pdp11

PDP-11
COBOL Language Reference Manual

Order No. AA-1749D-TC

digital

April 1977

This document is intended primarily for reference use. It describes how to use the PDP-11 COBOL compiler, its file structure, data formats, I/O devices, and some of its language features.

PDP-11
COBOL Language
Reference Manual

Order No. AA-1749D-TC

SUPERSESSION/UPDATE INFORMATION: This document supersedes the document of the same name, Order No. DEC-11-LCOBA-C-D, published January 1976.

OPERATING SYSTEM AND VERSION: RSTS/E V06B
RSX/11M V03
IAS V02

SOFTWARE VERSION: PDP-11 COBOL V03

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

digital equipment corporation · maynard, massachusetts

First Printing: November 1973
Revised: July 1974
January 1976
March 1977

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 be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11

CONTENTS

CONTENTS (CONT.)

CONTENTS (CONT.)

CONTENTS (CONT.)

FIGURES

TABLES

PREFACE

This manual describes the COBOL language as it has been implemented in the PDP-11. The goal of PDP-11 COBOL'S implementors was a strict adherence to the 1974 ANSI standard. Furthermore, the organization and textual material in this manual is based on the American National Standard COBOL, X3.23-1974 document. PDP-11 extensions to the ANSI standard are shaded.

Chapter 1 contains the overall language considerations; the reader should be familiar with its contents before using the remaining chapters. Chapters 2 through 5 detail the four major divisions of a COBOL program. Chapter 6 covers the Segmentation module, and Chapter 7 discusses the Library module, which provides a capability for specifying source text that is to be copied from a library file. Appendixes A and B contain the COBOL reserved word list and charts of the ASCII character set.

This manual is a reference manual intended primarily as an accurate presentation of the rules governing the syntax and semantics of all language elements implemented in PDP-11 COBOL. It assumes that the reader has a knowledge of the COBOL language; it is not a tutorial guide for beginning COBOL programmers. Those wishing to learn the COBOL language are referred to the following books:

Farina, Mario V., COBOL Simplified, New Jersey, Prentice Hall, Inc., 1968.

McCameron, Fritz A., COBOL Logic and Programming, Third Edition, Homewood, Illinois, Richard D. Irwin, Inc., 1974.

McCracken, Daniel D. and Garbassi, Umberto, A Guide to COBOL Programming, Second Edition, New York, John Wiley and Sons, Inc., 1970.

McCracken, Daniel D., A Simplified Guide to Structured COBOL Programming, New York, John Wiley & Sons, Inc., 1976.

The COBOL programmer is referred to the PDP-11 COBOL User's Guide and the SORT-11 User's Guide, the companion manuals to this language reference manual. They contain additional information on the compiler, the runtime system, a complete list of the PDP-11 COBOL compiler error messages and the utility programs (SORT, COBRG, and REFORMAT).

ACKNOWLEDGMENT

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 Programming Language 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

OVERALL LANGUAGE ELEMENTS AND TERMINOLOGY

1.1 NOTATIONS USED IN FORMATS AND RULES

This chapter contains general information about the special terms, language elements, and general formats required for an ANSI standard COBOL source program. It describes the documentation strategy used to present the language elements to you and also provides you with a description of the meta language elements which describe the COBOL language. Actual source language statements are discussed in subsequent chapters.

The COBOL language consists of the following elements:

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

These elements combine to form the framework for a COBOL source program. PDP-11 COBOL provides four divisions; the Identification Division, the Environment Division, the Data Division, and the Procedure Division. Each division can consist of zero or more sections containing zero or more paragraphs. Each paragraph can contain one or more sentences, clauses, statements, or entries. Each of these are composed of words made up from characters or character-strings.

The meta language elements, those elements which appear in text but are not actually coded into COBOL source statements, serve only to describe the language in terms of allowable use. These elements, completely described in Section 1.3, are as follows:

- Underlined Words
- Brackets and Braces
- Ellipsis

The COBOL language elements (divisions, sections, paragraphs, etc.) are presented according to the following outline:

1. Each COBOL division begins a separate chapter.
2. Each section, clause or statement constituting a division begins on a new page and constitutes a new section.
3. Each section, clause, or statement is described in subsections as follows:

    A. Name (section, clause, or statement)

    B. Description or function

    C. General Format

    D. Syntax Rules

    E. General Rules

    F. Example (if required)

1.1.1 General Format

A general format depicts the specific arrangement of the elements of a clause, statement, paragraph, etc. These elements are described in Section 1.5, Language Organization. Throughout this document, a format is shown adjacent to information defining a language element. When more than one specific arrangement is permitted, the general format is separated into numbered formats. You must write clauses in the sequence given in the general formats. Optional clauses, if you use them, must also appear in the sequence shown. In certain cases (stated explicitly in the rules associated with a given format) the clauses may appear in sequences other than that shown. Application requirements or restrictions are shown as rules.

1.1.2 Syntax Rules

Syntax rules define the order in which words or elements are arranged to form larger elements such as sentences, clauses, or statements. Syntax rules also impose restrictions on individual words or elements. These rules define how you must write the statements; that is, the order in which each element may appear, and what each represents.

1.1.3 General Rules

General rules define the meaning of an element or the relationship of meanings of a set of elements. They define the semantics of the statement and its effect on execution or compilation.

1.2 LANGUAGE ELEMENTS

The elements which make up a statement, clause, sentence, etc. consist of the COBOL character set, character-strings, COBOL words, reserved words, user-defined words, separators/punctuation, and literals.

1.2.1 COBOL Character Set

The basic and indivisible unit of the COBOL language is the character. The individual characters of the language are concatenated to form separators and character-strings. The set of characters used to form COBOL character-strings and separators includes the letters A through Z, the digits 0 through 9, and the special characters +, -, *, /, **, >, <, and =. (Separators are discussed in Section 1.2.5, Separators.)

NOTE

Special characters are always required when they appear in formats.

1.2.2 Character-Strings

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

1.2.3 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. Within a given source program, these classes are mutually exclusive; moreover a COBOL word may belong to one and only one of these classes.

1.2.3.1 **User-Defined Words** - A user-defined word is a COBOL word that you must supply to satisfy the format of a clause or statement. You must select each character of a user-defined word from the letters A through Z, the digits 0 through 9, and the hyphen (-).

NOTE

Do not use the hyphen as the first or last character of a user-defined word.

There are 12 types of user-defined words:

1. condition-name
2. data-name
3. file-name
4. index-name
5. level-number
6. mnemonic-name
7. paragraph-name
8. program-name
9. record-name
10. section-name
11. segment-number
12. text-name

Each of these user-defined word types is described in the glossary which appears at the end of this manual.

1.2.3.2 **Reserved Words** - A reserved word is a COBOL word that is one of a specific list that may be used in COBOL source programs but only

as specified in the general formats. You can not use a reserved word as a user-defined word; the two are mutually exclusive. (See Appendix A for a complete list of COBOL reserved words).

There are six types of reserved words:

1. Key words

   A key word is required when the format in which the word appears is used in a source program. Within each format, key words are upper case and underlined. Consider the following example.

   COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...

   =arithmetic-expression [;ON SIZE ERROR imperative-statement]

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

2. Optional Words

   Within each format, upper case words that are not underlined are called optional words. You may use or omit these words indiscriminately. The presence or absence of an optional word does not alter the semantics of the COBOL program in which it appears. Consider the previous example; the word ON in this case, is an optional word.

3. Connectives

   There are three types of connectives:

   a. Qualifier connectives that 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 that link two or more consecutive operands, (separator comma) or (separator semicolon).

   c. Logical connectives that are used in the formation of the following conditions:

      AND, OR, AND NOT, OR NOT.

4. Special Registers

   The PDP-11 COBOL compiler provides a reserved word that names and refers to a special register. This word, LINAGE-COUNTER, refers to a compiler generated storage area. It is used to store information produced in conjunction with the use of a specific COBOL feature. LINAGE-COUNTER is described in Section 4.9, The LINAGE Clause.

5. Figurative Constants

Certain reserved words, Figurative Constants, are used to name and refer to specific constant values.

Figurative constant values are generated by the compiler and referenced through the use of the reserved words given below. These words must not be bounded by quotation marks when used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

The figurative constant values and the reserved words used to reference them are as follows:

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's character set.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest ordinal position in the computer's collating sequence (octal 177).
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest ordinal position in the computer's collating sequence (octal 000).
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 is used for readability only.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

1. 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 specified by the figurative constant is repeated character by character (or truncated in the case of ALL literal) on the right until the size of the resultant string is equal to the size in characters of the associated data item. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

2. 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.

