KEY condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

4. For a relative file in random or dynamic access mode, that record identified by the contents of the RELATIVE KEY data item associated with file-name is logically removed from the file. An INVALID KEY condition can arise; the action taken is as follows:

   a. If the record specified by the contents of the RELATIVE KEY data item does not exist, the value 23 is placed in the FILE STATUS data item, if any, associated with the file to indicate an unsuccessful DELETE operation.

   b. If the contents of the RELATIVE KEY data item does not lie within the range of the key values corresponding to the allocated space for this file, a boundary violation exists. The value 24 is placed in the FILE STATUS data item, if any, associated with the file to indicate an unsuccessful DELETE operation.

5. For an indexed file accessed in random or dynamic mode, the record identified by the contents of the prime record key data item is logically removed from the file. If the specified record does not exist, a value of 23 (Invalid Key Condition) is placed in the FILE STATUS data item associated with file-name.

6. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

7. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

8. The current record pointer is not affected by the execution of a DELETE statement.

9. A DELETE statement will fail if it is executed for a record that is being simultaneously accessed by another process. The value 92 is placed in the FILE STATUS data item, if one was specified for the file.

10. If an unexplained error occurs during the execution of a DELETE statement, the execution will fail. A value of 30 is placed in the FILE STATUS data item, if one was specified for the file.

5.16 DISPLAY Statement

Function

The DISPLAY statement transfers low-volume data to an appropriate hardware device.

General Format

```
DISPLAY { identifier-1 } [ ,identifier-2 ] ...
        { literal-1    } [ ,literal-2    ]

    [UPON mnemonic-name][WITH NO ADVANCING]
```

Syntax Rules

1. The mnemonic-name is associated with a hardware device in the SPECIAL-NAMES paragraph in the Environment Division.
2. Each literal except ALL can be any figurative constant.
3. If the literal is numeric, it must be an unsigned integer.

General Rules

1. The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed, with no editing or conversion.
2. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.
3. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered.
4. When the WITH NO ADVANCING phrase is not specified, a line feed character is prefixed and a carriage return character is appended to the sending item. If the sending item exceeds the size of a line on the hardware device, the excess characters may appear on following line(s) or may be lost, depending on the device driver routine. Vertical and horizontal formatting characters may be placed in the sending item.
5. When the WITH NO ADVANCING phrase is specified, the carriage return character is not appended to the sending item. Depending on the device handler, the device will remain positioned on the same line and on the character position following the last character displayed. This is especially useful when typing prompting messages on the terminal.
6. If the UPON phrase is not used, the data is written on the user's standard display device.

5.17 DIVIDE Statement

Function

The DIVIDE statement divides one numeric data item into another and sets the value of data items to the quotient and remainder.

General Format

Format 1

```
DIVIDE { identifier-1 } INTO identifier-2 [ROUNDED]
       { literal-1    }

    [, identifier-3 [ROUNDED]] ...

    [ ; ON SIZE ERROR imperative-statement]
```

Format 2

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }      { literal-2    }

    [, identifier-4 [ROUNDED]] ...

    [ ; ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }   { literal-2    }

    [, identifier-4 [ROUNDED]] ...

    [ ; ON SIZE ERROR imperative-statement]
```

Format 4

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }      { literal-2    }

    REMAINDER identifier-4 [; ON SIZE ERROR imperative-statement]
```

(continued on next page)

Format 5

```
DIVIDE { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ROUNDED]
       { literal-1    }    { literal-2    }

REMAINDER identifier-4 [ ;ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal

3. The composite of operands, which is the hypothetical data item resulting from the superimposition of all receiving data items (except the REMAINDER data item) of a given statement aligned on their decimal points, must not contain more than 18 digits.

General Rules

1. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient; the same applies for identifier-1 or literal-1 and identifier-3, etc.

2. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2, and the result is stored in identifier-3, identifier-4, etc.

3. When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2, and the result is stored in identifier-3, identifier-4, etc.

4. Formats 4 and 5 are used when a remainder from the division operation is desired, normally identifier-4. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If identifier-3 is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field that contains the quotient of the DIVIDE statement, truncated rather than rounded.

5. In Formats 4 and 5, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described above. Appropriate decimal alignment and truncation (not rounding) will be performed for the content of the data item referenced by identifier-4, as needed.

6. When the ON SIZE ERROR phrase is used in Formats 4 and 5, the following rules apply:

   a. If the size error occurs on the quotient, no remainder calculation is meaningful. Therefore, the contents of the data items referenced by both identifier-3 and identifier-4 will remain unchanged.

   b. If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 remain unchanged. However, as with other instances of multiple results of arithmetic statements, you will have to do your own analysis to recognize which situation has occurred.

5.18 EXIT Statement

Function

The EXIT statement provides a common end point for a series of procedures, or marks the logical end of a called program.

General Format

```
EXIT [PROGRAM]
```

Syntax Rules

1. The EXIT statement without the PROGRAM phrase must appear only in a sentence by itself and comprise the only sentence in the paragraph.
2. If an EXIT PROGRAM statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules

1. An EXIT statement without the optional word PROGRAM serves only to enable you to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.
2. Execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. If the EXIT PROGRAM statement is executed in a program that is not under the control of a calling program, the EXIT PROGRAM statement causes execution of the program to continue with the next executable statement.

5.19 GO TO Statement

Function

The GO TO statement transfers control from one part of the Procedure Division to another.

General Format

Format 1

```
GO TO [procedure-name-1]
```

Format 2

```
GO TO procedure-name-1 [, procedure-name-2]..., procedure-name-n
      DEPENDING ON identifier
```

Syntax Rules

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.
2. When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format 1 GO TO statement.
3. A Format 1 GO TO statement without procedure-name-1 can only appear in a single statement paragraph.
4. If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules

1. When a GO TO statement represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.
2. If procedure-name-1 is not specified in Format 1, an ALTER statement referring to this GO TO statement must be executed prior to the execution of this GO TO statement.
3. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1, procedure-name-2, etc., depending on whether the value of the identifier is 1, 2, ..., n. If the value of the identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

5.20 IF Statement

Function

The IF statement causes a condition to be evaluated. The subsequent flow of control of the object program depends on whether the value of the condition is true or false.

General Format

```
IF condition;  { statement-1          }  [ ; ELSE statement-2          ]
               { NEXT SENTENCE        }  [ ; ELSE NEXT SENTENCE      ]
```

Syntax Rules

1. Statement-1 and statement-2 represent either an imperative statement or a conditional statement, and either can be followed by a conditional statement.
2. The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the terminal period of the sentence.

General Rules

1. When an IF statement is executed, the following transfers of control occur:
   a. If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not contain a procedure branching or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.
   b. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.
   c. If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching statement or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching or conditional statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

d. If the condition is false, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

2. Statement-1 and/or statement-2 can contain an IF statement. In this case the IF statement is said to be nested.

   IF statements within IF statements can be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

5.21 INSPECT Statement

Function

The INSPECT statement is used to count (Format 1), replace (Format 2), or count and replace (Format 3) occurrences of single characters in a data item.

General Format

Format 1

```
INSPECT identifier-1 TALLYING

{ , identifier-2 FOR { , { ALL      } identifier-3 } [ { BEFORE } INITIAL { identifier-4 } ] ... } ...
                     { , { LEADING  } literal-1    }   { AFTER  }         { literal-2    }
                     {   CHARACTERS }
```

Format 2

```
INSPECT identifier-1 REPLACING

{ CHARACTERS BY { identifier-6 } [ { BEFORE } INITIAL { identifier-7 } ]                                                                                  }
{               { literal-4    }   { AFTER  }         { literal-5    } ]                                                                                  }
{ { , { ALL     } } { , { identifier-5 } BY { identifier-6 } [ { BEFORE } INITIAL { identifier-7 } ] } ... } ...                                          }
{ {   { LEADING } }   { { literal-3    }    { literal-4    }   { AFTER  }         { literal-5    } ]   }     }     }
{ {   { FIRST   } }                                                                                   }     }     }
```

Format 3

```
INSPECT identifier-1 TALLYING

{ , identifier-2 FOR { , { ALL      } { identifier-3 } [ { BEFORE } INITIAL { identifier-4 } ] } ... } ...
                     { , { LEADING  } { literal-1    }   { AFTER  }         { literal-2    } ]   }     }
                     {   CHARACTERS }

REPLACING

{ CHARACTERS BY { identifier-6 } [ { BEFORE } INITIAL { identifier-7 } ]                                                                                  }
{               { literal-4    }   { AFTER  }         { literal-5    } ]                                                                                  }
{ { , { ALL     } } { , { identifier-5 } BY { identifier-6 } [ { BEFORE } INITIAL { identifier-7 } ] } ... } ...                                          }
{ {   { LEADING } }   { { literal-3    }    { literal-4    }   { AFTER  }         { literal-5    } ]   }     }     }
{ {   { FIRST   } }                                                                                   }     }     }
```

Syntax Rules

All Formats

1. Identifier-1 must reference either a group item or any category of elementary item described (either implicitly or explicitly) as USAGE IS DISPLAY.
2. Identifier-3...identifier-n must reference either an elementary alphabetic, alphanumeric, or numeric item described (either implicitly or explicitly) as USAGE IS DISPLAY.
3. Each literal must be nonnumeric and can be any figurative constant except ALL.
4. Literal-1, literal-2, literal-3, literal-4, and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, can be any length except as specifically restricted by syntax and general rules.

Formats 1 and 3 only

5. Identifier-2 must reference an elementary numeric data item.
6. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit 1-character data item.

Formats 2 and 3 only

7. The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.
8. When the CHARACTERS phrase is used, literal-4, literal-5, or the size of the data item referenced by identifier-6, identifier-7, must be one character in length.
9. When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

General Rules

All Formats

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position, as described in General Rules 4 through 6.

2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are treated as follows:

   a. If any of the identifiers is described as alphanumeric, the INSPECT statement treats the contents of each identifier as a character-string.

   b. If any of the identifiers is described as unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (see General Rule 2a) and the INSPECT statement had been written to reference the redefined data item.

   c. If any of the identifiers is described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in General Rule 2b had been applied. (See Section 5.22, MOVE Statement).

3. In General Rules 4 through 11, all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Formats 2 and 3).

5. The comparison operation to determine the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced occurs as follows:

   a. The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1, literal-3 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal, character for character.

b. If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

c. Whenever a match occurs, tallying and/or replacing takes place as described in General Rules 8 through 10. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1, literal-3.

d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

e. If the CHARACTERS phrase is specified, an implied 1-character operand participates in the cycle described in paragraphs 5a through 5d above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in General Rule 5 is affected by the BEFORE and AFTER phrases as follows:

   a. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in General Rule 5.

   b. If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles that involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in General Rule 5 is begun.

      If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is

considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase can participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1. The comparison begins from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in General Rule 5 is begun.

   If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Format 1

7. The contents of the data item referenced by identifier-2 are not initialized by the execution of the INSPECT statement.

8. The rules for tallying are as follows:

   a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

   b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

   c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one for each character matched, in the sense of General Rule 5e, within the contents of the data item referenced by identifier-1.

Format 2

9. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

10. The rules for replacement are as follows:

    a. When the CHARACTERS phrase is specified, each character matched, in the sense of General Rule 5e, in the contents of the data item referenced by identifier-1 is replaced by literal-4.

    b. When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

    c. When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

    d. When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

Format 3

11. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written, with one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the other statement being a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement, and the general rules given for matching and replacing apply to the Format 2 statement.

Examples

Following are six examples of the INSPECT statement:

```
INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A"
count-1 FOR LEADING "A" BEFORE INITIAL "L".

    Where word = LARGE, count = 1, count-1 = 0.
    Where word = ANALYST, count = 0, count-1 = 1.

INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING "A" BY
"E" AFTER INITIAL "L".

    Where word = CALLAR, count = 2, word = CALLAR.
    Where word = SALAMI, count = 1, word = SALEMI.
    Where word = LATTER, count = 1, word = LETTER.

INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

    Where word = ARXAX, word = GRXAX.
    Where word = HANDAX, word = HGNDGX.

INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J"
REPLACING ALL "A" BY "B".

    Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
    Where word = JACK, count = 3, word = JBCK.
    Where word = JUJMAB, count = 5, word = JUJMBB.

INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

    WORD BEFORE: 1    2 X Z A B C D
    WORD AFTER:   B B BB B A B C D

INSPECT word REPLACING ALL "X" BY "Y", "B" BY "Z", "W" BY "Q",
AFTER INITIAL "R".

    Where word = RXXBQWY, word = RYYZQQY.
    Where word = YZACDWBR, word = YZACDWBR.
    Where word = RAWRXEB, word = RAQRYEZ.
```

5.22 MOVE Statement

Function

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

General Format

Format 1

```
MOVE { identifier-1 } TO identifier-2 [,identifier-3]...
     { literal      }
```

Format 2

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

Syntax Rules

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3, ..., represent the receiving area.
2. CORR is an abbreviation for CORRESPONDING.
3. When the CORRESPONDING phrase is used, both identifiers must be group items.
4. An index data item cannot appear as an operand of a MOVE statement.

General Rules

1. If the CORRESPONDING phrase is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given in Section 5.7.3, CORRESPONDING Phrase. The results are the same as if you had referred to each pair of corresponding identifiers in separate MOVE statements.
2. The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, ... The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, ..., is evaluated immediately before the data is moved to the respective data item.

Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the receiving operands. Consider the following statement.

```
MOVE A (B) TO B, C (B)
```

The result of this statement is equivalent to:

```
MOVE A (B) TO temp
MOVE temp TO B
MOVE temp TO C (B)
```

where temp is an intermediate result item provided by the compiler.

3. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the numeric category, and nonnumeric literals belong to the alphanumeric category, while the figurative constant ZERO belongs to the numeric category. The figurative constant SPACE belongs to the alphabetic category. All other figurative constants belong to the alphanumeric category.

   The following rules apply to an elementary move between these categories:

   a. The figurative constant SPACE, a numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

   c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

   d. All other elementary moves are legal and are performed according to the rules given in General Rule 4.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, as does as any editing specified for the receiving data item:

   a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling take place as defined under Standard Alignment Rules, Section 4.2.7. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign

will not be moved; if the operational sign occupied a separate character position, that character will not be moved, and the size of the sending item is considered to be one less than its actual size (in terms of standard data format characters).

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling take place as defined under the Standard Alignment Rules, Section 4.2.7, except where zeros are replaced because of editing requirements.

   1. When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. (See Section 4.3.9, SIGN Clause.) Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

   2. When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

   3. When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

c. When a receiving field is described as alphabetic, justification and any necessary space-filling take place as defined under the Standard Alignment Rules, Section 4.2.7. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

5. Any non-elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area is filled without consideration for the individual elementary or group items contained within either the sending or receiving area.

6. The following table summarizes the permissible types of MOVE statements. References after slash marks (for example, /4c) refer to the applicable general rule for MOVE.

Table 5-4: Permissible MOVE Statements

Category of Sending Data Item	Category of Receiving Data Item		
	Alphabetic	**Alphanumeric Edited Alphanumeric**	**Numeric Integer Numeric Non-Integer Numeric Edited**
Alphabetic	Yes/4c	Yes/4a	No/3a
Alphanumeric	Yes/4c	Yes/4a	Yes/4b
Alphanumeric Edited	Yes/4c	Yes/4a	No/3a
Numeric Integer	No/3b	Yes/4a	Yes/4b
Numeric Non-Integer	No/3b	No/3c	Yes/4b
Numeric Edited	No/3b	Yes/4a	No/3a

5.23 MULTIPLY Statement

Function

The MULTIPLY statement multiplies numeric data items and sets the values of data items equal to the results.

General Format

Format 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1    }

    [, identifier-3 [ROUNDED]] ...

    [; ON SIZE ERROR imperative-statement]
```

Format 2

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ROUNDED]
         { literal-1    }    { literal-1    }

    [,identifier-4 [ROUNDED]] ...