A figurative constant may be used wherever a literal appears in a format, except that whenever the literal is restricted to numeric characters, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

Each reserved word that is used to reference a figurative constant value is a distinct character-string with the exception of the construction ALL literal, which is composed of two distinct character-strings.

6. Special-Character Words

The arithmetic operators +, -, *, /, ** and relation characters <,>, and = are reserved words.

1.2.4 Literals

A literal is a character-string whose value is determined by the ordered set of characters of which it is composed. You can also use a figurative constant as a literal. There are two types of literals, numeric, and alphanumeric.

1.2.4.1 **Numeric Literal** - A numeric literal is a character-string of from 1 to 20 characters selected from the digits 0 through 9, the plus sign, the minus sign, and/or the decimal point. The rules for the formation of numeric literals are as follows:

1. A literal must contain at least 1 digit and no more than 18 digits.

2. A 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 is unsigned, it is positive.

3. A literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

The word, integer, appearing in a general format, represents a non-zero, positive, numeric literal with no decimal point.

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is an alphanumeric literal and is treated as such by the compiler.

4. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric. (See Section 4.20, The PICTURE Clause.) The size of a numeric literal is equal to the number of digits specified by the user, including leading zeros, if any.

1.2.4.2 **Alphanumeric Literals** - An alphanumeric literal is a character-string representing from 1 to 132 characters, delimited on both ends by quotation marks and consisting of any allowable character in the computer's character set. An opening quotation mark must be immediately preceded by a space or left parenthesis. A closing quotation mark must be immediately followed by one of the separators (space, comma, semicolon, or right parenthesis) or by the terminator, period.

To represent a single quotation mark character within an alphanumeric literal, two contiguous quotation marks must be used. The value of an alphanumeric literal in the object program is the string of characters 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 rather than separators; all alphanumeric literals are category alphanumeric. (See Section 4.20, The PICTURE Clause.)

1.2.5 Separators

A separator is a string of one or more punctuation characters. The rules for forming separators are:

1. Space

   a. Anywhere a space is used as a separator, more than one space may be used.

   b. A space may immediately precede all separators except the closing quotation mark. Here the space is considered part of an alphanumeric literal, not a separator.

NOTE

The only exception to the above rules is described in Section 1.4, COBOL Source Reference Formats.

c. A space may immediately follow any separator except the open quotation mark. In this case, a following space is considered part of an alphanumeric literal, not a separator.

2. Comma and Semicolon

The punctuation characters, the comma and semicolon, are separators only when immediately followed by a space. You may 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.

3. Right Parenthesis and Left Parenthesis

Left parenthesis and right parenthesis are separators only when used in balanced pairs to delimit subscripts or indices.

4. Quotation Marks

Quotation marks may only be used in balanced pairs to delimit alphanumeric literals. (The rules which govern the format and use of alphanumeric literals are detailed in Section 1.2.4.2, Alphanumeric Literals.)

5. Horizontal Tab

The horizontal tab character is governed by the same rules that govern the space character. It is normally used to vertically align statements or clauses on successive lines of the source program listing. 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 1.4.1, Conventional Reference Format; and 1.4.2, Terminal Reference Format.)

1.2.6 Format Punctuation

The punctuation characters, the comma, semicolon, and period, appear in some formats. Where shown, the comma and semicolon are optional and interchangeable. You can specify a comma where a semicolon is specified or vice versa. The period, however, is mandatory. You must supply a period wherever one is shown. You also must specify a period to terminate a paragraph. (See Section 1.5.3, Paragraph, Paragraph Header, and Paragraph-name.)

1.2.7 Use of Certain Special Characters in Formats

The characters +, -, *, /, **, >, <, and =, when appearing in formats, although not underlined, are required.

1.3 META LANGUAGE ELEMENTS

Meta language elements appear in formats but are not coded into source language statements. They serve only to describe the allowable use of the language elements being described.

1.3.1 Underline

The underline is used to denote reserved key words (upper case words). Key words (upper case underlined words) are required when you use a function of which they are a part. The absence of an underline in an upper case word denotes that the word is optional. You may use or omit the word at your discretion.

NOTE

Upper case words, whether underlined or not, must be spelled correctly.

1.3.2 Brackets and Braces

When brackets, [], enclose a portion of a general format, it denotes that an optional portion that may be included or omitted as needed. Braces, {}, enclosing a portion of a general format denote that you must select one of the options within the braces. Consider the following example:

```
[ MEMORY  SIZE  integer  { WORDS      } ]
                         { CHARACTERS }
                         { MODULES    }
```

The brackets indicate that the entire clause is optional. The braces indicate that if the clause is used, a choice of one of the words vertically stacked within the braces must be specified.

Wherever a choice is required, the possibilities are vertically stacked either within brackets or braces. Consider the following example.

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

The outside brackets indicate that the entire clause is optional. The braces indicate that if the clause is used, a choice of a word vertically stacked within the braces must be made. The inside brackets indicate that you may optionally select a vertically stacked word within.

NOTE

Possibilities vertically stacked between brackets indicate that you have the option of overriding a default condition. The default condition is described in the general rules for the clause.

1.3.3 The Ellipsis

The ellipsis (...) indicates that you may repeat the item preceding it. The preceding item is usually enclosed either by brackets or braces to remove any ambiguity as to which item may be repeated. Consider the following example.

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

The ellipsis following the outside brackets indicates that the entire clause, if used, may be repeated. The ellipsis within the outside brackets and following the item which is enclosed in braces indicates that the item may also be repeated within the clause.

1.4 COBOL SOURCE REFERENCE FORMAT

PDP-11 COBOL provides you with two formats for coding your source programs, conventional and terminal. Both formats are described in terms of character positions in a line on an input medium.

NOTE

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

The conventional format is based on the traditional COBOL format as applied to an 80-column punched card. The terminal format is a DEC-specified format and allows a source line to be shortened by using horizontal tabs and carriage returns. The terminal format is very convenient for use with a context editor and an on-line computer terminal.

NOTE

The PDP-11 COBOL compiler assumes the terminal format as a default when compiling, but is capable of compiling either format. (See the PDP-11 COBOL User's Guide for operational details.)

A reformatting program (REFORMAT) is provided to reformat terminal format programs into conventional rormat for ease in transporting source programs to other COBOL compilers. (The REFORMAT program is described in the PDP-11 COBOL User's Guide.)

1.4.1 Conventional Reference Format

The conventional reference format is based on the traditional COBOL format as it applies to an 80-column punched card. If you choose this format, it is more than likely that you will code your source program on a standard COBOL coding form. For this reason, Figure 1-1 (COBOL Programming Form) is provided as a basis for this discussion.

COBOL PROGRAMMING FORM

PAGE _____ OF _____

PROGRAMMER _____

PROGRAM NAME _____

IDENT 73 80

PAGE SERIAL CONT

← START DIVISION, SECTION, PARAGRAPH OR FILE DESCRIPTION HERE
← START OPERAND OR CONTINUATION LINE HERE

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

Figure 1-1 COBOL Programming Form

1.4.1.1 **Sequence Numbers** - Character positions 1 through 6 of the standard format are reserved for source line sequence numbers. This sequence number field serves only to assist you in locating and editing source lines within a source file or listing. Sequence numbers are ignored by the compiler.

1.4.1.2 **Continuation/Comment Indicator Area** - Character position 7 gives you the ability to direct the compiler to process the source line consistent with the character content of this column. The options available to you are listed below:

Option	Results
blank ( )	Default - The source line is treated normally.
hyphen (-)	Continuation line - The compiler will process this line as a continuation of the previous source line. (See Section 1.4.1.6.)
asterisk (*)	Comment line - The compiler will transfer the contents of this line, as is, to the source listing. No syntax checking is performed on this line. (See Section 1.4.1.8.)
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 contents of the line.

1.4.1.3 **Area A** - Character positions 8 through 11 constitute Area A of the conventional format. This area is reserved for the beginning of division headings (Section 1.5.1), section-names (Section 1.5.2), paragraph-names (Section 1.5.3), level-indicators (Section 1.5.4), and certain level numbers (Section 1.5.4).

1.4.1.4 **Area B** - Character positions 12 through 72 constitute Area B of the conventional format. This area is reserved for all other COBOL text.

1.4.1.5 **Identification Field** - Character position 73 through 80 constitute the identification field, which is for documentation purposes only and has no effect on compilation.

1.4.1.6 **Continuation of Lines** - Any sentence or entry that requires more than one line must be continued in Area B of the next line.

When you break a word or numeric literal from one line to the next, you must place 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 word or numeric literal being continued.

When you break an alphanumeric literal from one line to the next, you must place a hyphen in character position 7 of the continuation line. You must also precede the first character of the continuation literal with a quotation mark. The literal may begin anywhere within 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 a numeric literal can be continued to another line, and source lines 001040 through 001060 show how an alphanumeric literal can be continued to another line. Finally, source lines 001090 through 001110 show how a sentence can be continued to successive lines.

1.4.1.7 **Blank Lines** - A blank line is blank from character position 7 through 72 and cannot immediately precede a continuation line. Otherwise, a blank line can appear anywhere in the source program.

1.4.1.8 **Comment Lines** - A comment line is any line with an asterisk (*) in character position 7. A comment line can not precede the Identification Division. Otherwise, a Comment line may appear anywhere in a source program.

A comment line may be composed of any of the characters from the full COBOL character set. Comments can begin in Area A or B of the source line. Each comment line will be reproduced on the source listing, but they serve as documentation only. Successive comment lines are allowed, but each must contain an asterisk in character position 7.

NOTE

If a slash character (/) is used instead of an asterisk (*), the results are the same except that the source listing is advanced to the top of the next page before the comment entry is printed.

1.4.1.9 **Short Lines and Tab Characters** - Conventional format source lines may be shortened if a medium other than punched cards is used. This is accomplished by terminating the line by a carriage return, inserting tab characters within the line to replace space characters, or a combination of both.

When the compiler recognizes a carriage return character, it treats it as a redefinition of character position 72. When a tab character is encountered, the compiler generates the required number of space characters consistent with the tab character position on the line. Tab stops are set within the compiler at character positions 7, 8, 12, 20, 28, 36, 44, 52, 60, 68, and 73. Consider the following example.

NOTE

(RET) = carriage return character

(TAB) = 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).
```

1.4.2 Terminal Reference Format

Terminal reference format is the PDP-11 COBOL default format. It makes your life easier by providing a format that is easy to use with a computer terminal. Terminal format is shorter and less space consuming than its conventional counterpart. The sequence number and identification fields are eliminated, and the indicator field is combined within Area A.

Tab characters can be used to position source entries within a line, and a line ends at the first occurrence of a carriage return character.

The terminal reference format for a source line is represented as follows:

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

NOTE

Continuation line (-), comment line (*), and skip to top of page (/) indicator characters, when used, must be placed in character position 1.

For the terminal format, Area A and Area B contain the same kinds of source entries as their conventional format counterparts. (See Sections 1.4.1.3 and 1.4.1.4.)

Like the conventional format, tab characters, when encountered by the compiler, will generate a commensurate 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.

1.5 LANGUAGE ORGANIZATION

A COBOL program is organized by division, divisions are organized by sections, sections are organized by paragraphs, and paragraphs by sentences, statements, clauses, entries, etc. Each of these divisions, sections, and paragraphs are composed of headers followed by source text. The following sections describe these headers and what source text comprises each.

1.5.1 Division Header

A division header is a combination of words followed by a period that indicates the beginning of a division. The division headers for a PDP-11 COBOL program in their order of appearance are:

```
IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.
```

A division header must start in Area A. After the division header, no text may appear before the following section header, paragraph header, or paragraph-name. The only exception is that the key word DECLARATIVES followed by a period may appear after the Procedure Division header.

1.5.2 Section Header

A section header is a combination of words followed by a period that 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 the word SECTION followed by a period. In the Procedure Division, a section header is composed of a user-defined word followed by the word SECTION followed by a 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

```
user-name SECTION
```

The section header must start in Area A. After the section header, no text may appear before the following paragraph header or paragraph-name. The only exception is the USE sentence in the Procedure Division.

A section consists of paragraphs in the Environment and Procedure Divisions, and data entries in the Data Division.

1.5.3 Paragraph, Paragraph Header, Paragraph-name

A paragraph in the Procedure Division consists of a paragraph-name followed by a period and zero, one, or more entries. In the Identification and Environment Divisions, a paragraph consists of a paragraph header followed by zero, one, or more entries.

A paragraph header consists of a reserved word followed by a period. A paragraph header indicates the beginning of a paragraph. 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-name is a user-defined word followed by a period. It identifies and begins a paragraph in the Procedure Division.

A paragraph header or paragraph name starts in Area A of the first source line following a division or section. The first sentence of a paragraph begins either on the same line as the paragraph header or paragraph-name 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 sentence or entry or in Area B of the next non-blank line that is not a comment line. Sentences that are too long to fit on one line may be continued on successive lines. (See Section 1.4.1.6.)

1.5.4 Data Division Entries

Each Data Division entry begins with a level indicator or a level-number, followed by a space, followed by the name of a data item, followed by a sequence of independent descriptive clauses. Each clause, except the last clause of an entry, may be terminated by the separator semicolon or the separator space. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level-number.

The only level indicator is FD.

In those Data Division entries that begin with a level indicator, the level indicator begins in Area A followed by a space and followed in area B with its associated data-name and appropriate descriptive information.

Those Data Division entries that begin with level-numbers are called data description entries.

A level-number has a value taken from the set of values 1 through 49, 66, 77, and 88. 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. At least one space must separate a level-number from the word following the level-number.

In those data description entries that begin with a level-number 01, 66, or 77, the level-number begins in Area A followed by a space and followed in area B by its associated record-name and appropriate descriptive information.

Successive data description entries may have the same format as the first or may be indented according to level-number. The entries in the output listing are indented only if the input is indented. Indentation does not affect the magnitude of a level-number.

When level-numbers are to be indented, each new level-number may begin any number of spaces to the right of margin A. The extent of indentation to the right is determined only by the width of Area B.

1.5.5 Declaratives

The key words DECLARATIVES and END DECLARATIVES, that precede and follow, respectively, the declaratives portion of the Procedure Division must appear on a line by themselves. Each must begin in Area A and be followed by a period and a space.

CHAPTER 2

IDENTIFICATION DIVISION

2.1 GENERAL DESCRIPTION

The Identification Division must be included in every COBOL source program. It identifies the source program and the resultant output listing. In addition, you may include the date the program was written and such other information as desired under the paragraphs in the general format shown below.

2.2 ORGANIZATION

Fixed paragraph names identify the type of information contained in the paragraph. You must specify the name of the program in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional; if included, they must be presented in the order shown by the general format below.

Structure

The following is the general format of the paragraphs in the Identification Division, as well as a definition of the order of presentation of the source program. Sections 2.3 and 2.4 define the PROGRAM-ID paragraph and the DATE-COMPILED paragraph. Although the other paragraphs are not defined, each general format is formed in the same manner.

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] ...]

Syntax Rules

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

General Rules

The comment-entry may be any combination of the characters from the full character set. The continuation of the comment-entry by using the hyphen in the continuation indicator area is not permitted; however, the comment-entry may be contained on one or more lines.

2.3 THE PROGRAM-ID PARAGRAPH

Function

The PROGRAM-ID paragraph identifies the program.

General Format

PROGRAM-ID. program-name.

Syntax Rules

The program-name must contain from one to nine characters from the set A through Z and 0 through 9. The hyphen is not allowed. Only the first six characters are meaningful.

General Rules

1. The PROGRAM-ID paragraph must contain program name and must be present in every program.

2. A program-name is a user-defined word that identifies a COBOL program. Program-names may not exceed nine characters in length (six characters under RSTS/E) and may not contain a hyphen. The program-name identifies the object program entry point.

DATE-COMPILED

2.4 THE DATE-COMPILED PARAGRAPH

Function

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing.

General Format

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

Syntax Rules

The comment-entry may be any combination of characters from the full character set. Continuation of the comment-entry by using the hyphen in the continuation indicator area is not permitted; however, the comment-entry may be contained on one or more lines.

General Rules

1. The paragraph-name, DATE-COMPILED, causes the current date to be inserted during program compilation. 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.

Example

DATE-COMPILED paragraph before compilation

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BAGINS.
AUTHOR. BILBO BAGINS.
DATE-COMPILED. TODAY.
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. PDP-11.
```

DATE-COMPILED paragraph after compilation.

```
00001       IDENTIFICATION DIVISION.
00002       PROGRAM-ID. BAGINS.
00003       AUTHOR. BILBO BAGINS.
00004       DATE-COMPILED. TODAY.
00005           27-OCT-76 .
00006       ENVIRONMENT DIVISION.
00007       CONFIGURATION SECTION.
00008       SOURCE-COMPUTER. PDP-11.
```

CHAPTER 3

ENVIRONMENT DIVISION

3.1 GENERAL DESCRIPTION

The Environment Division specifies a standard method of expressing those aspects of a COBOL program that are dependent upon the physical characteristics of a specific computer. The division allows you to specify the configuration of the compiling computer and object computer. You can also specify information about input-output control, special hardware characteristics, and control techniques.

NOTE

The Environment Division must be included in every COBOL source program.