[; ON SIZE ERROR imperative-statement ]
```

Syntax Rules

1. Each identifier must refer to a numeric elementary item, except that in Format 2 the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.
2. Each literal must be a numeric literal.
3. The composite of operands, which is that hypothetical data item resulting from the superimposition of all receiving data items of a given statement aligned on their decimal points, must not contain more than eighteen (18) digits.

General Rules

1. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product; the same result occurs for identifier-1 or literal-1 and identifier-3, etc.
2. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

5.24 OPEN Statement (Sequential)

Function

The OPEN statement initiates the processing of files. It also performs checking and/or label writing and other input-output operations.

General Format

```
OPEN { INPUT file-name-1 [WITH NO REWIND] [, file-name-2 [WITH NO REWIND]] ...
     { OUTPUT file-name-3 [WITH NO REWIND] [, file-name-4 [WITH NO REWIND]] ...
     {
     { I-O file-name-5 [, file-name-6] ...
     { EXTEND file-name-7 [, file-name-8] ...
                                                                              } ...
```

Syntax Rules

1. The NO REWIND phrase can be used only for sequential files.
2. The I-O phrase can be used only for files on directory devices.
3. The EXTEND phrase can be used only for sequential files.
4. The EXTEND phrase must not be specified for files on multiple file reels.
5. The files referenced in the OPEN statement need not all have the same organization or access.

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file's being in an open mode.
2. The successful execution of an OPEN statement makes the associated record area available to the program.
3. Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that explicitly or implicitly refers to that file.
4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 5–5, an X indicates that the specified statement, used in the sequential access mode, can be used with the sequential file organization and open mode given at the top of the column.

Table 5-5: Permissible Input-Output Statements for Sequential Files

Statement	Open Mode			
	Input	Output	Input-Output	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

5. A file can be opened with the INPUT, OUTPUT, EXTEND and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, without the REEL, UNIT, or LOCK phrase, for that file.
6. Execution of the OPEN statement does not obtain or release the first data record.
7. If label records are specified for the file, the beginning labels are processed as follows:
   a. When the INPUT phrase is specified, execution of the OPEN statement causes the labels to be checked in accordance with the Record Management Services conventions for input label checking.
   b. When the OUTPUT phrase is specified, execution of the OPEN statement causes the labels to be written in accordance with the Record Management Services conventions for output label writing.

   The behavior of the OPEN statement when label records are specified but not present, or when label records are not specified but are present, is undefined.
8. The file-description-entry for file-name-1, file-name-2, file-name-5, file-name-6, file-name-7, or file-name-8 must be equivalent to that used when the file was created.
9. If an input file is designated with the OPTIONAL clause in its SELECT statement, the object program causes an interrogation for the presence or absence of this file when the OPEN statement is executed. If the file is not present, the first READ statement for this file causes the AT END condition to occur.
10. The NO REWIND phrase can be used only with sequential single reel/unit files.
11. The WITH NO REWIND phrase is ignored if it does not apply to the storage medium on which the file resides.

12. If the storage medium for the file permits rewinding, the following rules apply:

    a. When neither the EXTEND nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

    b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file must be already positioned at its beginning prior to execution of the OPEN statement.

13. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set so that the next executed READ statement for the file will result in an AT END condition.

14. When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing that file will add records to the file as though the file had been opened with the OUTPUT phrase.

15. When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. Beginning file labels are processed only in the case of a single reel/unit file.

    b. Beginning reel/unit labels on the last existing reel/unit are processed as though the file was being opened with the INPUT phrase.

    c. Existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.

    d. Processing then proceeds as though the file had been opened with the OUTPUT phrase.

16. The I-O phrase permits the opening of a directory file for both input and output operations. Because this phrase implies the existence of the file, it cannot be used if the directory file is being created.

17. When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

    a. Labels are checked in accordance with the specified conventions for input-output label checking.

b. New labels are written in accordance with the standard conventions for input-output label writing.

18. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

19. If the execution of an OPEN statement fails and a USE procedure is specified for the file (either explicitly or implicitly), the USE procedure is executed. Execution of an OPEN statement fails for any of the following reasons:

NOTE:

The value in parentheses after each of the following statements is the value that is placed in the FILE STATUS data item, if one was specified for the file.

a. An OPEN statement executed for a file that is already opened for exclusive access by another task. (91)

b. An OPEN statement executed for a device that has no available file space. (95)

c. An OPEN statement executed for a file that shares buffer space with an already opened file. (96)

d. An OPEN statement executed for a file that cannot be found on its associated I/O device. (97)

5.25 OPEN Statement (Indexed & Relative)

Function

The OPEN statement initiates the processing of files.

General Format

```
                    ⎧ INPUT  file-name-1 [, file-name-2] ... ⎫
OPEN                ⎪ OUTPUT file-name-3 [, file-name-4] ... ⎬ ...
                    ⎩ I-O    file-name-5 [, file-name-6] ... ⎭
```

Syntax Rule

The files referenced in the OPEN statement need not all have the same organization or access.

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file's being in an open mode.
2. The successful execution of the OPEN statement makes the associated record area available to the program.
3. Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that explicitly or implicitly references that file.
4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 5-6, an X indicates that the specified statement used in the access mode given for that row can be used with indexed or relative file organizations and the open mode given at the top of the column.

Table 5-6: Permissible Input-Output Statements for Indexed and Relative Files

File Access Mode	Open Mode			
	Statement	Input	Output	Input-Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	READ NEXT	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

5. A file can be opened with the INPUT, OUTPUT, and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, without the LOCK phrase, for that file.

6. Execution of the OPEN statement does not obtain or release the first data record.

7. The file-description-entry for file-name-1, file-name-2, file-name-5, or file-name-6 must be equivalent to that used when this file was created.

8. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. For indexed files, the prime record key is established as the key of reference and is used to determine the first record to be accessed. If no records exist in the file, the next executed sequentially accessed READ statement for the file results in an AT END condition.

9. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

10. If the execution of an OPEN statement fails and a USE procedure is specified for the file (either explicitly or implicitly), the USE procedure is executed. The execution of an OPEN statement will fail for any of the following reasons:

NOTE:

The value in parentheses after each of the following statements is the value that is placed in the FILE STATUS data item, if one was specified for the file.

a. An OPEN statement executed for a file that is already opened for exclusive access by another task. (91)

b. An OPEN statement executed for a device that has no available file space. (95)

c. An OPEN statement executed for a file that shares buffer space with an already opened file. (96)

d. An OPEN statement executed for a file that cannot be found on its associated I/O device. (97)

5.26 PERFORM Statement

Function

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

General Format

Format 1

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                         { THRU    }
```

Format 2

```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ]
                         { THRU    }
{ identifier-1 } TIMES
{ integer      }
```

Format 3

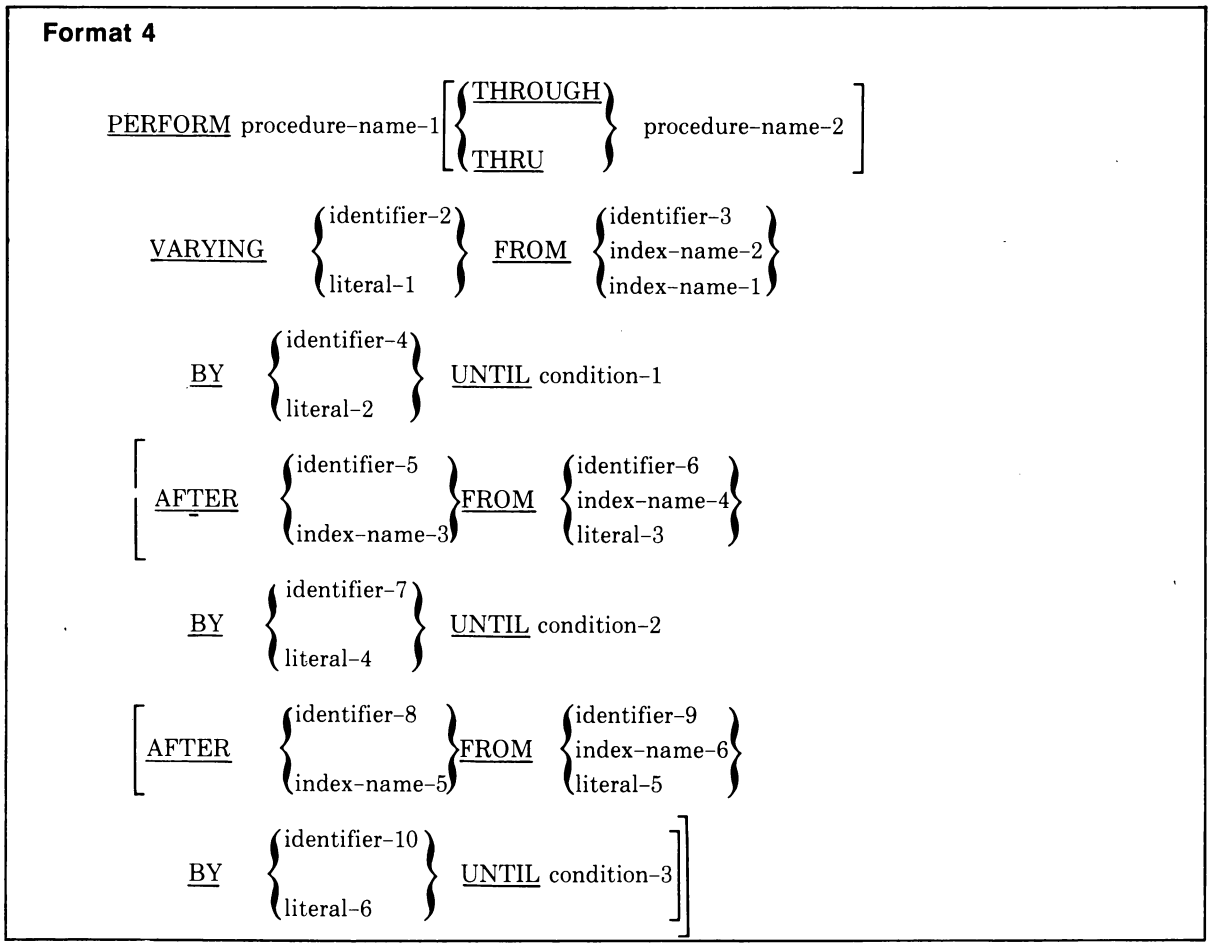
```
PERFORM procedure-name-1 [ { THROUGH } procedure-name-2 ] UNTIL condition-1
                         { THRU    }
```

(continued on next page)

PERFORM

Continued

Format 4



Syntax Rules

1. Each identifier represents a numeric elementary item described in the Data Division. In Format 2, identifier-1 must be described as a numeric integer.
2. Each literal represents a numeric literal.
3. The words THRU and THROUGH are equivalent.
4. If an index-name is specified in the VARYING or AFTER phrase, then:
   a. The identifier in the associated FROM and BY phrases must be an integer data item.
   b. The literal in the associated FROM phrase must be a positive integer.
   c. The literal in the associated BY phrase must be a non-zero integer.

5. If an index-name is specified in the FROM phrase, then:

   a. The identifier in the associated VARYING or AFTER phrase must be an integer data item.

   b. The identifier in the associated BY phrase must be an integer data item.

   c. The literal in the associated BY phrase must be an integer.

6. Literal in the BY phrase must not be 0.

7. Condition-1, condition-2, condition-3 can be any conditional expression as described in Section 5.6, Conditional Expressions.

8. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declarative section of the program, then both must be procedure-names in the same declarative section.

General Rules

1. The data items referenced by identifier-4, identifier-7, and identifier-10 must not have a zero value.

2. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, then the data item referenced by the identifier must have a positive value.

3. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1 (except as indicated in General Rules 6b, 6c, and 6d). This transfer of control occurs only once for each execution of a PERFORM statement. Where a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

   a. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return is after the last statement of procedure-name-1.

   b. If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

   c. If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

   d. If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

4. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements can occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 can be the name of a paragraph consisting of the EXIT statement to where all of these paths must lead.

5. If control passes to these procedures other than by a PERFORM statement, control passes through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

6. The PERFORM statements operate as follows, with Rule 5 above applying to all formats:

   a. Format 1 is the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is executed once and control then passes to the next executable statement following the PERFORM statement.

   b. Format 2 is the PERFORM...TIMES. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to 0 or is negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the next executable statement following the PERFORM statement.

      During execution of the PERFORM statement, references to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

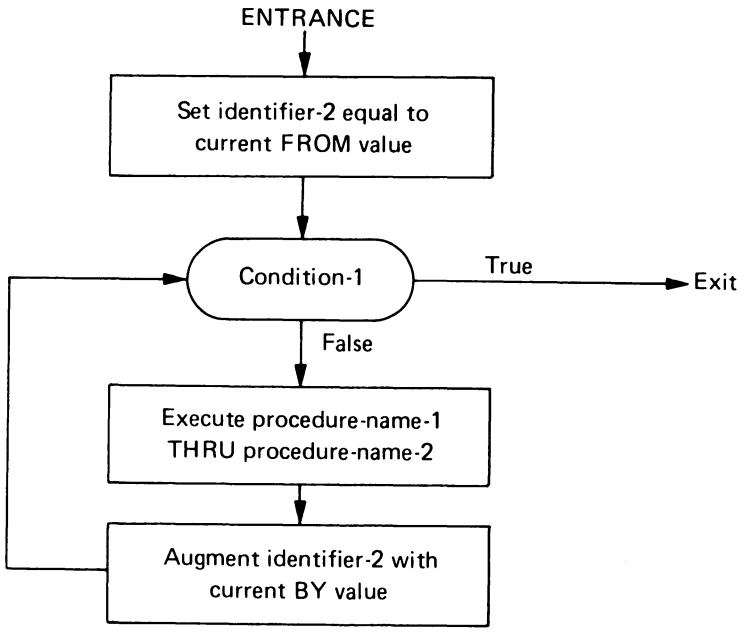
   c. Format 3 is the PERFORM...UNTIL. The specified procedures are performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the next executable statement after the PERFORM statement. If the condition is true when the PERFORM statement is entered, no transfer to procedure-name-1 takes place, and control is passed to the next executable statement following the PERFORM statement.

d. Format 4 is the PERFORM...VARYING. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER and FROM (current value) phrases also refers to index-names. When index-name appears in a VARYING and/or AFTER phrase, it is initialized and subsequently augmented (as described below) according to the rules of the SET statement. When index-name appears in the FROM, VARYING or AFTER phrase, it is initialized according to the rules of the SET statement; subsequent augmentation is described below.