3.2 ORGANIZATION

Two sections make up the Environment Division: the Configuration Section and the Input-Output Section.

The Configuration Section describes the source computer and object computer characteristics. This section is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run; and the SPECIAL-NAMES paragraph, which relates the specific compiler features available to the mnemonic-names used in the source program.

The Input-Output Section describes the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files with external media; and the I-O-CONTROL paragraph, which defines special control techniques to be used in the object program.

3.3 STRUCTURE

The general format of the sections and paragraphs in the Environment Division, in the in order of presentation in the source program, follows:

```
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] ]
```

CONFIGURATION-SECTION SOURCE-COMPUTER

3.4 CONFIGURATION SECTION

3.4.1 The SOURCE-COMPUTER Paragraph

Function

The SOURCE-COMPUTER paragraph identifies the computer in which the program is to be compiled.

General Format

SOURCE-COMPUTER. PDP-11.

General Rules

This paragraph is for documentation only.

OBJECT-COMPUTER

3.4.2 The OBJECT-COMPUTER Paragraph

Function

The OBJECT-COMPUTER paragraph identifies the computer in which the program is to be executed.

General Format

```
OBJECT-COMPUTER.    PDP-11  [ , MEMORY SIZE integer  { WORDS      } ]
                            [                       { CHARACTERS } ]
                            [                       { MODULES    } ]

     [, PROGRAM COLLATING SEQUENCE IS alphabet-name]

     [, SEGMENT-LIMIT IS segment-number].
```

Syntax Rule

Segment-number must be an integer ranging in value from 00 through 49.

General Rules

1. The MEMORY SIZE clause is for documentation purposes only.

2. The PROGRAM COLLATING SEQUENCE clause is for documentation purposes only. The native collating sequence is ASCII.

3. Use the SEGMENT-LIMIT clause to specify a number from which the compiler can determine which program segments are overlayable or non-overlayable. (Program segmentation is described in Chapter 6.)

   a. Program segments which have a segment number equal to or greater than the segment limit are overlayable segments.

   b. Program segments which have a segment number less than the segment limit are non-overlayable segments.

4. When the SEGMENT-LIMIT clause is omitted, all program segments are non-overlayable.

SPECIAL-NAMES

3.4.3 The SPECIAL-NAMES Paragraph

Function

The SPECIAL-NAMES paragraph relates PDP-11 COBOL features to user-specified mnemonic-names and alphabet-names (specified in the object-computer paragraph) to character sets and/or collating sequences.

General Format

```
[SPECIAL-NAMES.

  [ { { CARD-READER       }                      }     ]
  [ { { PAPER-TAPE-READER }                      }     ]
  [ { { CONSOLE           }  IS mnemonic name    } ... ]
  [ { { LINE-PRINTER      }                      }     ]
  [ { { PAPER-TAPE-PUNCH  }                      }     ]

[SWITCH integer-1

  { ON STATUS IS condition-name-1  [,OFF STATUS IS condition-name-2] }
  {                                                                  } ] ...
  { OFF STATUS IS condition-name-2 [,ON STATUS IS condition-name-1]  }

      [                    { NATIVE     } ]
      [ Alphabet-name IS   {            } ]
      [                    { STANDARD-1 } ]

      [CURRENCY SIGN IS literal-2]

      [DECIMAL-POINT IS COMMA].  ]
```

Syntax Rules

1. The SPECIAL-NAMES paragraph is required only if mnemonic-names, condition-names, alphabet-names, the DECIMAL-POINT clause, or the CURRENCY SIGN clause are used.

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. The assigned mnemonic-names may be used with the ACCEPT verb in the Procedure Division to transfer data from the device.

2. The names CONSOLE, LINE-PRINTER, and PAPER-TAPE-PUNCH refer to output devices. The assigned mnemonic names may be used with the DISPLAY verb in the Procedure Division to transfer data to the device.

3. The name SWITCH refers to a logical switch to which a value can be assigned by the operator at run-time. The condition-name specified for the ON or OFF STATUS of a switch, can be used in a conditional expression. (See Section 5.6.8, Switch Status Condition.)

4. A condition-name is assigned to a specific value, set of values, or range of values within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

   Condition-names may be defined in the Data Division or in the SPECIAL-NAMES paragraph in the Environment Division where a condition-name must be assigned to the ON STATUS or OFF STATUS of switches that may be set at program execution time.

   A condition-name is used only in conditions as an abbreviation for the relation condition; such use causes the associated conditional variable to be tested for equality with any one of the set of values to which that condition-name is assigned.

5. The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause, the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry (see Section 4.12, The CODE-SET Clause), the alphabet-name clause specifies a character code set.

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

   b. Since the native character code set of the PDP-11 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. The literal is limited to a single character and must not be one 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 *, +, -, ,, ., ;, (,), ", / or =

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

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

3.5 INPUT-OUTPUT SECTION

The Input-output section provides you with the ability to access records of data stored on an external media in various file organizations. The file organizations supported by PDP-11 COBOL and the access methods available for processing them are introduced below. You are advised to refer to the PDP-11 COBOL User's Guide for a complete and in-depth discussion on file organizations and accessing methods.

3.5.1 File Organizations

PDP-11 COBOL provides you with three file organizations:

- Sequential
- Relative
- Indexed

Sequential files are organized such that records are positioned one behind the other. Each record (except the last) has another record following it. The location of any particular record is fixed in relationship to the records preceding and succeeding it. Sequential files can be processed only in a serial fashion. That is, to access a record in the middle of the file, all the records immediately preceding it must be processed.

Relative files, which can only be created on disk storage devices, consist of successively numbered records. Each record is assigned a record number relative to its position in the file. Therefore, 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 within a relative file can be directly accessed by specifying its relative record number. Also, like sequential files, records can be addressed in a serial fashion.

Indexed files, like relative files, can be created only on disk devices. Indexed files are organized such that records are arranged according to a hierarchy of indexes according to a key(s) within each record.

Indexed files have a more complex structure than sequential or relative files. However, instead of being accessed by the specification of a relative record number, indexed files are accessed by the contents of a specified data field(s) (also called keys) within each record. Indexed files can also be accessed in a serial fashion.

3.5.2 Access Modes

File organization determines the access modes that can be used to retrieve and store records within the file. Its organization is fixed when the file is created and it cannot be altered. An access mode, however, is fixed at the time a program opens a particular file. Therefore, the access mode used to process records within a file can differ for each program that opens the file.

PDP-11 COBOL supports three access modes:

- Sequential
- Random
- Dynamic

Sequential access is the process of accessing records from a file in a serial fashion. The first record must be accessed before the second can become available, the second before the third, and so on.

Random access is the process of accessing records individually by the specification of a random record number or a data key.

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

Only certain combinations of file organizations and access mode are permitted. Table 3-1 lists these allowed combinations.

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 relationship between the access modes and the supported file organizations is described in the PDP-11 COBOL User's Guide.

FILE-CONTROL

3.6 THE FILE-CONTROL PARAGRAPH

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format

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

3.6.1 The File-Control Entry

Function

The file-control entry names a file and may specify other file-related information.

General Formats

Format 1

```
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

```
SELECT file-name
    ASSIGN TO literal-1
[ ; RESERVE integer-1 { AREA  } ]
                      { AREAS }
  ; ORGANIZATION IS RELATIVE
[                                { SEQUENTIAL [, RELATIVE KEY IS data-name-1] }  ]
[ ; ACCESS MODE IS               { { RANDOM  }                                  }  ]
[                                { { DYNAMIC }  RELATIVE KEY IS data-name-1     }  ]
[ ; FILE STATUS IS data-name-4 ]
```

Format 3

```
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. The SELECT clause must be specified first in the file control entry. The clauses that follow the SELECT clause may appear in any order.

2. Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the Data Division.

3. Sequential access is assumed if the ACCESS MODE IS clause is not specified.

4. Literal-1 must be an alphanumeric literal.

5. Data-name-4 must be defined in the Working-Storage Section of the Data Division as a 2-character alphanumeric data item.

6. Data-name-1, data-name-2, data-name-3, and data-name-4 may be qualified.

Format 1

7. Sequential organization is assumed if the ORGANIZATION IS SEQUENTIAL clause is not specified.

8. The OPTIONAL phrase may be specified only for input files. Its specification is required for input files that are not always present each time the object program is executed.

Format 2

9. If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be specified for that file.

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 within 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 association of the file referenced by file-name to a storage medium. Literal-1 must be a file specification in command-string format. (See Section 4.11, The VALUE OF Clause).

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

3. If the FILE STATUS clause is specified, a value is placed into the specified 2-character data item (data-name-4) during the execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, or WRITE statement, before any applicable USE procedure is executed. This value indicates to the COBOL program the status 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 of the FILE STATUS data item is known as Status Key 2. It is used to further describe 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

NOTES

1. The possible combinations of Status Keys 1 and 2 are shown in Table 3-2.

2. See Appendix C for a complete listing of the possible File Status Keys and a description of each.

3. See also the Chapters on COBOL-supported file types in the PDP-11 COBOL User's Guide.

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

Status Key 1	Status Key 2									
	No Further Information (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 relatively organized files only.
*** Valid for indexed organized files only.
! File locked by another task
!! Record locked by another task
!!! No sequential READ previous to a REWRITE or DELETE operation

Format 1

4. With this format the RESERVE clause allows you to specify the number of input-output areas allocated for sequential files. The number of input-output areas allocated is equal to the value of integer-1. However, integer-1 cannot be less than 1 or greater than 2. If the RESERVE clause is not specified, a value of 1 is assumed.

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. With this format, the RESERVE clause allows you to specify the number of input-output areas allocated for relative files. The number of input-output areas allocated is equal to the value of integer-1. However, integer-1 cannot be less than 1 or greater than 2. If the RESERVE clause is not specified, a value of 1 is assumed.

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 may be accessed sequentially and/or randomly.

10. All records stored in a relative file are uniquely identified by relative record numbers. 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 user and the Record Management Services.

Format 3

12. With this format, the RESERVE clause allows you to specify the number of input-output areas allocated for indexed files. The number of input-output areas allocated is equal to the value of integer-1. However, integer-1 must be greater than or equal to 2. If the RESERVE clause is omitted, a value of 2 is assumed.

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

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

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

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

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. The data descriptions of data-name-2 and data-name-3 as well as their relative locations within a record must be the same as those used when the file was created. Alternate key specification sequencing must be the same as those used when the file was created.

19. The DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the file records. If the DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated in any of the records file.

I-O-CONTROL

3.7 THE 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 reel.

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. The I-O-CONTROL paragraph is optional.

2. The two forms of the SAME clause (SAME AREA, SAME RECORD AREA) are considered separately in the following:

   More than one SAME clause may be included in a program, however:

   a. A file-name must not appear in more than one SAME AREA clause.

   b. A file-name must not appear in more than one SAME RECORD AREA clause.

   c. If one or more file-names of 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 not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause.

3. The files referenced in the SAME AREA or SAME RECORD AREA clause need not have the same organization or access.

General Rules

1. The SAME AREA clause specifies that two or more files are to use the same memory area during processing. The area being shared includes all buffer areas assigned to the files specified; 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. All the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and 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 rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

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 may be open at one time.

5. Default techniques are used when the APPLY clause is not present; hence, the clause is never required.

   A sequential file may be written with WRITE statements that use the ADVANCING clause to control line spacing. The APPLY PRINT-CONTROL clause may be specified for a printable file if the FD entry does not specify a LINAGE clause. The APPLY PRINT-CONTROL clause supplies a default LINAGE clause. If neither PRINT-CONTROL nor LINAGE is specified for the file, a WRITE statement with the ADVANCING option will cause formatting information to be included in the user-record.

CHAPTER 4

DATA DIVISION

4.1 OVERALL APPROACH

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed falls into three categories:

1. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.

2. That which is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

3. Constants which are defined by the user.

4.1.1 Data Division Organization

The Data Division, which is one of the required divisions in a program, is subdivided into three sections; the File Section, the Working-Storage Section, and the Linkage Section.

The File Section defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions. Record descriptions are written immediately following the file description entry.

The Working-Storage Section describes records and noncontiguous data items that are not part of external data files but are developed and processed internally. It also describes data items whose values are assigned in the source program.

The Linkage Section appears only in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the Working-Storage Section.

4.1.2 Data Division Structure

The following information gives the general format of the sections in the Data Division and defines the order of their presentation in the source program.

```
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   ] ...  ]
```

4.2 FILE SECTION

The File Section contains descriptions of files required by the object program.

4.2.1 File-Description-Entry

In a COBOL program the file-description-entry (FD) represents the highest level of organization in the File Section. The File Section header is followed by a file-description-entry consisting of a level indicator (FD), a file-name, and a series of independent clauses. The FD clauses specify the size of the logical and physical records, the presence or absence of label record, the value of DEC-defined label items, and the names of the data records that make up the file. The entry itself is terminated by a period.

4.2.2 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 by a data-name if required, followed by a series of independent clauses as required. A record description has a hierarchical structure; therefore, the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries.

4.3 WORKING-STORAGE SECTION

The Working-Storage Section is composed of the section header, followed by data description entries for noncontiguous data items and/or record description entries. Each Working-Storage Section data name must be unique.

4.3.1 Noncontiguous Working-Storage

Items and constants in Working-Storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need more subdividing. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry.

4.3.2 Working-Storage Records

Data elements and constants in Working-Storage that bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of record descriptions. All clauses used in record descriptions in the File Section can be used in record descriptions in the Working-Storage Section.

4.3.3 Initial Values

The initial value of any item in the Working-Storage Section, except an index data item, is specified by using the VALUE clause (see Section 4.11) with the data item. The initial value of any index data item is unpredictable.

4.4 LINKAGE SECTION

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

The Linkage Section is used for describing data that is available through the calling program but is 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 of that program. 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 called and calling program always refer to separate indices.

Data items defined in the Linkage Section of the called program may 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 paragraph 3 above.

5. Condition-names and index-names associated with data items that meet any of the above conditions.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by Record Description entries.

4.5 THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

Function

The file description furnishes information about the physical structure, identification, and record names pertaining to a given file.

General Format

```
FD file-name
[; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS    }]
                                           {CHARACTERS}
[; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
  ; LABEL {RECORD IS  } {STANDARD}
          {RECORDS ARE} {OMITTED }
[; VALUE OF ID IS {data-name-1}]
                  {literal-1  }
[; DATA {RECORD IS  } data-name-3 [, data-name-4] ...]
        {RECORDS ARE}
[; LINAGE IS {data-name-5} LINES [, WITH FOOTING AT {data-name-6}]
             {integer-5  }                          {integer-6  }
    [, LINES AT TOP {data-name-7}], [LINES AT BOTTOM {data-name-8}]]
                    {integer-7  }                    {integer-8  }
[; CODE-SET IS alphabet-name]
```

Syntax Rules

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.
2. The clauses that follow the name of the file are optional in many cases, and their order of appearance is immaterial.
3. One or more record description entries must follow the file description entry.

BLOCK CONTAINS

4.6 THE BLOCK CONTAINS CLAUSE

Function

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

General Format

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

Syntax Rules

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

General Rules

1. Integer-1, if present, is ignored.

2. The size of the block may be stated in terms of RECORDS,

   a. If the file is assigned to magnetic tape and the records are fixed in size (see the RECORD CONTAINS clause, Section 4.10), each block except the last block will contain integer-2 records. Integer-1, if present, is ignored.

   b. If the file is assigned to magnetic tape and the records are variable in size (see The RECORD CONTAINS clause, Section 4.10), the value of integer-2 is used to calculate a buffer size by multiplying the largest record size, plus four bytes, by integer-2. The input-output system then blocks or unblocks as many variable sized records from this buffer as will fit.

   c. For a sequential file assigned to a disk device, the values of integer-1, if present, and integer-2 are ignored. There are no unused bytes on any block and the records may span block boundaries.

d. If the file has relative or indexed organization and it is assigned to a directory device, the value of integer-2 is used to calculate the size of the block. This size may or may not be equal to the record size times integer-2 because of overhead bytes. (See the PDP-11 COBOL User's Guide.)

3. The size of the block may be stated in terms of characters,

   a. If the file is assigned to a magnetic tape, the size of the block is the maximum of either,

      (1). Integer-2 bytes

      (2). The size of the largest record (add four overhead bytes for variable length records)

   b. If the file has sequential organization and is assigned to a disk device, the clause is ignored and the records are packed together on each physical block. There are no unused bytes on any block and the records may span block boundaries.

   c If the file has relative or indexed organization, the size of the block 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. (See the PDP-11 COBOL User's Guide.)

4. When the clause is not present, the size of the block is calculated in the following manner:

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

   b. If the file has sequential organization and is assigned to a disk device, the records are packed together on each physical block. There are no unused bytes on any block, and the records may span block boundaries.

   c. If the file has relative or indexed organization, the least number of physical blocks that can contain one record, plus any overhead bytes is the block size.

CODE-SET

4.7 THE CODE-SET CLAUSE

Function

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

General Format

CODE-SET IS alphabet-name

Syntax Rules

1. When the CODE-SET clause is specified for a file, all data in that file must be described as USAGE IS DISPLAY and any signed numeric data must be described with the SIGN IS SEPARATE clause.