In Format 4, when one identifier is varied, identifier-2 is set to the value of literal-1 or the current value of identifier-3 at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The value of identifier-2 is augmented by the specified increment or decrement value (the value of identifier-4 or literal-2) and condition-1 is evaluated again. The cycle continues until this condition is true, at which point, control is transferred to the next executable statement following the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the next executable statement following the PERFORM statement.

The following flowchart shows the VARYING phrase of a PERFORM statement having one condition:

Figure 5-1: VARYING Phrase of PERFORM with One Condition

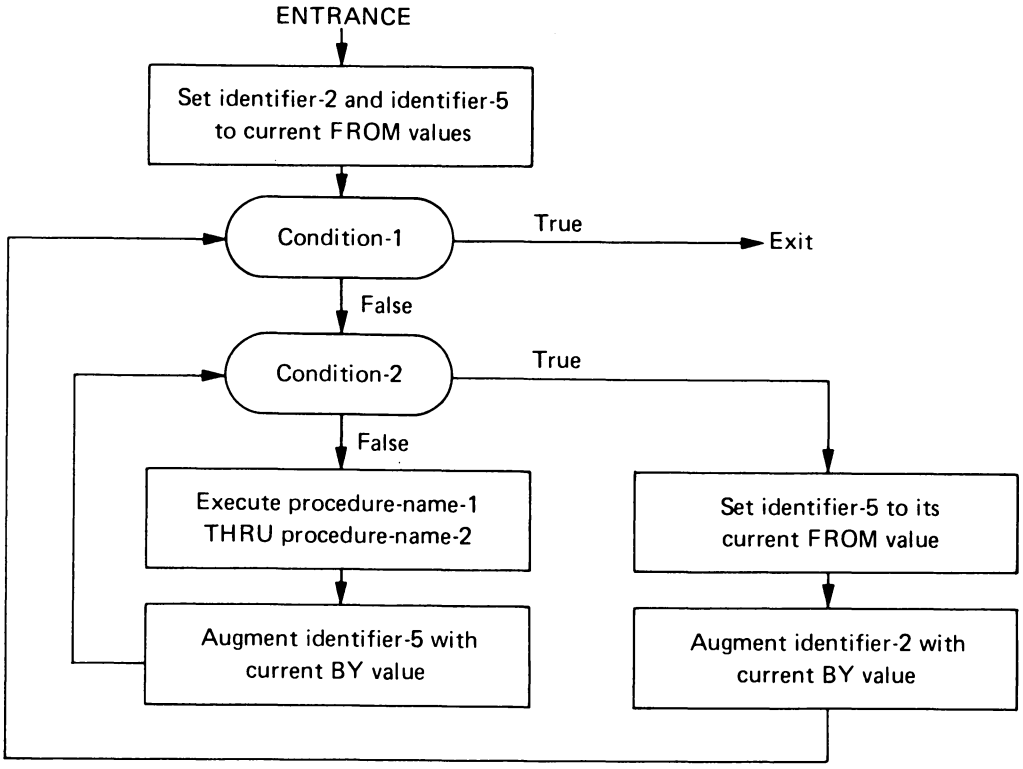


In Format 4, when two identifiers are varied, identifier-2 and identifier-5 are set to the current value of identifier-3 and identifier-6, respectively. After the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the next executable statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 are executed once, then identifier-5 is augmented by identifier-7 or literal-4 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When condition-2 is true, identifier-5 is set to the value of literal-3 or the current value of identifier-6, identifier-2 is augmented by identifier-4 and condition-1 is reevaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true.

During the execution of the procedures associated with the PERFORM statement, any change to the VARYING variable (identifier-2 and index-name-1), the BY variable (identifier-4), the AFTER variable (identifier-5 and index-name-3), or the FROM variable (identifier-3 and index-name-2) will be taken into consideration and will affect the operation of the PERFORM statement.

The following flowchart shows the VARYING phrase of a PERFORM statement having two conditions:

Figure 5-2: VARYING Phrase of PERFORM with Two Conditions



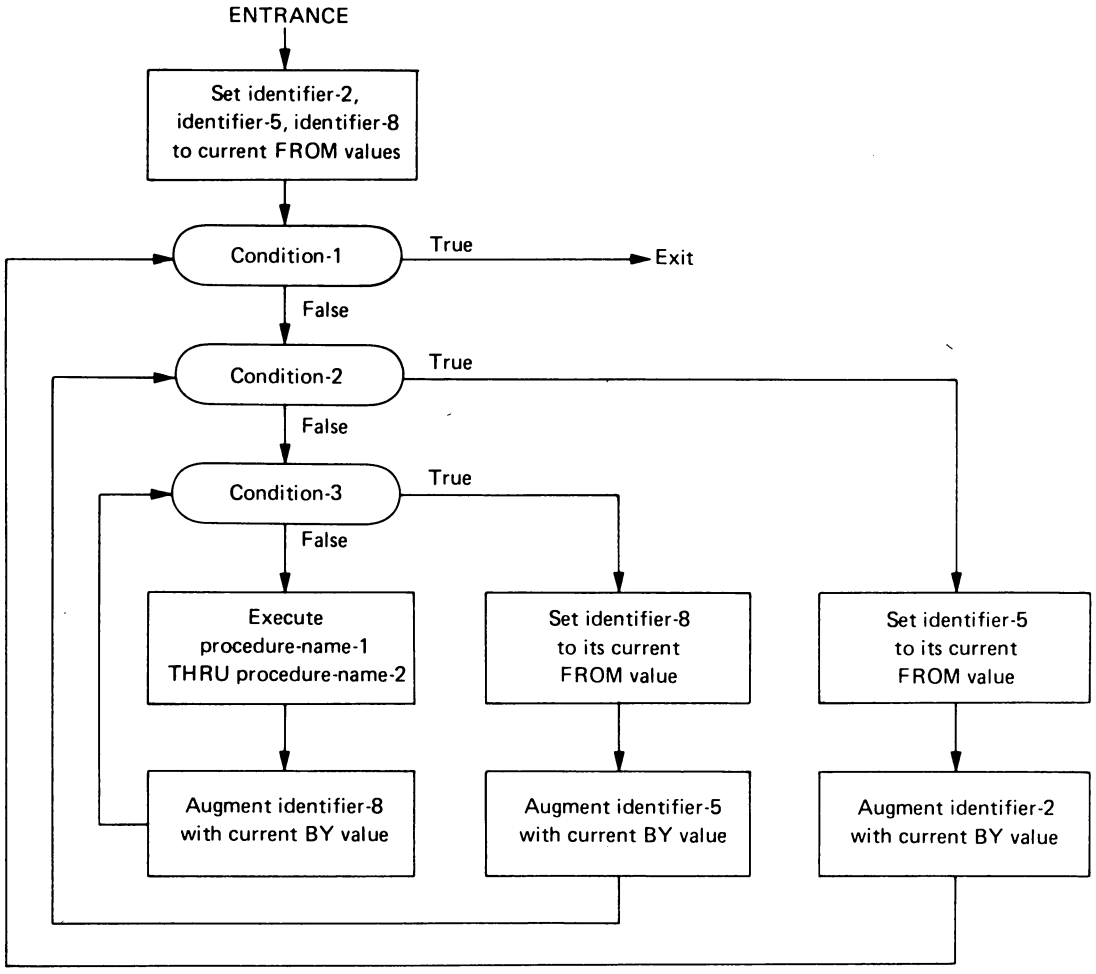
At the termination of the PERFORM statement, identifier–5 contains the current value of identifier–6. Identifier–2 has a value that exceeds the last used setting by an increment or decrement value, unless condition–1 was true when the PERFORM statement was entered. Then, identifier–2 contains the current value of identifier–3.

When two identifiers are varied, identifier–5 goes through a complete cycle (FROM, BY, UNTIL) each time identifier–2 is varied.

For three identifiers, the mechanism is the same as for two identifiers except that identifier–8 goes through a complete cycle each time that identifier–5 is augmented by identifier–7 or literal–4, which in turn goes through a complete cycle each time identifier–2 is varied.

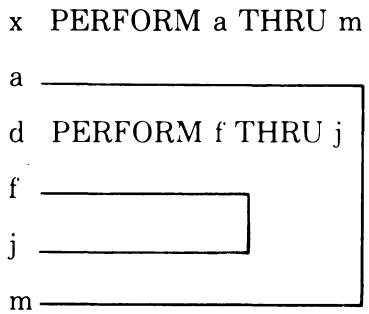
The following flowchart shows the VARYING phrase of a PERFORM statement having three conditions:

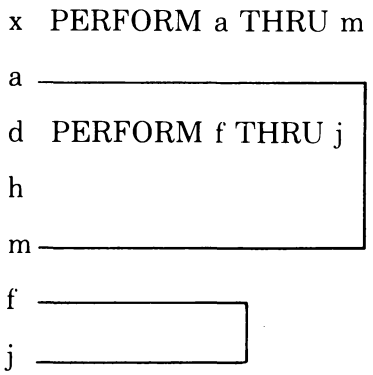
Figure 5–3: VARYING Phrase of PERFORM with Three Conditions

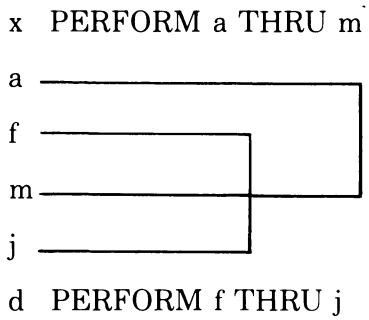


After the completion of a Format 4 PERFORM statement, identifier-5 and identifier-8 contain the current value of identifier-6 and identifier-9, respectively. Identifier-2 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-2 contains the current value of identifier-3.

7. If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements cannot have a common exit. See the illustrations below.







8. A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

   a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

   b. Sections and/or paragraphs wholly contained in a single independent segment.

9. A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

   a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

   b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

5.27 Read Statement (Sequential)

Function

The READ statement makes available the next logical record from a file.

General Format

```
READ file-name RECORD [INTO identifier] [; AT END imperative-statement]
```

Syntax Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be allocated to the same storage area.
2. The AT END phrase must be specified if no applicable USE procedure is specified for file-name.

General Rules

1. The associated file must be open in the INPUT or I-O mode at the time this statement is executed.
2. The record to be made available by the READ statement is determined as follows:
   a. If the current record pointer was positioned by the execution of the OPEN statement, the record pointed to by the current record pointer is made available.
   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.
3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.
4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged because a record is available to the object program prior to the execution of any statement following the READ statement.
5. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

7. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

8. If, at the time of execution of a READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful. The FILE STATUS data item, if any, associated with the file is set to one of the values detailed in General Rules 11 and 14.

9. If the end of a reel or unit is recognized during execution of a READ statement and the logical end of the file has not been reached, the following operations are executed:

   a. The standard ending reel/unit label procedure.

   b. A reel/unit swap.

   c. The standard beginning reel/unit label procedure.

   d. The first data record of the new reel/unit is made available.

10. If a file described with the OPTIONAL clause is not present at the time the file is opened, then, at the time of execution of the first READ statement for the file, the AT END condition occurs and the execution of the READ statement is unsuccessful. The standard end-of-file procedures are not performed. Execution of the program then proceeds as specified in General Rule 11 a, b, and c.

11. If, at the time of the execution of a READ statement, no next logical record exists in the file, the AT END condition occurs and the execution of the READ statement is considered unsuccessful.

    When the AT END condition is recognized, the following actions are taken in the specified order:

    a. The value 10 is placed into the FILE STATUS data item, if any, associated with this file to indicate an AT END condition.

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file. That USE procedure is executed.

12. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

13. When the AT END condition has been recognized, a READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

14. A Format 1 or Format 2 READ statement that fails for an undetermined reason will cause the value 30 to be placed in the FILE STATUS data item, if one was specified for the file.

5.28 READ Statement (Relative)

Function

For sequential access, the READ statement makes available the next logical record from a file on a directory device. For random access, the READ statement makes available a specified record from a file on a directory device. For dynamic access, two forms of the READ statement are available, allowing the next logical record or a specified logical record to be made available.

General Format

Format 1

READ file-name[NEXT]RECORD[INTO identifier][; AT END imperative-statement]

Format 2

READ file-name RECORD[INTO identifier][; INVALID KEY imperative-statement]

Syntax Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be allocated to the same storage area.
2. Format 1 must be used for all files in sequential access mode.
3. Format 1 with the NEXT phrase specified must be used for files in dynamic access mode when records are to be retrieved sequentially.
4. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.
5. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

General Rules

1. The associated files must be open in the INPUT or I-O mode at the time this statement is executed.
2. The record to be made available by a Format 1 READ statement is determined by updating the current record pointer to point to the next existing record in the file.
3. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged because a record is available to the object program prior to the execution of any statement following the READ statement.

4. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

5. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

6. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

7. If, at the time of execution of a Format 1 READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful. The FILE STATUS data item, if any, associated with the file is set to one of the values described in General Rules 13, 14, and 15.

8. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs and the execution of the READ statement is considered unsuccessful.

   When the AT END condition is recognized, the following actions are taken in the specified order:

   a. The value 10 is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.

   b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

   c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file. That USE procedure is executed.

9. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

10. When the AT END condition has been recognized, a Format 1 READ statement for that file must not be executed without first executing one of the following:

    a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful START statement for that file.

    c. A successful Format 2 READ statement for that file.

11. For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from the file (as described in General Rule 2).

12. If the RELATIVE KEY clause is specified, the execution of a Format 1 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

13. The execution of a Format 2 READ statement sets the current record pointer to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY clause for the file. An INVALID KEY condition can arise; the READ is considered unsuccessful and the following actions are taken:

    a. If the record specified by the contents of the RELATIVE KEY data item does not exist, the value 23 is placed in the FILE STATUS data item, if any, associated with this file to indicate an unsuccessful READ operation.

    b. If the contents of the RELATIVE KEY data item do not lie within the range of the key values corresponding to the allocated space for this file, a boundary violation exists. The value 24 is placed in the FILE STATUS data item, if any, associated with the file to indicate an unsuccessful READ operation.

    c. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any USE procedure specified for this file is not executed.

    d. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

    When the INVALID KEY condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

14. A Format 1 or Format 2 READ statement issued to a file that is being simultaneously accessed by another task can fail. The value 92 is placed into the FILE STATUS data item, if one was specified for the file.