2. The CODE-SET clause may only be specified for files whose organization is sequential.

General Rules

1. If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from/to the native character codes. This code conversion occurs during the execution of an input or output operation. (See Section 3.4.3, The SPECIAL-NAMES Paragraph.)

2. If the CODE-SET clause is not specified the native character code set is assumed for data on the external media.

DATA RECORDS

4.8 THE DATA RECORDS CLAUSE

Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

General Format

```
DATA { RECORD IS  } data-name-1 [, data-name-2] ...
     { RECORDS ARE}
```

Syntax Rules

1. Data-name-1 and data-name-2 are the names of data records and must have 01 level-number record descriptions, with the same names, associated with them.

General Rules

1. This clause is documentary only and is never required. The names of the records are not checked against the names appearing in the following 01 record descriptions.

2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

LABEL RECORDS

4.9 THE LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present.

General Format

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

Syntax Rules

This clause is required in every file description entry.

General Rules

1. OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned. OMITTED may be specified only for files that are assigned to non-directory devices.

2. STANDARD specifies that labels exist for the file or the device to which the file is assigned and that the labels conform to the file system label specifications.

3. STANDARD is required for all files assigned to directory devices.

4.10 THE LINAGE CLAUSE

Function

The LINAGE clause may only be used for sequential output files. It is provided as a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

General Format

```
LINAGE IS { data-name-1 } LINES [, WITH FOOTING AT { data-name-2 }]
          { integer-1   }       [                  { integer-2   }]

[, LINES AT TOP { data-name-3 }] [, LINES AT BOTTOM { data-name-4 }]
                { integer-3   }                     { integer-4   }
```

Syntax Rules

1. Data-name-1, data-name-2, data-name-3, 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 integer-1.

4. The value of integer-3, integer-4 may be zero.

General Rules

1. The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values for these functions are zero. If the FOOTING phrase is not specified, the assumed value is equal to integer-1 or the contents of the data item referenced by data-name-1, whichever is specified.

   There is not necessarily any relationship between the size of the logical page and the size of a physical page.

2. The value of integer-1 or the data item referenced by data-name-1 specifies the number of lines that can be written and/or spaced on the logical page. The value must be greater than zero. That part of the logical page in which these lines can be written and/or spaced is called the page body.

3. The value of integer-3 or the data item referenced by data-name-3 specifies the number of lines constituting the top margin on the logical page. The value may be zero.

4. The value of integer-4 or the data item referenced by data-name-4 specifies the number of lines that make up the bottom margin on the logical page. The value may be zero.

5. The value of integer-2 or the data item referenced by data-name-2 specifies the line number within the page body at which the footing area begins. The value must be greater than zero and not greater than the value of integer-1 or the data item referenced by data-name-1.

   The footing area comprises the 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.

6. During the execution of an OPEN statement with the OUTPUT phrase specified, the value of integer-1, integer-3, and integer-4, if specified, will be used at the time the file is opened to specify the number of lines that make up each of the indicated sections of a logical page. The value of integer-2, if specified, will be 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.

7. The values of the data items referenced by data-name-1, data-name-3, and data-name-4, if specified, will be used as follows:

   a. The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, will be used to specify the number of lines that are to constitute each of the indicated sections for the first logical page.

   b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or page overflow condition occurs (see Section 5.42, The WRITE Statement), will be used to specify the number of lines that are to constitute each of the indicated sections for the next logical page.

8. The value of the data item referenced by data-name-2, if specified, at the time an OPEN statement with the OUTPUT phrase is executed for the file, will be used to define the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, it will be used to define the footing area for the next logical page.

9. A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:

   a. A separate LINAGE-COUNTER is supplied for each file described in the File Section whose file description entry contains a LINAGE clause.

   b. LINAGE-COUNTER may be referenced but not modified by Procedure Division statements. Because more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary. LINAGE-COUNTER is implicitly described as PIC S9999 COMP.

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

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

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

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

      (4) The value of LINAGE-COUNTER is automatically reset to one when the device is repositioned to the first line that can be written on for each of the succeeding logical pages. (See Section 5.44, The WRITE Statement.)

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

10. Each logical page is contiguous to the next with no additional spacing provided.

RECORD CONTAINS

4.11 THE 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 size of each data record is completely defined within the record description entry; therefore, this clause never alters the storage allocated to the records in the computer memory.

2. For files with relative organization, record size on the storage media 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 files with sequential or indexed organization, record size on the storage media may be fixed or variable, If the record descriptions for a file yield record sizes that vary in size, the record storage areas allocated on the storage media will vary in size and will be preceded by a byte count word supplied automatically by the Record Management Services.

4. If the record descriptions for a file yield record sizes that are all the same size, the record storage areas allocated on the storage media will be fixed in size and will not be preceded by a byte count word. However, the programmer may force a variable size record format, with the byte count word suffix on each record, by using a RECORD CONTAINS clause with the "integer-1 TO" phrase.

   a. Integer-2 may not be used by itself unless all the 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 are both shown, they refer to the number of characters in the smallest size data record and the number of characters in the largest size data record, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all elementary items, plus any characters required by implicit or explicit synchronization.

c. Except for the use of the RECORD CONTAINS clause to force a variable record storage size on the media, the clause is documentary.

VALUE OF

4.12 THE VALUE OF CLAUSE

Function

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

General Format

```
VALUE OF ID IS  { data-name }
                { literal   }
```

Syntax Rules

1. The VALUE OF ID clause must not be stated when LABEL RECORDS ARE OMITTED is specified.
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 literal or data-name is used to supply information to the Record Management System to properly locate and identify the desired file.
2. For an output file, the value of literal or data-name is used to supply information to the Record Management System to properly create the desired file.
3. The value of literal or data-name is taken as a file specification in command string format.

DATA DESCRIPTION

4.13 DATA DESCRIPTION CONCEPT

To make data as computer independent as possible, the characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. This standard data format is oriented to general data processing applications. It uses the decimal system to represent numbers and the remaining characters in the COBOL character set to describe alphanumeric data items.

4.13.1 Logical Record and File Concept

The approach taken in defining file information is to distinguish between the physical aspects of the file and the conceptual characteristics of the data contained within the file.

4.13.2 Physical Aspects of a File

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

1. The grouping of logical records within the physical limitations of the file medium.

2. The means by which the file can be identified.

4.13.3 Conceptual Characteristics of a File

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit.

A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and bears no direct relationship to the size of the file of information contained on a device.

One or more logical record(s) may be contained within a single physical unit; or, in the case of formatted storage media, a logical record may require more than one physical unit to contain it. There are source language methods available for describing the relationship of

logical records and physical units. When a permissible relationship has been established, control of the accessibility of logical records as related to the physical unit is provided by the interaction of the object program on the hardware and/or software system. In this document, references to records means to logical records, 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 may be grouped into logical records and defined by a series of record description entries.

4.13.4 Record Concepts

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

4.13.5 Concept of Levels

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a record for data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

To refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of one or more groups, etc. Thus, an elementary item may belong to more than one group.

4.13.6 Level-Numbers

A system of level-numbers shows the 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 (not necessarily successive) level-numbers not greater in value than 49. There are special level-numbers, 66, 77, and 88, which are exceptions to this rule (see below). 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 which are immediately subordinate to a given group item must be described using identical level-numbers 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 non-contiguous working storage data items
3. Entries that specify condition-names

Entries that specify RENAMES items and can be used only as described in Format 2 of the data description skeleton have been assigned the special level-number 66.

Entries that specify non-contiguous data items which are not subdivisions of other items and are not themselves subdivided, have been assigned the special level-number 77.

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

4.13.7 Concept of Classes of Data

The five categories of data items (see Section 4.20, The PICTURE Clause) 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 (without editing). 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 at object time as alphanumeric regardless of the class of elementary items subordinate to that group item. The following chart depicts the relationship of the class and categories of 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

4.13.8 Selection of Numeric Character Representation

The value of a numeric item may be represented in either binary or decimal form. The form of representation may be selected with the USAGE clause of the data-description entry; the binary form (with an assumed decimal position) is the COMPUTATIONAL or COMP item (see Section 4.25, The USAGE Clause), and the decimal form (ASCII 8-bit) is the DISPLAY item (see Section 4.25, The USAGE Clause). (DISPLAY, DISPLAY-6, and DISPLAY-7 are all synonymous and may be used interchangeably.)

4.13.9 Algebraic Signs

Algebraic signs fall into two categories: operational signs, which are associated with signed numeric data items to indicate their algebraic properties; and editing signs, which appear on edited reports to identify the sign of the item.

The SIGN clause permits the programmer to state explicitly the location of the operational sign. The clause is optional; if it is not used, operational signs will be represented by a default. (See Section 4.23, The SIGN Clause.)

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

4.13.10 Standard Alignment Rules

The standard rules for positioning data within an elementary item 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 4.17, The JUSTIFIED Clause.)

4.13.11. Item Alignment for Increased Object-Code Efficiency

All COMPUTATIONAL items are automatically SYNCHRONIZED RIGHT and word aligned as follows: if the item is specified as being less than 5 characters long, it will be in 1 machine word; if it is from 5 to 9 characters long, it will be in 2 words; from 10 to 14 characters will be in 3 words, and from 15 to 18 characters will be in 4 words. The SYNCHRONIZED clause (see Section 4.24, The SYNCHRONIZED Clause) may be used to control word alignment of DISPLAY data if desired.

All INDEX data items are automatically SYNCHRONIZED RIGHT and occupy one machine word.

DATA DESCRIPTION - SKELETON

4.14 THE DATA DESCRIPTION - COMPLETE ENTRY SKELETON

Function

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

General Format

Format 1

```
level-number { data-name-1 }
             { FILLER      }

  [; REDEFINES data-name-2]

[; { PICTURE } IS character-string]
   { PIC     }

[; [USAGE IS] { COMPUTATIONAL }]
              { COMP          }
              { DISPLAY       }
              { DISPLAY-6     }
              { DISPLAY-7     }

[; [SIGN IS] { LEADING  } [SEPARATE CHARACTER]]
             { TRAILING }

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

[; { JUSTIFIED } RIGHT]
   { JUST      }

  [; BLANK WHEN ZERO]

  [; VALUE IS literal]

[OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 }
        { integer-2 TIMES                                       }

[ { ASCENDING  } KEY IS data-name-4 [data-name-5] ...] ...
  { DESCENDING }

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

Format 2

```
66 data-name-1; RENAMES data-name-2
                [ {THROUGH} data-name-3 ].
                {THRU   }
```

Format 3

```
88 condition-name; {VALUE IS  } literal-1 [ {THROUGH} literal-2 ]
                  {VALUES ARE}            {THRU   }
   [ , literal-3 [ {THROUGH} literal-4 ] ]... .
                 {THRU   }
```

Syntax Rules

1. The level-number in Format 1 may be any number from 01-49 or 77.

2. The clauses may be written 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 every elementary item except an index data item, in which case use of this clause is prohibited.

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 which can neither be subscripted nor indexed unless specifically permitted by the rules of that format. Data-names need not begin with an alphabetic character; the alphabetic characters may be positioned anywhere within the data-name. Qualification is permitted; therefore, all data-names need not be unique.

2. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses must not be specified except 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. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains 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

   d. A level 66 item.

4.15 THE BLANK WHEN ZERO CLAUSE

Function

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

General Format

BLANK WHEN ZERO

Syntax Rules

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric edited or numeric. (See Section 4.20, The PICTURE Clause.)

General Rules

1. When the BLANK WHEN ZERO clause is used as a receiving field for a numeric value, the items 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.

DATA-NAME/FILLER

4.16 THE 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 Rules

1. 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 Rules

1. The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to explicitly. However, the key word FILLER may be used as a conditional variable because such use does not require explicit references to the FILLER item, but to its value. (See General Rule 3 in Section 4.14 for an explanation of the condition variable.)

4.17 THE JUSTIFIED CLAUSE

Function

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

General Format

$$\left\{\begin{matrix}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{matrix}\right\} \quad \text{RIGHT}$$

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. When a receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.
2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. (See Section 4.13.10, Standard Alignment Rules.)

LEVEL-NUMBER

4.18 LEVEL-NUMBER

Function

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

General Format

level-number

Syntax Rules

1. A level-number is required as 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 there is no real concept of level:

   a. Level-number 77 is assigned to identify noncontiguous working storage data items or noncontiguous data items and can be used only as described by Format 1 of the data description skeleton. (See Section 4.14, The Data Description - Complete Entry Skeleton.)

   b. Level-number 66 is assigned to entries that define RENAMES items and can be used only in Format 2 of the data description skeleton. (See Section 4.14, the Data Description-Complete Entry Skeleton.)

   c. Level-number 88 is assigned to entries that define condition-names associated with a conditional variable and can be used only in Format 3 of the data description skeleton. (See Section 4.14, The Data Description - Complete Entry Skeleton.)

3. Multiple level 01 entries subordinate to an FD level indicator represent implicit redefinitions of the same area.

4.19 THE 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 indices.

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, ... may be qualified.

4. Data-name-2 must either be 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 is not defined elsewhere, because its allocation and format are dependent on the hardware and, not being data, cannot be associated with any data hierarchy.

7. A data description entry that contains Format 2 of the OCCURS clause may only be followed, within that record description, by data description entries which are subordinate to it.

8. The OCCURS clause cannot be specified in a data description entry that:

   a. Has a 01, 77, or an 88 level-number.

   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 within the range of the first character position defined by the data description entry containing the OCCURS clause and 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 within 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. There must not be any entry that contains an OCCURS clause 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 within the program.

General Rules

1. The OCCURS clause is used in defining 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 is referred to in a 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 represents the exact number of occurrences.

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

   This format specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. 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 makes the contents of data items, whose occurrence numbers now exceed the value of the data item referenced by data-name-1, unpredictable.

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

5. The KEY IS phrase is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, etc. The ascending or descending 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.

PICTURE

4.20 THE PICTURE CLAUSE

Function

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

General Format

$$\left\{\begin{matrix}\underline{\text{PICTURE}} \\ \underline{\text{PIC}}\end{matrix}\right\} \text{ IS character-string}$$

Syntax Rules

1. A PICTURE clause can be specified only at the elementary item level.
2. A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.
3. The maximum number of characters allowed in the character-string is 30.
4. The PICTURE clause must be specified for every elementary item except an index data item, where the clause is prohibited.
5. PIC is an abbreviation for PICTURE.
6. The asterisk when used as the zero suppression symbol, and the BLANK WHEN ZERO clause may not appear in the same entry.

General Rules

1. There are five categories of data that 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 only contain the symbols A and B, and

b. Its contents when represented in standard data format must be any combination of the 26 letters of the alphabet and the space from the COBOL character set.

3. To define an item as numeric:

a. Its PICTURE character-string can only contain 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.

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 may also contain a +, -, or other representation of an operational sign (See Section 4.23, The 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 X's. A PICTURE character-string which contains all A's or all 9's does not define an alphanumeric item.

b. Its contents when represented in standard data format are allowable characters in the full 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 full 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, ,, ., *, +, -, CR, DB, and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and 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, *, +, ,, ., -, 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 one of the numerals.

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

8. The functions of the symbols used to describe an elementary item are explained 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 in the character-string represents a character position into which the space character will be inserted.

P Each P indicates an assumed decimal scaling position. It is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items. The scaling position character P can appear only to the left or right as a continuous string of Ps within a PICTURE description; because the scaling position character 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. The character P and the insertion character . (decimal point) cannot both occur in the same PICTURE character-string. If, in any operation involving conversion of data from one form of internal representation to another, 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 letter S is used in a character-string to indicate 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 (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause that specifies the optional SEPARATE CHARACTER phrase. (See Section 4.23, The SIGN Clause.)

V The V is used in a character-string to indicate the location of the assumed decimal point. It may appear only once in a character-string. The V does not represent a character position and, therefore, is not counted in 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 is used to represent a character position that contains any allowable character from the full character set.

Z Each Z in a character-string may 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 the size of the item.

9 Each 9 in the character-string represents a character position that contains a numeral and is counted in the size of the item.

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

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

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

. When the character . (period) appears in the character-string it is an editing symbol that represents the decimal point for alignment purposes and, in addition, represents a character position into which the character . will be inserted. The character . is counted in the size of the item. For a given program, the functions of the period and comma are exchanged if the DECIMAL-POINT IS COMMA clause is stated in the SPECIAL-NAMES paragraph. In this exchange, the rules for

the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause. The insertion character . must not be the last character in the PICTURE character-string.

+, -, CR, DB These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol 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) in the character-string represents a leading numeric character position into which an * will be placed when the content of that position is 0. Each * is counted in the size of the item.

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

Editing Rules

1. There are two general methods of performing editing in the PICTURE clause, 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 may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given 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 rules in rule 3, below.

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

4. Simple Insertion Editing. The , (comma), B (space), 0 (zero), and / (stroke) are used as the insertion characters. The insertion characters are counted in 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 to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. 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 only 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 and they must represent the rightmost character positions that are counted in the size of the item. The symbols + or -, when used, must be either the leftmost or rightmost character position to be counted in 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 occupied in the PICTURE character-string. Editing sign control symbols produce the following results, depending upon the value of the data item:

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. As such they are mutually exclusive in a given PICTURE character-string.

   Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the fixed insertion symbols or have fixed insertion characters immediately to the right of this string. These simple insertion characters are part of the floating string.

   The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols 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 may replace all the characters at or to the right of this limit.

   In a PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

   If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that 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 preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, 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 when the insertion character is only to the left of the decimal point.

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 alphabetic character Z or the character * (asterisk) as suppression symbols in a PICTURE character-string. 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. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

   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.

   If all numeric character positions in the PICTURE character-string are represented by suppression symbols 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 *, (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 chart 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 may precede, in a given character-string, 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.

Non-floating insertion symbols + and -, floating insertion symbols Z, *, +, -, and cs, and other symbol P appear twice in the following PICTURE character precedence chart. 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 in the chart represents its use to the right of the decimal point position.

PICTURE Character Precedence Chart

	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

REDEFINES

4.21 THE REDEFINES CLAUSE

Function

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

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. Level-number and data-name-1 are not part of the REDEFINES clause.

Syntax Rules

1. The REDEFINES clause, when specified, must immediately follow data-name-1.

2. The level-numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88. (Level 77 items may 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 may 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 may 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 may not 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 4.19, The OCCURS Clause.)

5. No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 may 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 that the data item referenced by data-name-2 contains. It is important to observe 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.

RENAMES

4.22 THE RENAMES CLAUSE

Function

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

General Format