15. A Format 1 or Format 2 READ statement that fails for an undetermined reason will cause the value 30 to be placed in the FILE STATUS data item, if one was specified for the file.

5.29 READ Statement (Indexed)

Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file. For dynamic access, both sequential and random access can be used to obtain the next logical record in a file.

General Format

Format 1

```
READ file-name [NEXT] RECORD [INTO identifier]
[; AT END imperative-statement]
```

Format 1

```
READ file-name RECORD [INTO identifier]
[; KEY IS data-name]
[; INVALID KEY imperative-statement]
```

Syntax Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes, which are indicated by their record descriptions. The storage area associated with identifier and the storage area that is the record area associated with file-name must not be the same.
2. Data-name must be the name of a data item specified as a record key associated with file-name.
3. Data-name can be qualified.
4. Use Format 1 for all files in sequential access mode.
5. Use Format 1 with the NEXT phrase specified for files in dynamic access mode when records are to be retrieved sequentially.
6. Use Format 2 for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.
7. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

General Rules

1. The associated file must be open in the INPUT or I-O mode at the time this statement is executed.
2. The record to be made available by a Format 1 READ statement is determined as follows:
   a. The record pointed to by the current record pointer is made available, provided that the current record pointer has been positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, which may have been caused by the deletion of the record or a change in an alternate record key, the current record pointer is updated to point to the next existing record within the established key of reference. Then, that record is then made available.
   b. If the current record pointer has been positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file within the established key of reference. Then, that record is made available.
3. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available to the object program prior to the execution of any statement following the READ statement.
4. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.
5. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.
6. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.
7. If, at the time of execution of a Format 1 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful. The FILE STATUS data item, if any, associated with the file is set to one of the values described in General Rules 15, 16, or 17.

8. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

   When the AT END condition is recognized, the following actions are taken in the specified order:

   a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.

   b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative statement. Any USE procedure specified for this file is not executed.

   c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file, and that procedure is executed.

9. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined. For indexed files the key or reference is also undefined.

10. When the AT END condition has been recognized, a Format 1 READ statement for that file must not be executed without first executing one of the following:

    a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful START statement for that file.

    c. A successful Format 2 READ statement for that file.

11. For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in General Rule 2.

12. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key that is the key of reference are made available in the same order in which they are released by execution of WRITE statements or by execution of REWRITE statements that create such duplicate values.

13. If the KEY phrase is specified in a Format 2 READ statement for an indexed file, data-name is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for it.

14. If the KEY phrase is not specified in a Format 2 READ statement, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key is established for the file.

15. Execution of a Format 2 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file. When the first record having an equal value is found, the current record pointer is positioned to this record, making it available for processing. If no record containing the key value is found, an INVALID KEY condition exists.

    When the INVALID KEY condition is recognized, actions are taken in the following order:

    a. The value 23 is placed into the FILE STATUS data item (if specified for this file) to indicate an INVALID KEY condition.

    b. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any USE procedure specified for this file is not executed.

    c. If the INVALID KEY phrase is not specified, but a USE procedure is specified for this file, either explicitly or implicitly, that procedure is executed.

    When the INVALID KEY condition occurs, execution of the input-output statement that recognized the condition is unsuccessful, and the file is not affected.

16. A Format 1 or Format 2 READ statement issued to a record that is being simultaneously accessed by another task can fail. The value 92 is placed into the FILE STATUS data item, if one was specified for the file.

17. A Format 1 or Format 2 READ statement that fails for an undetermined reason will cause a value of 30 to be placed in the FILE STATUS data item, if one was specified for the file.

5.30 REWRITE Statement (Sequential)

Function

The REWRITE statement logically replaces a record existing in a file on a directory device.

General Format

```
REWRITE record-name [FROM identifier]
```

Syntax Rules

1. Record-name and identifier must not refer to data that is allocated to the same storage area; record-name can be qualified.
2. Record-name is the name of a logical record in the File Section of the Data Division.

General Rules

1. The file associated with record-name must be a file on a directory device and must be open in the I-O mode at the time of execution of this statement.
2. The last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The record that was accessed by the READ statement is logically replaced. If the last input-output statement executed for the associated file was not a successfully executed READ statement, the REWRITE statement is not attempted, and the value 93 is placed in the FILE STATUS data item, if any, associated with the file to indicate an unsuccessful REWRITE operation. The data in the record area is unaffected.
3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.
4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause, in which case the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file as well as to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   MOVE identifier TO record-name

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. A REWRITE statement that is unsuccessful for any reason will cause a 30 to be stored in the FILE STATUS data item, if one was specified for the file.

5.31 REWRITE Statement (Relative)

Function

The REWRITE statement logically replaces a record existing in a file on a directory device.

General Format

```
REWRITE record-name[FROM identifier][; INVALID KEY imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not refer to data that is allocated to the same storage area.
2. Record-name is the name of a logical record in the File Section of the Data Division; record-name can be qualified.
3. The INVALID KEY phrase must not be specified for a REWRITE statement that references a file in sequential access mode.
4. The INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an applicable USE procedure is not specified.

General Rules

1. The file associated with record-name must be open in the I-O mode at the time of execution of this statement.
2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The record that was accessed by the READ statement is logically replaced. If the last input-output statement executed for the associated file was not a successfully executed READ statement, the REWRITE statement is not attempted and the value 93 is placed in the FILE STATUS data item, if any, associated with the file to indicate an unsuccessful REWRITE statement. The data in the current record area is unaffected.
3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.
4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause. In that case, the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   MOVE identifier TO record-name

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. For a file accessed in either random or dynamic access mode, the record specified by the contents of the RELATIVE KEY data item associated with the file is logically replaced. An INVALID KEY condition can arise; the REWRITE is considered unsuccessful, the data in the current record area is unaffected, and the following action is taken:

   a. If the record specified by the contents of the RELATIVE KEY data item does not exist, the value 23 is placed in the FILE STATUS data item, if any, associated with this file to indicate an unsuccessful REWRITE operation.

   b. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

   c. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

8. A REWRITE statement attempting to replace a record that is being simultaneously accessed by another task will be unsuccessful. The FILE STATUS data item, if one was specified for the file, is set to 92.

9. A REWRITE statement that is unsuccessful for an undetermined reason causes a 30 to be stored in the FILE STATUS data item, if one was specified for the file.

5.32 REWRITE Statement (Indexed)

Function

The REWRITE statement logically replaces a record existing in a mass storage file.

General Format

```
REWRITE record-name[FROM identifier][; INVALID KEY imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not refer to data that is allocated to the same storage area.
2. Record-name is the name of a logical record in the File Section of the Data Division and can be qualified.
3. The INVALID KEY phrase must be specified in the REWRITE statement for files for which an applicable USE procedure is not specified.

General Rules

1. The file associated with record-name must be open in the I-O mode at the time of execution of this statement.
2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The record that was accessed by the READ statement is logically replaced. If the last input-output statement executed for the associated file was not a successfully executed READ statement, the REWRITE statement is not attempted and the value 93 is placed in the FILE STATUS data item, if any, associated with the file to indicate an unsuccessful REWRITE statement. The data in the current record area is unaffected.
3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.
4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause. In that case, the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file. It is also available to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   ```
   MOVE identifier TO record-name
   ```

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. For a file in the sequential access mode, the record to be replaced is specified by the value contained in the prime record key. When the REWRITE statement is executed, the value contained in the prime record key data item of the record to be replaced must be equal to the value of the prime record key of the last record read from this file. If this relationship does not occur, then an INVALID KEY condition exists.

   When the INVALID KEY condition is recognized, actions are taken in the following order:

   a. The value 21 is placed into the FILE STATUS data item, if specified for this file to indicate an INVALID KEY condition.

   b. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any USE procedure specified for this file is not executed.

   c. If the INVALID KEY phrase is not specified but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

      When the INVALID KEY condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

8. For a file in either random or dynamic access mode, the record specified by the contents of the prime record key data item associated with the file is logically replaced. If the value contained in the prime record key does not equal that of any record stored in the file, an INVALID KEY condition exists. The value 23 is placed in the FILE STATUS data item, if one was specified for the file. See General Rules 7b and c.

9. The contents of alternate record key data items of the record being rewritten can differ from those in the record being replaced. The Record Management Services utilize the contents of the record key data items during the execution of the REWRITE statement to allow subsequent access of

the record based upon any of the specified record keys. If the value contained in an alternate record key for which a DUPLICATE clause has not been specified is equal to that of a record already stored in the file, the INVALID KEY condition exists. The value 02 is placed in the FILE STATUS data item if one was specified for the file. See General Rules 7b and c.

10. A REWRITE statement attempting to replace a record that is being simultaneously accessed by another task will fail. The value 92 is placed into the FILE STATUS data item if one was specified for the file.

11. A REWRITE statement that fails for an undetermined reason will cause the value 30 to be placed in the FILE STATUS data item if one was specified for the file.

5.33 SEARCH Statement

Function

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

General Format

Format 1

```
SEARCH identifier-1 [ VARYING { identifier-2 } ]
                              { index-name-1 }

    [; AT END imperative-statement-1]

     ; WHEN condition-1 { imperative-statement-2 }
                        { NEXT SENTENCE          }

    [ , WHEN condition-2 { imperative-statement-3 } ] ...
                         { NEXT SENTENCE          }
```

Format 2

```
SEARCH ALL identifier-1   [; AT END imperative-statement-1]

     ; WHEN { data-name-1      { IS EQUAL TO } { identifier-3          } }
            {                  { IS =         } { literal-1            } }
            {                                   { arithmetic-expression-1} }
            { condition-name-1                                           }

    [       { data-name-2      { IS EQUAL TO } { identifier-4          } } ]
    [ AND   {                  { IS =         } { literal-2            } } ] ....
    [       { condition-name-2                 { arithmetic-expression-2} } ]

    { imperative-statement-2 }
    { NEXT SENTENCE          }
```

NOTE:

The required relational character = (equal sign) is not underlined to avoid confusion with other symbols.

Syntax Rules

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

2. Identifier-2, when specified, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.

3. In Format 1, condition-1, condition-2, etc., can be any condition as described in Section 5.6, Conditional Expressions.

4. In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2 must be indexed by the first index-name associated with identifier-1 along with other indexes or literals as required, and must be referenced in the KEY clause of identifier-1. Identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2 must not be referenced in the KEY clause of identifier-1 or be indexed by the first index-name associated with identifier-1.

   In Format 2, when a data-name in the KEY clause of identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY clause of identifier-1 is referenced, all preceding data-names in the KEY clause of identifier-1 or their associated condition-names must also be referenced.

General Rules

1. If Format 1 of the SEARCH is used, a serial search operation takes place, starting with the current index setting.

   a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. The number of occurrences of identifier-1, the last of which is the highest permissible, is discussed in the OCCURS clause. (See Section 4.3.5, OCCURS Clause.) Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next executable sentence.

b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1 (the number of occurrences of identifier-1, the last of which is the highest permissible is discussed in the OCCURS clause; see Section 4.3.5, OCCURS Clause), the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested.

   If none of the conditions is satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element outside the permissible range of occurrence values. In that case, the search terminates, as indicated in 1a.

   If one of the conditions is satisfied upon evaluation, the search terminates immediately and the imperative-statement associated with that condition is executed; the index-name remains set at the occurrence that caused the condition to be satisfied.

2. In a Format 2 SEARCH, the results of the SEARCH ALL operation are predictable only when the following conditions are met:

   a. The data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of identifier-1.

   b. The contents of the key(s) referenced in the WHEN clause are sufficient to identify a unique table element.

3. If Format 2 of the SEARCH is used, a nonserial search may take place; the initial setting of the index-name for identifier-1 is ignored, and its setting is varied during the search operation, with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the table or that is less than the value that corresponds to the first element of the table. (The User's Guide contains further information on the SEARCH statement.) The length of the table is discussed in the OCCURS clause. (See Section 4.3.5, OCCURS Clause.)

   If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next executable sentence when this phrase is not specified; in either case the final setting of the index is not predictable.

   If all the conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to imperative-statement-2.

4. After execution of imperative-statement–1, imperative-statement–2, or imperative-statement–3 that does not terminate with a GO TO statement, control passes to the next executable sentence.

5. In Format 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier–1. Any other index-names for identifier–1 remain unchanged.

6. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier–1. Any other index-names for identifier–1 remain unchanged.

7. In Format 1, if the VARYING index-name–1 phrase is specified, and if index-name–1 appears in the INDEXED BY phrase of identifier–1, that index-name is used for this search. If this is not the case, or if the VARYING identifier–2 phrase is specified, the first (or only) index-name given in the INDEXED BY phrase of identifier–1 is used for the search. In addition, the following operations will occur:

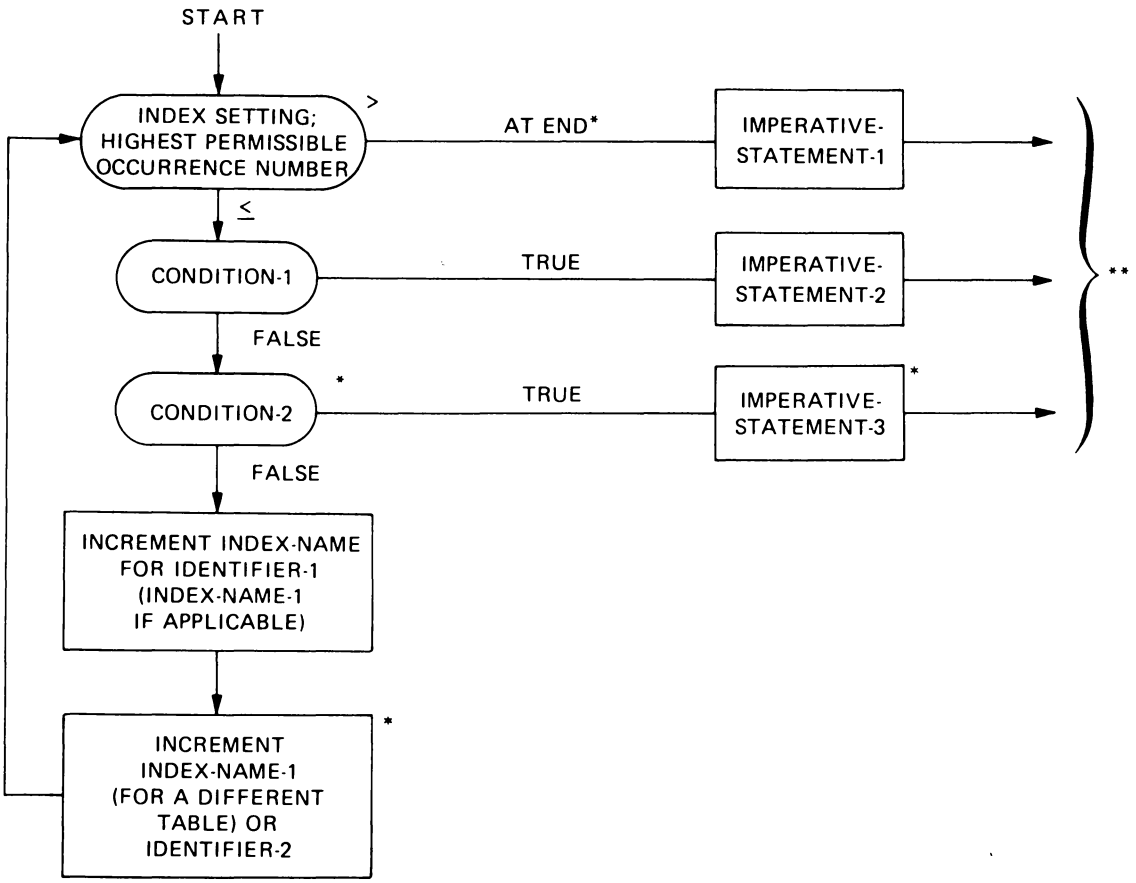
   a. If the VARYING index-name–1 phrase is used, and if index-name–1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name–1 is incremented by the same amount as and at the same time as the occurrence number represented by the index-name associated with identifier–1 is incremented.

   b. If the VARYING identifier–2 phrase is specified, and identifier–2 is an index data item, then the data item referenced by identifier–2 is incremented by the same amount as and at the same time as the index associated with identifier–1 is incremented. If identifier–2 is not an index data item, the data item referenced by identifier–2 is incremented by the value one (1) at the same time as the index referenced by the index-name associated with identifier–1 is incremented.

8. If identifier–1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a 2 or 3 dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier–1 (and the data item identifier–2 or index-name–1, if present) is modified by the execution of the SEARCH statement. To search an entire 2 or 3 dimensional table, it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

   A flowchart of the Format 1 SEARCH operation containing two WHEN phrases follows:

Figure 5-4: Format 1 SEARCH With Two WHEN Phrases



* These operations are options included only when specified in the SEARCH statement.
** Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

5.34 SET Statement

Function

The SET statement establishes a value in an index-name or index data-item.

General Format

Format 1

```
SET { identifier-1 [, identifier-2] ... } TO { identifier-3 }
    { index-name-1 [, index-name-2] ... }    { index-name-3 }
                                             { integer-1    }
```

Format 2

```
SET  index-name-4 [, index-name-5] ...  { UP BY   } { identifier-4 }
                                        { DOWN BY } { integer-2    }
```

Syntax Rules

1. All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.
2. Identifier-1 and identifier-3 must name either an index data item or an elementary item described as a numeric integer.
3. Identifier-4 must be described as an elementary numeric integer.
4. Integer-1 and integer-2 can be signed. Integer-1 must be positive.

General Rules

1. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.
2. If index-name-3 is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the associated table. This is guaranteed by the fact that the compiler automatically initializes all index-names with a value corresponding to an occurrence number of one.

   If index-name-4, index-name-5 is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. If index-name-1, index-name-2 is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the associated table.

3. In Format 1, the following steps occur:

   a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1.

   b. If identifier-1 is an index data item, it can be set equal to the contents of either the occurrence number portion of index-name-3 or to identifier-3 where identifier-3 is also an index data item.

   c. If identifier-1 is not an index data item, it can be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

   d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time, the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of literal-2 or identifier-4; thereafter, the process is repeated for index-name-5, etc. Each time, the value of identifier-4 is used as it was at the beginning of the execution of the statement.

5. Data in the following table represents the validity of various operand combinations in the SET statement. References after a slash mark (for example, /3b) refer to the applicable general rule for the SET statement.

Table 5-7: Permissible Operand Combinations in the SET Statement

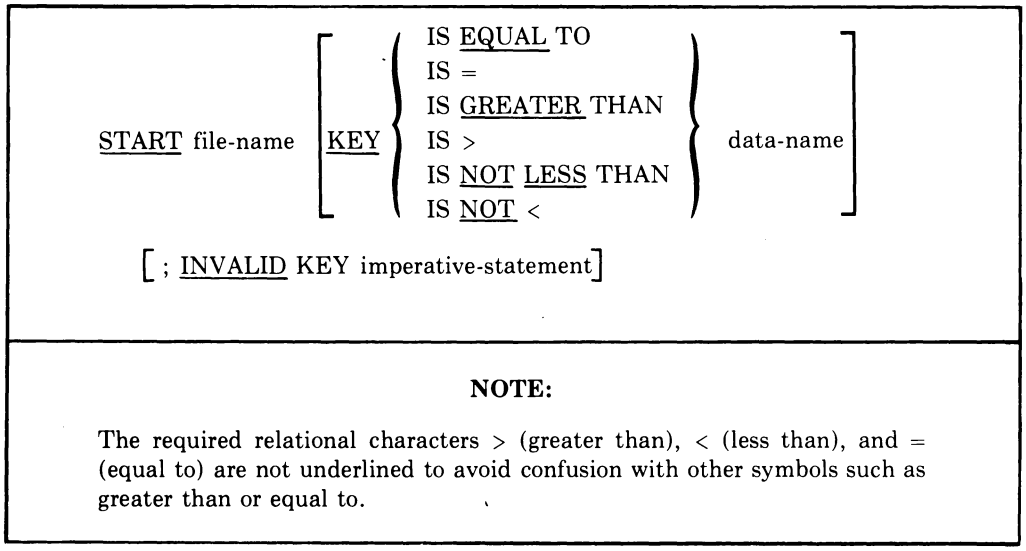
Sending Item	Receiving Item		
	Integer Data Item	**Index-Name**	**Index Data Item**
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data Item	No/3c	Valid/3a	No/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b
Index Data Item	No/3c	Valid/3a	Valid/3b

5.35 START Statement (Relative)

Function

The START statement logically positions a relative file for subsequent sequential retrieval of records.

General Format



NOTE:

The required relational characters > (greater than), < (less than), and = (equal to) are not underlined to avoid confusion with other symbols such as greater than or equal to.

Syntax Rules

1. File-name must be the name of a file with sequential or dynamic access.
2. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.
3. Data-name, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

General Rules

1. File-name must be open in the INPUT or I-O mode at the time of execution of the START statement.
2. If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied.
3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and the data item referenced by the RELATIVE KEY clause associated with file-name.

a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. The following action is taken:

   (1) If the contents of the RELATIVE KEY data item are within the range of the key values corresponding to the allocated space for this file, the value 23 is placed in the FILE STATUS data item, if any, associated with the file.

   (2) If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any USE procedure specified for this file is not executed.

   (3) If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement that recognized the condition is unsuccessful and the file is not affected.

4. A START statement that repositions the current record pointer to a record that is being simultaneously accessed by another task can fail. The FILE STATUS data item, if one was specified for the file, is set to 92.

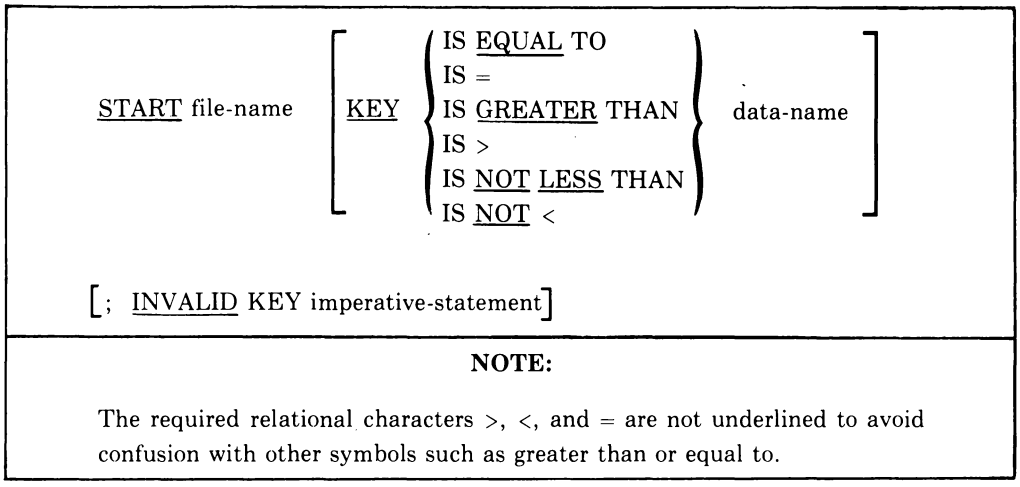
5. A START statement that is unsuccessful for an undetermined reason will cause a 30 to be stored in the FILE STATUS data item, if one was specified for the file.

5.36 START Statement (Indexed)

Function

The START statement provides a basis for logical positioning within an indexed file, for subsequent sequential retrieval of records.

General Format



NOTE:

The required relational characters >, <, and = are not underlined to avoid confusion with other symbols such as greater than or equal to.

Syntax Rules

1. File-name must be the name of a file with sequential or dynamic access.
2. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.
3. If file-name is the name of an indexed file, and if the KEY phrase is specified, data-name can reference a data item specified as a record key associated with file-name, or it can reference any data item of category alphanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.

General Rules

1. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed.
2. If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied.
3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in General Rule 6. If file-name

references an indexed file and the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply.

a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists. The execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. The FILE STATUS data item, if one was specified for the file, is set to 23.

4. If the KEY phrase is specified, the comparison described in General Rule 3 uses the data item referenced by data-name.

5. If the KEY phrase is not specified, the comparison described in General Rule 3 uses the data item referenced in the RECORD KEY clause associated with file-name.

6. Upon completion of the successful execution of the START statement, a key of reference is established and used in subsequent Format 1 READ statements as follows:

   a. If the KEY phrase is not specified, the prime record key specified for file-name becomes the key of reference.

   b. If the KEY phrase is specified, and data-name is specified as a record key for file-name, that record key becomes the key of reference.

   c. If the KEY phrase is specified, and data-name is not specified as a record key for file-name, the record key whose leftmost character position corresponds to the leftmost character position of the data item specified by data-name becomes the key of reference.

7. If the execution of the START statement is not successful, the key of reference is undefined.

8. A START statement that repositions the current record pointer to a record that is being simultaneously accessed by another task can fail. The FILE STATUS data item, if one was specified for the file, is set to 92.

9. A START statement that is unsuccessful for an undetermined reason causes a 30 to be stored in the FILE STATUS data item, if one was specified for the file.

5.37 STOP Statement

Function

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

General Format

```
STOP { RUN     }
     { literal }
```

Syntax Rules

1. The literal can be numeric or nonnumeric, or any figurative constant except ALL.
2. If the literal is numeric, then it must be an unsigned integer.
3. If a STOP RUN statement appears in a consecutive sequence of imperative-statements within a sentence, it must appear as the last statement in that sequence.

General Rules

1. If the RUN phrase is used, the standard ending procedure is executed, and object program execution is terminated.
2. If STOP literal is specified, the literal is displayed on the user's standard display device. Control returns to the command language level without terminating the image. Entering a VMS CONTINUE command causes the image to resume at the next executable statement in sequence.

   Interrupting program execution is discussed in the *VAX/VMS Command Language User's Guide*.

5.38 STRING Statement

Function

The STRING statement provides concatenation of the partial or complete contents of two or more data items into a single data item.

General Format

```
STRING { identifier-1 } [, identifier-2 ]
       { literal-1    } [, literal-2    ]  ... DELIMITED BY { identifier-3 }
                                                           { literal-3    }
                                                           { SIZE         }

[ , { identifier-4 } [, identifier-5 ]                     { identifier-6 } ... ]
    { literal-4    } [, literal-5    ]  ... DELIMITED BY { literal-6    }
                                                         { SIZE         }

    INTO identifier-7 [WITH POINTER identifier-8]
    [; ON OVERFLOW imperative-statement]
```

Syntax Rules

1. Each literal can be any figurative constant without the optional word ALL.
2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-8, must be described implicitly or explicitly as USAGE IS DISPLAY.
3. Identifier-7 must represent an elementary alphanumeric data item without editing symbols or the JUSTIFIED clause.
4. Identifier-8 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size, plus 1, of the area referenced by identifier-7. The symbol P cannot be used in the PICTURE character-string of identifier-8.
5. Where identifier-1, identifier-2, ..., or identifier-6 is an elementary numeric data item, it must be described as an integer without the symbol P in its PICTURE character-string.

General Rules

1. All references to identifier-1, identifier-2, identifier-3, literal-1, literal-2, literal-3 apply equally to identifier-4, identifier-5, identifier-6, literal-4, literal-5 and literal-6, respectively, and all recursions thereof.

2. Identifier-1, literal-1, identifier-2, literal-2 represent the sending items. Identifier-7 represents the receiving item.

3. Literal-3, identifier-3, indicate the character(s) delimiting the move. If the SIZE phrase is used, the complete data item defined by identifier-1, literal-1, identifier-2, literal-2 is moved. When a figurative constant is used as the delimiter, it stands for a single-character, nonnumeric literal.

4. When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit 1-character data item whose USAGE IS DISPLAY.

5. When the STRING statement is executed, the transfer of data is governed by the following rules:

   a. Those characters from literal-1, literal-2, or from the contents of the data item referenced by identifier-1, identifier-2 are transferred to the contents of identifier-7 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space-filling will be provided. (See Section 5.22, MOVE Statement).

   b. If the DELIMITED phrase is specified without the SIZE phrase, the contents of the data item referenced by identifier-1, identifier-2, or the value of literal-1, literal-2, are transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character and continuing from left to right until the end of the data item is reached or until the character(s) specified by literal-3 or by the contents of identifier-3 are encountered. The character(s) specified by literal-3 or by the data item referenced by identifier-3 are not transferred.

   c. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, literal-2, or the contents of the data item referenced by identifier-1, identifier-2, are transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-7 until all data has been transferred or the end of the data item referenced by identifier-7 has been reached.

6. If the POINTER phrase is specified, identifier-8 is explicitly available to you, and you are responsible for setting its initial value. The initial value must not be less than one.

7. If the POINTER phrase is not specified, General Rules 8 through 11 apply as if the user had specified identifier-8 with an initial value of 1.

8. When characters are transferred to the data item referenced by identifier-7, the moves behave as though the characters were moved one at a time from the source into the character position of the data item referenced by identifier-7 designated by the value associated with identifier-8,

and then identifier-8 was increased by one prior to the move of the next character. The value associated with identifier-8 is changed during execution of the STRING statement only by the behavior specified above.

9. At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-7 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-7 will contain data that was present before this execution of the STRING statement.

10. If at any point at or after initialization of the STRING statement, but before execution of the statement is completed, the value associated with identifier-8 is either less than one or exceeds the number of character positions in the data item referenced by identifier-7, no (further) data is transferred to the data item referenced by identifier-7, and the imperative statement in the ON OVERFLOW phrase is executed, if specified.

11. If the ON OVERFLOW phrase is not specified when the conditions described in General Rule 10 are encountered, control passes to the next executable statement.

5.39 SUBTRACT Statement

Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from an item and to set the value of an item equal to the results.

General Format

Format 1

```
SUBTRACT { identifier-1 } [ ,identifier-2 ] ... FROM identifier-m [ROUNDED]
         { literal-1    } [ ,literal-2   ]

  [identifier-n [ROUNDED] ]...

  [ ; ON SIZE ERROR imperative-statement]
```

Format 2

```
SUBTRACT { identifier-1 } [ ,identifier-2 ] ... FROM { identifier-m }
         { literal-1    } [ ,literal-2   ]          { literal-m    }

  GIVING identifier-n [ROUNDED], [identifier-o [ROUNDED]] ...

  [ ; ON SIZE ERROR imperative-statement ]
```

Format 3

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ROUNDED]
         { CORR          }

  [ ; ON SIZE ERROR imperative-statement ]
```

Syntax Rules

1. Each identifier must refer to a numeric elementary item, except that:
   a. In Format 2, each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.
   b. In Format 3, each identifier must refer to a group item.
2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits. (See Section 5.7.4, Arithmetic Statements.)

   a. In Format 1, the composite of operands is determined by using all of the operands in a given statement.

   b. In Format 2, the composite of operands is determined by using all of the operands in a given statement excluding the data item that follows the word GIVING.

   c. In Format 3, the composite of operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. In Format 1, all literals or identifiers preceding the word FROM are added together, and this total is subtracted from the current value of identifier-m, storing the result immediately into identifier-m, and repeating this process respectively for each operand following identifier-m.

2. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m, and the result of the subtraction is stored as the new value of identifier-n, identifier-o, etc.

3. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

4. The compiler insures that enough places are carried (unless an intermediate result exceeds the 18-digit limitation) to avoid losing significant digits during execution.

5.40 UNSTRING Statement

Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

General 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]
[ , identifier-7 [,DELIMITER IN identifier-8 ] [, COUNT IN identifier-9] ] ...
[ WITH POINTER identifier-10] [TALLYING IN identifier-11]
[ ; ON OVERFLOW imperative-statement]
```

Syntax Rules

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant without the optional word ALL.
2. Identifier-1, identifier-2, identifier-3, identifier-5, and identifier-8 must be described, implicitly or explicitly, as an alphanumeric data item.
3. Identifier-4 and identifier-7 can be described as either alphabetic (except that the symbol B cannot be used in the PICTURE character-string), alphanumeric, or numeric (except that the symbol P cannot be used in the PICTURE character-string), and must be described as USAGE IS DISPLAY.
4. Identifier-6, identifier-9, identifier-10, and identifier-11 must be described as elementary numeric integer data items (except that the symbol P cannot be used in the PICTURE character-string).
5. No identifier can name a level 88 entry.
6. The DELIMITER IN phrase and the COUNT IN phrase can be specified only if the DELIMITED BY phrase is specified.

General Rules

1. All references to identifier-2, literal-1, identifier-4, identifier-5 and identifier-6 apply equally to identifier-3, literal-2, identifier-7, identifier-8, and identifier-9, respectively, and all recursions thereof.

2. Identifier-1 represents the sending area.

3. Identifier-4 represents the data receiving area. Identifier-5 represents the receiving area for delimiters.

4. Literal-1 or the data item referenced by identifier-2 specifies a delimiter.

5. Identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 that were isolated by the delimiters for the move to identifier-4. This value does not include a count of the delimiter character(s).

6. The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1.

7. The data item referenced by identifier-11 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.

8. When a figurative constant is used as the delimiter, it stands for a 1-character nonnumeric literal.

   When the ALL phrase is specified, one occurrence, or two or more contiguous occurrences of literal-1 (figurative constant or not) or the contents of the data item referenced by identifier-2 are treated as if it were only one occurrence, and this occurrence is moved to the receiving data item according to the rules in General Rule 13d.

9. When any examination encounters two contiguous delimiters, the current receiving area is either space or zero filled according to the description of the receiving area.

10. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the computer character set.

11. Each literal-1 or each data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item and in the order given to be recognized as a delimiter.

12. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

13. When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data

item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

a. If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined, beginning with the relative character position indicated by the content of the data item referenced by identifier-10. If the POINTER phrase is not specified, the string of characters is examined, beginning with the leftmost character position.

b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by the value of literal-1 or the data item referenced by identifier-2 is encountered. (See General Rule 11.) If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

   If the end of the data item referenced by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item and are moved into the current receiving area according to the rules for the MOVE statement. (See Section 5.22, MOVE Statement.)

d. If the DELIMITER IN phrase is specified, the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement. (See Section 5.22, MOVE Statement.) If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space filled.

e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into the area referenced by identifier-6, according to the rules for an elementary move.

f. If the DELIMITED BY phrase is specified, the string of characters is further examined, beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined, beginning with the character to the right of the last character transferred.

g. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The behavior described in paragraph 13b through 13f is repeated until either all the characters are exhausted in the data item referenced by identifier-1 or until there are no more receiving areas.

14. It is your responsibility to initialize the contents of the data items associated with the POINTER phrase or the TALLYING phrase.

15. The contents of the data item referenced by identifier-10 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the contents of the data item referenced by identifier-10 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of the data item referenced by identifier-11 contain a value equal to its initial value, plus the number of data-receiving items acted upon.

17. Either of the following situations causes an overflow condition:

    a. An UNSTRING is initiated, and the value in the data item referenced by identifier-10 is less than 1 or greater than the size of the data item referenced by identifier-1.

    b. If, during execution of an UNSTRING statement, all receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

18. When an overflow condition exists, the UNSTRING operation is terminated. If an ON OVERFLOW phrase has been specified, the imperative-statement included in the ON OVERFLOW phrase is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next executable statement.

19. The evaluation of subscripting and indexing for the identifiers is as follows:

    a. Any subscripting or indexing associated with identifier-1, identifier-10, identifier-11 is evaluated only once, immediately before any data is transferred as the result of the execution of the UNSTRING statement.

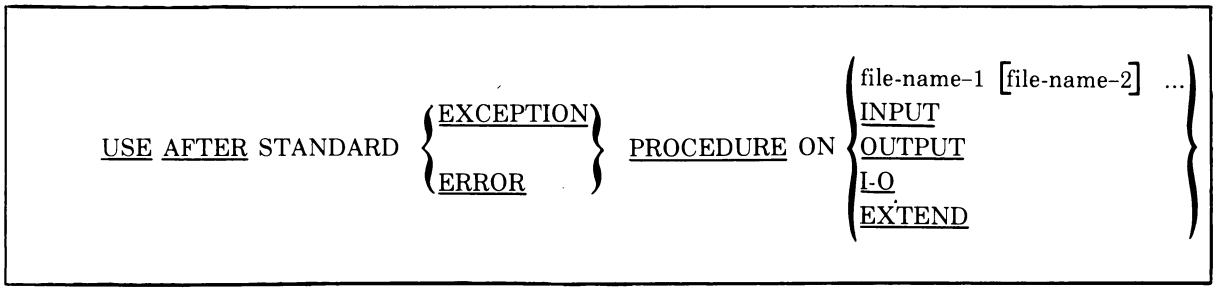
    b. Any subscripting or indexing associated with identifier-2, identifier-3, identifier-4, identifier-5, identifier-6 is evaluated immediately before the transfer of data into the respective data item.

5.41 USE Statement

Function

The USE statement specifies procedures for input-output error handling that supplement the standard procedures provided by the file system.

General Format



Syntax Rules

1. A USE statement, when present, must immediately follow a section header in the declaratives section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one, or more procedural paragraphs that define the procedures to be used.
2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.
3. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.
4. The words ERROR and EXCEPTION are synonymous and can be used interchangeably.
5. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

General Rules

1. The designated procedures are executed by the input-output system after completing the standard input-output error routine or upon recognition of the INVALID KEY or AT END condition when the INVALID KEY phrase or AT END phrase has not been specified in the input-output statement.
2. After execution of a USE procedure, control is returned to the invoking routine.

3. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements can refer to a USE statement or to the procedures associated with such a USE statement.

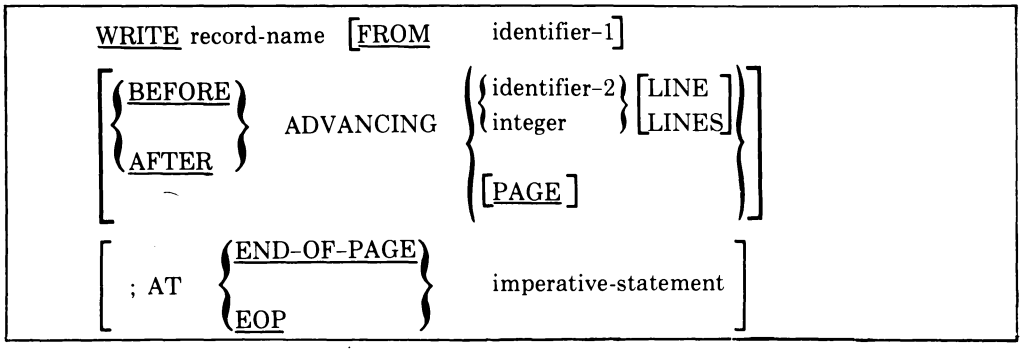
4. Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

5.42 WRITE Statement (Sequential)

Function

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of lines within a logical page.

General Format



Syntax Rules

1. Record-name and identifier-1 must not refer to data that is allocated to the same storage area.
2. Record-name is the name of a logical record in the File Section of the Data Division; record-name can be qualified.
3. When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.
4. Integer, or the value of the data item referenced by identifier-2, can be zero.
5. If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file-description-entry for the associated file.
6. The words END-OF-PAGE and EOP are equivalent.

General Rules

1. The associated file must be open in the OUTPUT or EXTEND mode at the time of the execution of this statement.
2. The logical record released by the successful execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier-1 TO record-name

      according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

   The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

   After execution of the WRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name may not be.

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

6. The number of character positions on a mass storage device required to store a logical record in a file can or cannot be equal to the number of character positions defined by the logical description of that record in the program.

7. The execution of the WRITE statement releases a logical record to the file systems.

8. Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

   a. If identifier-2 is specified, the representation of the printed page is advanced the number of lines equal to the current value associated with identifier-2.

   b. If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.

   c. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to Rules a and b above.

d. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to Rules a and b above.

e. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file whose file-description-entry contains a LINAGE clause, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause. If the record to be written is associated with a file whose file-description-entry does not contain a LINAGE clause, the repositioning to the next logical page is accomplished in accordance with the normal file system techniques. If page has no meaning in conjunction with a specific device, then advancing will be provided to act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

9. If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, the imperative-statement specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name.

10. An end-of-page condition is reached whenever the execution of a given WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and then the imperative-statement in the END-OF-PAGE phrase is executed.

    An automatic page overflow condition is reached whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

    This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. The imperative-statement in the END-OF-PAGE clause, if specified, is executed after the record is written and the device has been repositioned.

    If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

If integer–2 or data-name-2 of the LINAGE clause is specified, but the execution of a given WRITE statement would cause LINAGE–COUNTER to simultaneously exceed the value of both integer–2 (or the data item referenced by data-name-2) and integer–1 (or the data item referenced by data-name-1), then the operation proceeds as if integer–2 (or data-name-2) had not been specified.

11. When an attempt is made to write beyond the externally defined boundaries of a sequential file, the Record Management Services will attempt to extend the space allocated to the file on the medium. If that attempt is successful, the WRITE will be executed normally. If it is unsuccessful, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

    a. The value of the FILE STATUS data item, if any, of the associated file is set to a value of 34 indicating a boundary violation.

    b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure will then be executed.

    c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the execution of the object program is terminated.

12. After the recognition of an end-of-reel or an end-of-unit of an output file that is contained on more than one physical reel/unit, the WRITE statement performs the following operations:

    a. The standard ending reel/unit label procedure.

    b. A reel/unit swap.

    c. The standard beginning reel/unit label procedure.

13. A WRITE statement that is unsuccessful for an undetermined reason will cause a 30 to be stored in the FILE STATUS data item, if one was specified for the file.

5.43 WRITE Statement (Relative)

Function

The WRITE statement releases a logical record for an output or input-output file.

General Format

```
WRITE  record-name  [FROM  identifier]  [;  INVALID KEY  imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not refer to data that is allocated to the same storage area.
2. Record-name is the name of a logical record in the File Section of the Data Division; record-name can be qualified.
3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement.
2. The logical record released by the successful execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.
3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier TO record-name

      according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

   The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

   After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be.

4. The current record pointer is unaffected by the execution of a WRITE statement.
5. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.
6. The number of character positions on a storage medium required to store a logical record in a file will be greater than the number of character positions defined by the logical description of that record in the program.
7. The execution of the WRITE statement releases a logical record to Record Management Services.
8. When a file is opened in the output mode, records can be placed into the file by one of the following:

   a. If the access mode is sequential, the WRITE statement will cause a record to be released to the file control system. The first record will have a relative record number of one, and subsequent records released will have relative record numbers of 2, 3, 4, ... If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item during execution of the WRITE statement.

   b. If the access mode is random or dynamic, prior to the execution of the WRITE statement, the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released to Record Management Services by execution of the WRITE statement.

9. When a file is opened in the I-O mode and the access mode is random or dynamic, the WRITE statement allows records to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the Record Management Services.
10. An INVALID KEY condition can arise; the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the following actions take place.

    a. If the access mode is sequential, a boundary violation can occur if the WRITE statement attempted to write beyond the allocated space for the file and the Record Management Services was unsuccessful in obtaining additional space for the file. The value 24 is placed in the FILE STATUS data item, if any, associated with the file.

b. If the access mode is random or dynamic and the contents of the RELATIVE KEY data item specifies a record which already exists in the file, the value 22 is placed in the FILE STATUS data item, if any, associated with the file.

c. If the access mode is random or dynamic and the contents of the RELATIVE KEY data item do not lie in the range of key values associated with the file, a boundary violation can occur if the Record Management Services is unsuccessful in obtaining additional space for the file. The value 24 is placed in the FILE STATUS data item, if any, associated with the file.

11. A WRITE statement issued to a file that is being simultaneously accessed by another task will be unsuccessful. The FILE STATUS data item, if one was specified for the file, is set to 92.

12. A WRITE statement that is unsuccessful for an undetermined reason will cause a 30 to be stored in the FILE STATUS data item, if one was specified for the file.

5.44 WRITE Statement (Indexed)

Function

The WRITE statement releases a logical record for an output or input-output file.

General Format

```
WRITE  record-name  [FROM  identifier]  [;  INVALID KEY  imperative-statement ]
```

Syntax Rules

1. Record-name and identifier must not refer to data that is allocated to the same storage area.
2. Record-name is the name of a logical record in the File Section of the Data Division.
3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement.
2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement is unsuccessful. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.
3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier TO record-name

   according to the rules specified for the MOVE statement, followed by:

b. The same WRITE statement without the FROM phrase.

The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be. (See General Rule 2).

4. The current record pointer is unaffected by the execution of a WRITE statement.
5. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.
6. The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.
7. The execution of the WRITE statement releases a logical record to the Record Management Services.
8. Execution of the WRITE statement causes the contents of the record area to be released. The Record Management Services utilizes the content of the record keys in such a way that subsequent access of the record may be made based upon any of those specified record keys.
9. The value of the prime record key must be unique within the records in the file.
10. The data item specified as the prime record key must be set by the program to the desired value prior to the execution of the WRITE statement.
11. If sequential access mode is specified for the file, records must be released to the Record Management Services in ascending order of prime record key values.
12. If random or dynamic access mode is specified, records may be released to the Record Management Services in any program-specified order.
13. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the Record Management Services provides storage of records such that, when records are accessed sequentially, the order of retrieval of those records is the order in which they were released to the Record Management Services.

14. The INVALID KEY condition exists under the following circumstances:

NOTE:

The value in parentheses immediately following each statement is the value that is placed in the FILE STATUS data item, if one was specified for the file.

a. When sequential access mode is specified for a file opened in the output mode and the value of the prime record key is not greater than the value of the prime record key of the previous record, (21)

b. When the file is opened in the output or I-O mode and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file, (22)

c. When the file is opened in the output or I-O mode and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file, (22)

d. When the device to which the file is assigned has no more space to contain the new record, (24)

e. When an attempt is made to write a record that is being simultaneously accessed by another task, (92)

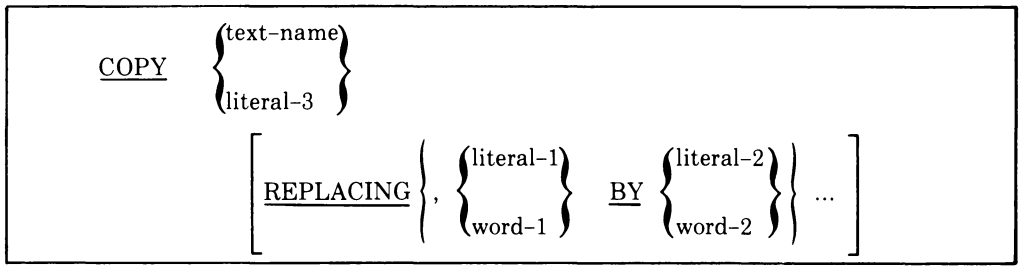
f. When an unidentifiable error occurs. (30)

Chapter 6
The Library Module

Function

The library module provides a capability for specifying text that is to be copied from a library file. The COPY statement incorporates this text into a COBOL source program.

General Format



Syntax Rules

1. Each text-name must be unique within the file directories available to the compiler. (See Section 1.1.2.1, User-Defined Words.)
2. The COPY statement must be preceded by a space and terminated by the separator period.
3. Word-1 or word-2 may be any single COBOL word.
4. COPY statement may occur in the source program anywhere a character-string or a separator may occur except that a COPY statement must not occur within a COPY statement.
5. Literal-3 is a non-numeric literal containing a file specification. The use of text-name is equivalent to specifying "SYS$DISK:textname.LIB".

General Rules

1. When a COPY statement is specified, the library text associated with text-name is copied into the source program. The entire COPY statement is logically replaced, beginning with the reserved word COPY and ending with the punctuation character period, inclusive.

2. If the REPLACING phrase is not specified, the library text is copied unchanged.

   If the REPLACING phrase is specified, the library text is copied and each properly matched occurrence of word-1 and literal-1 in the library text is replaced by the corresponding word-2, or literal-2.

3. The comparison operation to determine text replacement occurs as follows:

   Any separator comma, semicolon and/or space(s) preceding the leftmost library text-word is copied into the source program. Starting with the leftmost library text-word and the first word-1, or literal-1, that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to a library text-word.

   Word-1, or literal-1, matches the library text if, and only if, the text-word that forms word-1, or literal-1 is equal, character for character, to the library text-word.

   If no match occurs, the comparison is repeated with each next successive word-1, or literal-1, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

   When all the REPLACING operands have been compared and no match has occurred, the leftmost library text-word is copied into the source program. The next successive library text-word is then considered as the leftmost library text-word, and the comparison cycle starts again with the first word-1, or literal-1, specified in the REPLACING phrase.

   Whenever a match occurs between word-1, or literal-1, and the library text, the corresponding word-2, or literal-2, is placed into the source program. The library text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost library text-word. The comparison cycle starts again with the first word-1, or literal-1 specified in the REPLACING phrase.

   The comparison operation continues until the rightmost text-word in the library text has either participated in a match or been considered as a leftmost library text-word and participated in a complete comparison cycle.

4. Comment lines appearing in library text are copied unchanged into the source program.

5. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

6. Library text must conform to the rules for COBOL reference format. A program written in conventional reference format must COPY only library files also written in conventional reference format. COPY statements appearing in a file that was created using terminal format, can only refer to library files that were created using the same format.

Appendix A
Reserved Words

The following is a list of reserved words taken from American National Standard COBOL, with some additional words that represent this compiler's extensions to the COBOL language. Words that are not reserved by the standard are indicated by an asterisk. All of the following words are reserved by the compiler and must not be used as user-defined words.

ACCEPT
ACCESS
* ACTUAL
ADD
* ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
* APPLY
ARE
AREA
AREAS
ASCENDING
* ASCII
ASSIGN
AT
AUTHOR

BEFORE
* BEGINNING
* BINARY
BLANK
BLOCK
BOTTOM
BY

CALL
CANCEL
* CARD-PUNCH
* CARD-READER
CD
CF
CH
* CHANNEL
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
* COMP-1
* COMP-3
* COMP-6
COMPUTATIONAL
* COMPUTATIONAL-1
* COMPUTATIONAL-3
* COMPUTATIONAL-6
COMPUTE
CONFIGURATION
* CONSOLE

CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
* DECSYSTEM-10
* DEFERRED
DELETE
DELIMITED
DELIMITER

* DENSITY
DEPENDING
DEPTH
DESCENDING
* DESCRIPTOR
DESTINATION
DETAIL
DISABLE
DISPLAY
* DISPLAY-6
* DISPLAY-7
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

* EBCDIC
EGI
ELSE
EMI
ENABLE
END
* ENDING
END-OF-PAGE
ENTER
* ENTRY
ENVIRONMENT
EOP
EQUAL
* EQUALS
ERROR
ESI
* EVEN
EVERY
* EXAMINE
EXCEPTION
EXIT
EXTEND

FD
FILE
FILE-CONTROL
* FILE-LIMIT
* FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
* FORTRAN
* FORTRAN-IV
FROM

GENERATE
GIVING
GO
* GO BACK
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

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

JUST
JUSTIFIED

KEY
* KEYS

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

* MACRO
* MAP4
* MAP5
* MAP6
* MAP7
* MAP8
MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
NOT
* NOTE
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
* ODD
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER
* PAPER-TAPE-PUNCH
* PAPER-TAPE-READER
* PARITY
* PDP-10
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
* PRINT-CONTROL
PRINTING
PROCEDURE
PROCEDURES
PROCEED
* PROCESSING
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES

RANDOM
RD

READ
* READ-AHEAD
RECEIVE
RECORD
* RECORDING
RECORDS
REDEFINES
REEL
* REFERENCE
REFERENCES
RELATIVE
RELEASE
REMAINDER
* REMARKS
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
* SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
* SWITCH
SYMBOLIC
SYNC
SYNCHRONIZED

TABLE
* TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
* TODAY
TOP
* TRACE
TRAILING
TYPE

UNIT
* UNLOCK
UNSTRING
UNTIL
UP
UPON
USAGE
USE
* USER-NUMBER
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
* WRITE-BEHIND

ZERO
ZEROES
ZEROS

+
-
*
/
**
>
<
=

Appendix B
Character Sets

The following table shows the characters of the computer character set (ASCII) with each character's decimal and hexadecimal equivalent.

Characters belonging to set "C" constitute the COBOL character set. Set "L" contains those characters that can appear in nonnumeric literals. The characters in set "X" delimit lines of the source text.

Table B-1: Character Sets

Decimal	Hex	Character	Set		Decimal	Hex	Character	Set	
000	00	NUL		L	032	20	space	C	L
001	01	SOH		L	033	21	!		L
002	02	STX		L	034	22	"	C	L
003	03	ETX		L	035	23	#		L
004	04	EOT		L	036	24	$	C	L
005	05	ENQ		L	037	25	%		L
006	06	ACK		L	038	26	&		L
007	07	BEL		L	039	27	'		L
008	08	BS		L	040	28	(	C	L
009	09	HT	C		041	29	)	C	L
010	0A	LF		X	042	2A	*	C	L
011	0B	VT		X	043	2B	+	C	L
012	0C	FF		X	044	2C	,	C	L
013	0D	CR		X	045	2D	-	C	L
014	0E	SO		L	046	2E	.	C	L
015	0F	SI		L	047	2F	/	C	L
016	10	DLE		L	048	30	0	C	L
017	11	DC1		L	049	31	1	C	L
018	12	DC2		L	050	32	2	C	L
019	13	DC3		L	051	33	3	C	L
020	14	DC4		L	052	34	4	C	L
021	15	NAK		L	053	35	5	C	L
022	16	SYN		L	054	36	6	C	L
023	17	ETB		L	055	37	7	C	L
024	18	CAN		L	056	38	8	C	L
025	19	EM		L	057	39	9	C	L
026	1A	SUB		L	058	3A	:		L
027	1B	ESC		L	059	3B	;	C	L
028	1C	FS		L	060	3C	<	C	L
029	1D	GS		L	061	3D	=	C	L
030	1E	RS		L	062	3E	>	C	L
031	1F	US		L	063	3F	?		L

Characters belonging to set "C" constitute the COBOL character set. Set "L" contains those characters that can appear in nonnumeric literals. The characters in set "X" delimit lines of the source text.

Table B-1: Character Sets (continued)

Decimal	Hex	Character	Set	Decimal	Hex	Character	Set
064	40	@	L	096	60	`	L
065	41	A	C L	097	61	a	L
066	42	B	C L	098	62	b	L
067	43	C	C L	099	63	c	L
068	44	D	C L	100	64	d	L
069	45	E	C L	101	65	e	L
070	46	F	C L	102	66	f	L
071	47	G	C L	103	67	g	L
072	48	H	C L	104	68	h	L
073	49	I	C L	105	69	i	L
074	4A	J	C L	106	6A	j	L
075	4B	K	C L	107	6B	k	L
076	4C	L	C L	108	6C	l	L
077	4D	M	C L	109	6D	m	L
078	4E	N	C L	110	6E	n	L
079	4F	O	C L	111	6F	o	L
080	50	P	C L	112	70	p	L
081	51	Q	C L	113	71	q	L
082	52	R	C L	114	72	r	L
083	53	S	C L	115	73	s	L
084	54	T	C L	116	74	t	L
085	55	U	C L	117	75	u	L
086	56	V	C L	118	76	v	L
087	57	W	C L	119	77	w	L
088	58	X	C L	120	78	x	L
089	59	Y	C L	121	79	y	L
090	5A	Z	C L	122	7A	z	L
091	5B	[	L	123	7B	{	L
092	5C	\	L	124	7C	|	L
093	5D	]	L	125	7D	}	L
094	5E	^	L	126	7E	~	L
095	5F	_	L	127	7F	DEL	L

Appendix C
File Status Key Values

Table C-1: Sequential I/O File Status Key Values

Status Key Code	Meaning
00	No further information (successful).
10	End-of-file indicator detected.
30	Permanent error.
34	Permanent error (boundary error on WRITE statement).
91	File locked by another task.
93	REWRITE attempted without prior READ.
94	Improper operation attempted.
95	Allocation failure on OPEN (no file space on device).
96	No buffer space. Program tried to open a file that is sharing buffer space (SAME AREA) with another file.
97	No such file. The file named in an OPEN statement was not found.
98	Close error. Error discovered while in the process of closing the file.

Table C-2: Relative and Indexed I/O File Status Key Values

Status Key Code	Meaning
00	No further information (successful).
02	A record written into an indexed file by a WRITE or REWRITE statement contains at least one key value that was already present in another record.
10	End-of-file indicator detected.
21	Sequence error on primary key during the execution of a WRITE or REWRITE statement.
22	Duplicate key error.
23	No such record.
24	Boundary error on WRITE statement.
30	Permanent error.
91	File locked by another task.
92	Record locked by another task.
93	REWRITE or DELETE attempted without prior READ.
94	Improper operation attempted.
95	Allocation failure (no file space on device).
96	No buffer space. Program tried to open a file that is sharing buffer space (SAME AREA) with another file.
97	No such file. The file named in an OPEN statement was not found.
98	Close error. Error discovered while in the process of closing a file.

Glossary

Abbreviated Combined Relation Condition

The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Abnormal Termination

The premature end of execution of a program due to the detection by the operating system of a situation that prevents further successful operation of that program.

Access Mode

How records are to be operated upon in a file. The COBOL access modes are SEQUENTIAL, RANDOM, and DYNAMIC.

Actual Decimal Point

The physical representation, using a period (.) or comma (,), of the decimal point position in a data item. (See also Assumed Decimal Point.)

Alphabet-Name

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

Alphabetic Character

A character from the following set of characters A–Z and the space. (See also Alphanumeric Character, Numeric Character.)

Alphanumeric Character

Any character in the computer character set. (See also Alphabetic Character and Numeric Character.)

Alphanumeric Literal

(See Nonnumeric Literal.)

Alternate Record Key

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

Arithmetic Expression

An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operation

The process started by the execution of an arithmetic statement or the evaluation of an arithmetic expression that results in a mathematically correct solution to that expression, using the arguments presented.

Arithmetic Operator

A single character or a fixed 2-character combination of the character(s) that belong to the following set:

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

Ascending Key

A key upon whose values data is ordered, starting with the lowest key value and going to the highest key value in accordance with the rules for comparing data items.

Assumed Decimal Point

A decimal point position that is not an actual character in the data item. The assumed decimal point has logical meaning but no physical representation. (See also Actual Decimal Point.)

AT END Condition

A condition that results during:

1. The execution of a READ statement for a sequentially accessed file when no next logical record exists for the file or when an optional file is not present.
2. The execution of a SEARCH statement when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

Bit

The smallest unit in a computer storage structure capable of expressing two distinct alternatives.

Block

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

Bottom Margin

An empty area that follows the page body.

Byte

Synonymous with Character Position.

Called Program

A program that is the object of a CALL statement. It is combined with the calling program to produce a run unit.

Calling program

A program that executes a CALL to another program.

Character

The basic, indivisible unit of the COBOL language.

Character Data Item

A data item consisting entirely of Standard Data Format characters.

Character Position

The amount of physical storage required to store a single Standard Data Format character whose usage is DISPLAY.

Character-String

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

Class Condition

The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or wholly numeric.

Clause

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

COBOL Character Set

The set of characters that combine to form COBOL character-strings and separators. The COBOL character set is listed in Appendix B. (See also Computer Character Set.)

COBOL Word

(See Word.)

Collating Sequence

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

Column

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

Combined Condition

A condition resulting from the connection of two or more conditions with the AND or OR logical operator.

Comment-Entry

An entry in the Identification Division that can be any combination of characters from the computer character set.

Comment Line

A source program line with an asterisk in the indicator area of the line. Areas A and B can contain any characters from the computer character set. The comment line serves only for program documentation. A special form of comment line causes page ejection prior to printing the comment; it is identified by a stroke (/) in the indicator area of the line.

Compile Time

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

Compiler

A program that translates a source program into an object program.

Compiler-Directing Statement

A statement beginning with a compiler-directing verb that causes the compiler to take a specific action during compilation.

Complex Condition

A condition in which one or more logical operators act upon one or more conditions. (See Negated Simple Condition; Combined Condition; Negated Combined Condition.)

Computer Character Set

The set of all characters that can be represented or stored in the computer. As used in this manual, the set sometimes excludes the source program line delimiters. The computer character set is listed in Appendix B. (See also COBOL Character Set.)

Computer-Name

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

Concurrent Run Unit

A run unit, other than the current run unit, that has been initiated but not terminated during the time in which the current run unit has been initiated but not terminated.

Condition

A program status at execution time for which a truth value can be determined. Where the term "condition" (condition-1, condition-2, ...) appears in these language specifications in or in reference to a general format, it is a conditional expression for which a truth value can be determined. It consists of either: (1) a simple condition optionally parenthesized, or (2) a combined condition consisting of simple conditions, logical operators, and parentheses.

Condition-Name

Either a user-defined word that assigns a name to a subset of values that a conditional variable can assume or a user-defined word assigned to a status of a switch or device. When "condition-name" is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a condition-name and qualifiers, subscripts, and indexes, as required for uniqueness of reference.

Condition-Name Condition

The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the value set attributed to a condition-name associated with the conditional variable.

Conditional Expression

A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement. (See Simple Condition and Complex Condition.)

Conditional Statement

A statement specifying that the truth value of a condition is to be determined and that the subsequent action of the object program depends upon the truth value.

Conditional Variable

A data item whose value(s) has a condition-name assigned to it.

CONFIGURATION SECTION

A section of the Environment Division that describes overall specifications for source and object computers.

Connective

A reserved word that:

1. Associates a data-name, paragraph-name, condition-name or text-name with its qualifier.
2. Links two or more operands written in a series.
3. Forms conditions (logical connectives). (See Logical Operator.)

Counter

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

Currency Sign

The $ character in the COBOL character set.

Currency Symbol

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

Current Record

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

Data Clause

A clause in a data-description-entry in the Data Division of a COBOL program.

Data-Description-Entry

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

Data Item

A unit of data (excluding literals) defined by the COBOL program.

Data-Name

A user-defined word that names a data item described in a data-description-entry. When used in the general formats, data-name represents a word that must not be reference-modified, subscripted, indexed, or qualified unless specifically permitted by the rules of the format.

Declarative-Sentence

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

Declaratives

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

De-edit

The logical removal of all editing characters from a numeric edited data item to determine its unedited numeric value.

Delimiter

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

Descending Key

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

Digit Position

The amount of physical storage required to store a single digit. The amount can vary, depending on the usage specified in the data-description-entry that defines the data item. If the data-description-entry specifies that usage is DISPLAY, then a digit position is synonymous with a character position.

Division

A collection of zero, one, or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four divisions in a COBOL program:

IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE

Division Header

A combination of words, followed by a separator period, that indicates the beginning of a division. The division headers in a COBOL program are:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING [data-name-1] ...] .

Dynamic Access

An access mode in which specific logical records can be obtained from or placed into a mass storage file in a non-sequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement. (See Random Access; Sequential Access.)

Editing Character

A single character or a fixed 2-character combination belonging to the following set:

B	space	Z	zero suppress
0	zero	*	check protect
+	plus	$	currency sign
-	minus	,	comma (decimal point)
CR	credit	.	period (decimal point)
DB	debit	/	stroke (virgule, slash)

Elementary Item

A data item that is described as not being further logically subdivided.

Empty Set

A set containing no member records.

End of Procedure Division

The physical position of a COBOL source program after which no further procedures appear.

Entry

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

Environment Clause

A clause that appears as part of an Environment Division entry.

Execution Time

(See Object Time.)

Extend Mode

The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file, and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

External Switch

A hardware or software device used to indicate that one of two alternate states exist.

Figurative Constant

A compiler-generated value referenced by using certain reserved words.

File

A collection of records.

File Clause

A clause that appears as part of a File Description (FD) entry in the Data Division of a COBOL program.

FILE-CONTROL

The name of an Environment Division paragraph where the data files for a given source program are declared.

File-Description-Entry

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

File-Name

A user-defined word that names a file described in a file-description-entry within the File Section of the Data Division.

File Organization

The permanent logical file structure established when a file is created.

FILE SECTION

The section of the Data Division that contains file-description-entries and their associated record descriptions.

Fixed-Length Record

A record associated with a file whose file-description-entry requires that all records contain the same number of character positions.

Footing Area

The position of the page body next to the bottom margin.

Format

A specific arrangement of a set of data.

Group Item

A data item that is composed of subordinate data items.

High-Order End

The leftmost character of a string of characters.

I–O–CONTROL

The name of an Environment Division paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

I–O Mode

The state of a file after execution of an OPEN statement, with the I–O phrase specified for that file, and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

Identifier

A syntactically correct combination of a data-name, reference modifier and qualifiers, subscripts and indexes, as required for uniqueness of reference, that names a data item. The rules for "identifier" associated with the general formats may, however, specifically prohibit reference modification, qualification, subscripting, or indexing.

Imperative-Statement

A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative-statement can consist of a sequence of imperative-statements.

Index

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

Index Data Item

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

Index-Name

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

Indexed Data-Name

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

Indexed File

A file with indexed organization.

Indexed Organization

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

Input File

A file that is opened in the input mode.

Input Mode

The state of a file after execution of an OPEN statement with the INPUT phrase specified for that file and before the execution of a CLOSE statement without the file REEL or UNIT phrase.

Input-Output File

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

INPUT-OUTPUT SECTION

The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Integer

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term appears in general formats, it must not be a numeric data item, must not be signed, and must not be zero unless explicitly allowed by the rules of the format.

Intermediate Data Item

A signed numeric data item that contains the results developed during an arithmetic operation before the final result is moved to the resultant-identifier, if any.

INVALID KEY Condition

At object time, a condition caused when the specific value of the key associated with an indexed or relative file is determined to be invalid.

Key

A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key of Reference

The prime or alternate key currently used to access records in an indexed file.

Key Word

A reserved word needed when the format in which the word appears is used in a source program.

Level Indicator

Two alphabetic characters that identify a specific type of file or a position in a hierarchy.

Level-Number

A user-defined word, expressed as a 1 or 2 digit number, which indicates the hierarchical position of a data item or the special properties of a data-description-entry. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77 and 88 identify special properties of a data-description-entry.

Library

A file containing library text that can be included in a COBOL source program by the COPY verb.

Library-Name

A user-defined word naming a COBOL library for compiler use in a given source program compilation.

Library Text

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

LINAGE-COUNTER

A special register whose value points to the current position in the page body.

Line

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

LINKAGE SECTION

The section in the Data Division of the called program that describes data items available from the calling program. These items may be referred to by the calling and the called program.

Literal

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

Logical Operator

One of the reserved words AND, OR, or NOT. In the formation of a condition, AND or OR (or both) can be used as logical connectives. NOT can be used for logical negation.

Logical Page

A conceptual entity consisting of the top margin, the page body, and the bottom margin.

Logical Record

The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group item.

Low-Order End

The rightmost character of a string of characters.

Mass Storage

A storage medium where data can be organized and maintained in a sequential and nonsequential manner.

Mass Storage File

A collection of records assigned to a mass storage medium.

Mnemonic-Name

A user-defined word associated in the Environment Division with a specific implementor-name.

Native Character Set

The character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence

The collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Combined Condition

The NOT logical operator immediately followed by a parenthetical combined condition.

Negated Simple Condition

The NOT logical operator immediately followed by a simple condition.

Next Executable Sentence

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

Next Executable Statement

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

Next Record

The record which logically follows the current file record.

Next Record Pointer

A conceptual entity that either points to the next logical record, indicates the AT END condition, or is set to indicate that no valid next record has been established.

Nonnumeric Item

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

Nonnumeric Literal

A literal bounded by quotation marks. The string of characters can include any character in the computer character set (except certain source program line delimiters), some or all of which may be represented by a symbolic-character-string.

Numeric Character

A character that belongs to the set of digits 0 through 9.

Numeric Item

A data item whose description restricts its contents to a value represented by characters chosen from the digits 0 through 9; if signed, the item can also contain a +, –, or some other representation of an operational sign.

Numeric Literal

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

OBJECT-COMPUTER

The name of an Environment Division paragraph that describes the computer environment in which the object program is executed.

Object Program

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

Object Time

When an object program is executed.

Open Mode

The condition of a file between the time an OPEN statement is issued and the time a CLOSE statement is executed.

Operand

The general definition of operand is a component which is operated upon. In this manual, however, any lower-case word(s) that appears in a statement or entry format may be considered an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign

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

Optional Word

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

Output File

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

Output Mode

The state of a file after an OPEN statement is executed with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

Padding Character

An alphanumeric character that fills the unused character positions in a physical record.

Page

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

Page Body

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

Page Footing

The logical end of a report page.

Page Heading

The logical beginning of a report page.

Paragraph

In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more entries. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header

A reserved word followed by the separator period that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers are:

In the Identification Division:

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

In the Environment Division:

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

Paragraph-Name

A user-defined word that identifies and begins a paragraph in the Procedure Division.

Phrase

An ordered set of one or more consecutive COBOL character-strings that forms a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record

(See Block.)

Prime Record Key

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

Procedure

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

Procedure-Name

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

Program-Name

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

Pseudo-File-Name

A user-defined word that names a file residing on a multiple file tape for which no file-description-entry is specified.

Punctuation Character

A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
:	colon
.	period (full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
	space
=	equal sign

Qualified Data-Name

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

Qualifier

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

Random Access

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

Record

(See Logical Record.)

Record Area

A storage area allocated to process the record described in a record-description-entry in the File Section of the Data Division.

Record Description

(See Record-Description-Entry.)

Record-Description-Entry

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

Record Key

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

Record-Name

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

Record Type

The collection of records described by a record-description-entry.

Reference Format

A format that provides a standard method for describing COBOL source programs.

Relation

(See Relational Operator.)

Relation Character

A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition

The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item. (See Relational Operator.)

Relational Operator

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

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

Relative File

A file with relative organization.

Relative Key

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

Relative Organization

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

Repeating Group

A group data item whose description contains an OCCURS clause or a group data item subordinate to a data item whose description contains an OCCURS clause.

Reserved Word

A COBOL word that has special meaning to the compiler; a reserved word must not appear in a program as a user-defined word or system-name.

Resultant-Identifier

A user-defined data item that is to contain the result of an arithmetic operation.

Section

A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header

A combination of words followed by a separator period. It indicates the beginning of a section in the Environment, Data, and Procedure Divisions.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a separator period. The permissible section headers are:

In the Environment Division:

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

In the Data Division:

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

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

Section-Name

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

Segment-Number

A user-defined word that classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers can contain only the characters "0", "1", ..., "9". A segment-number can be expressed either as a 1 or 2 digit number.

Sentence

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

Separator

A character or two contiguous characters used to delimit character strings.

Sequential Access

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

Sequential File

A file with sequential organization.

Sequential Organization

The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Sign Condition

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

Simple Condition

Any single condition chosen from the set:

relation condition
class condition
condition-name condition
switch-status condition
sign condition
(simple-condition)

SOURCE-COMPUTER

The name of an Environment Division paragraph that describes the computer environment in which the program is to be compiled.

Source Program

Although it is recognized that a source program may be represented by other forms and symbols, in this manual it always refers to a syntactically correct set of COBOL statements. A COBOL source program begins with an Identification Division and terminates with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word "program" by itself may be used in place of the phrase "source program".

Special Character

A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
!	exclamation point
#	number sign
%	percent
&	ampersand
'	apostrophe
:	colon
?	question mark
@	commercial at

Special-Character Word

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

SPECIAL-NAMES

The name of an Environment Division paragraph in which hardware devices are related to user-specified mnemonic-names.

Special Registers

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

Standard Data Format

The concept used in describing data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page rather than a form oriented to the manner in which the data is stored internally in the computer or on a particular external medium.

Statement

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

Subprogram

(See Called Program.)

Subscript

An integer whose value identifies a particular element in a table.

Subscripted Data-Name

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

Switch-Status Condition

The proposition, for which a truth value can be determined, that a specified switch, capable of being set to an ON or OFF status, has been set to a specific status.

Symbolic-Character

A group of from one (1) to thirty (30) characters combined from the letters A through Z and the numbers 1 through 9, used in a nonnumeric literal to represent a specific character in a particular character set.

Symbolic-Character-String

A symbolic-character or a group of symbolic-characters that appears within a nonnumeric literal enclosed in quotation marks and separated from each other by either the separator comma or space. Each symbolic-character represents a character within a given character set.

System-Name

A COBOL word used to communicate with the operating environment.

Table

A set of logically consecutive items of data defined in the Data Division of a COBOL program by means of the OCCURS clause.

Table Element

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

Text-Name

A user-defined word that identifies library text.

Text-Word

Any character-string or separator, except space, in a COBOL library.

Top Margin

An empty area that precedes the page body.

Truth Value

The representation of the result of the evaluation of a condition in terms of one of two values: True or False.

Unary Operator

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

Unsuccessful Execution

The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

User-Defined Word

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

Variable

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

Variable-Length Record

A record associated with a file whose file-description-entry permits records to contain a varying number of character positions.

Variable-Occurrence Data Item

A variable-occurrence data item is a table element that is repeated a variable number of times. Such an item must contain a Format 2 OCCURS clause in its data-description-entry or be subordinate to such an item.

Verb

A word that expresses an action to be taken by a COBOL compiler or object program.

Word

A syntactically correct character-string of not more than 30 characters. A user-defined word, system-name, or reserved word.

WORKING-STORAGE SECTION

The section of the Data Division that describes working-storage data items and constants composed either of noncontiguous items or working-storage records or both.

Index

Boldface page numbers indicate primary entries.

VAX-11 COBOL-74 Language Reference Manual
AA-C985A-TE

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

—Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Commercial Engineering Publications MK1-2/H3
DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK N.H. 03054

— Do Not Tear - Fold Here and Tape —

Cut Along Dotted Line

digital

PRINTED IN USA