```
66  data-name-1;  RENAMES data-name-2 [ { THROUGH } data-name-3 ].
                                       { THRU    }
```

NOTE

Level-number 66, data-name-1 and the semicolon are shown in the above format to improve clarity. Level-number and data-name-1 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 may have an OCCURS clause in its data description entry nor 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 may be qualified.

6. The words THRU and THROUGH are equivalent.

7. None of the items within the range, including data-name-2 and data-name-3, if specified, can be an item whose size is variable as defined in the OCCURS clause (see Section 4.19).

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 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 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 item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

SIGN

4.23 THE SIGN CLAUSE

Function

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

General Format

$$[\underline{\text{SIGN}}\ \text{IS}] \left\{ \begin{matrix} \underline{\text{LEADING}} \\ \underline{\text{TRAILING}} \end{matrix} \right\} [\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

Syntax Rules

1. The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S, or a group item containing at least one such numeric data description entry.

2. The numeric data description entries to which the SIGN clause applies must be described as USAGE IS DISPLAY.

3. At most one SIGN clause may apply to any given numeric data description entry.

General Rules

1. The optional SIGN clause, if present, 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, but neither the representation nor, necessarily, the position of the operational sign.

2. A numeric data description entry whose PICTURE contains the character S, but to which no optional SIGN clause applies, has an operational sign. The character S does not specify the representation nor, necessarily, the position of the operational sign. In this (default) case, the sign is a part of the right-most (trailing) digit (much like an overpunch) within the item.

3. If the optional 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 are shown in the following chart for all combinations of the numeric digits, and the positive and negative sign values.

		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 optional 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 purposes of computation or comparisons, conversion takes place automatically.

SYNCHRONIZE

4.24 THE SYNCHRONIZED CLAUSE

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on the PDP-11 memory word. (See Section 4.13.11, Item Alignment for Increased Object-Code Efficiency.)

General Format

$$\left\{\begin{matrix}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{matrix}\right\} \quad \left\{\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right\}$$

Syntax Rules

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

General Rules

1. This clause specifies that the subject data item is to be aligned on PDP-11 words such that no other data item occupies any of the words delimiting this data item. If the number of character positions required to store this data item is odd, the unused character is not used for any other data item. Such 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 be positioned such that it will begin at the even byte address of the PDP-11 word.
4. SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the odd byte address of the PDP-11 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 in determining 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. When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is SYNCHRONIZED. If the size of a group item that contains an OCCURS clause is odd, and one or more items within the group are SYNCHRONIZED, a fill byte will be added to the group following the last item, causing the group size to become even and ensuring that each occurrence of an item in the group will map into PDP-11 words in the same manner as the first occurrence of that item. (See Section 4.19, The OCCURS Clause.)

8. All COMP items and all INDEX items are automatically synchronized and occupy an integral number of words; for further information (see Section 4.25, The USAGE Clause).

9. All record descriptions in both the File Section and Working-Storage Section and all non-contiguous data items in the Working-Storage Section are automatically SYNCHRONIZED.

USAGE

4.25 THE USAGE CLAUSE

Function

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

General Format

```
[USAGE IS] { COMPUTATIONAL }
           { COMP          }
           { DISPLAY       }
           { DISPLAY-6     }
           { DISPLAY-7     }
           { INDEX         }
```

Syntax Rules

1. The PICTURE character-string of a COMPUTATIONAL item can contain only 9s, the operational sign character S, the implied decimal point character V, one or more Ps. (See Section 4.20, The PICTURE Clause.)

2. COMP is an abbreviation for COMPUTATIONAL.

3. DISPLAY, DISPLAY-6 and DISPLAY-7 are all equivalent.

4. 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.

5. 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 the USAGE clause 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 storage of the PDP-11. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to.

3. A COMPUTATIONAL item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group item itself is not COMPUTATIONAL (cannot be used in computations).

4. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which 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 the USAGE IS INDEX clause, the elementary items in the group are all index data items. The group 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 no conversion will take place.

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 COMPUTATIONAL item is a binary value with an assumed decimal point that is automatically SYNCHRONIZED and stored in memory as follows: if the item is specified as being less than five characters long (PIC S9 to PIC S9(4)), it will be in one computer word; if it is specified as being from five to nine characters long (PIC S9(5)) to PIC S9(9)), it will be in two words; from 10 to 14 will be in three words, and from 15 to 18 will be in four words.

   The representation of the binary value in one to four PDP-11 words is independent of the presence of V or one or more Ps in its PICTURE character-string. The binary value of a COMPUTATIONAL 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 properly 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 the representation is binary.

8. A DISPLAY item is a string of bytes stored in PDP-11 memory as two bytes per word. The item may begin in the even address byte or the odd address byte subject to the implicit or explicit synchronization. (See Section 4.24, The SYNCHRONIZED Clause.)

9. 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 within record descriptions, they may cause automatic fill bytes to be supplied.

VALUE

4.26 THE 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  }
{VALUES ARE} literal-1 [{THROUGH} literal-2]
                       {THRU   }
    [, Literal-3 [{THROUGH} Literal-4]] ...
                 {THRU   }
```

Syntax Rules

1. The words THRU and THROUGH are equivalent.

2. A signed numeric literal must have associated with it a signed numeric PICTURE character-string.

3. All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause and must not have a value which would require truncation of nonzero 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 within 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, the literal is aligned in the data item according to the standard alignment rules. (See Section 4.13.10, 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 paragraph 4.13.10, The Standard Alignment Rules.) Editing characters in the PICTURE clause are included in determining the size of the data item (see Section 4.20, The PICTURE Clause) but have no effect on initialization of the data item.

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

2. A figurative constant may 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 1.2.3.1 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 may 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 may 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 contains an OCCURS clause or in an entry that is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries. (See Section 4.19, The OCCURS Clause.)

3. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause or in an entry that is subordinate to an entry containing 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 within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

5. The VALUE clause must not be written for a group containing items with descriptions including 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 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 preceded by the 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, the USE statement.)

5.1.2 Procedures

Within the Procedure Division a procedure is composed of either 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.)

5.1.3 Execution

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 THE 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. Their descriptions, however, 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 may appear more than once in the USING phrase of a CALL statement.

Data items defined in the Linkage Section of the called program may 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 paragraph 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 statements, compiler directing statements, and imperative statements.

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

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.

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 consists of a compiler directing verb and its operands. The compiler directing verbs are COPY, and USE (see Sections 7 and 5.41 respectively). A compiler directing statement causes the compiler to take a specific action during compilation.

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 may consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT		
ADD(1)	GO	SET
ALTER		
CALL	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.

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.

Categories of Statements

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) 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.

5.4.7 Specific Statement Formats

The specific statement formats, and a detailed discussion of the restrictions and limitations associated with each, appear in alphabetic sequence beginning at Section 5.8.

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 by those names associated with level-number 01, then names associated with level-number 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, may 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, no name can be both a data-name and 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}
```

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 referred to 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 may 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 4.19, The OCCURS Clause in the Data Division.)

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 may be qualified but not subscripted.

The subscript may 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 the balanced pair of separators, left parenthesis and right parenthesis, 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:

```
{data-name     }
{condition-name} (subscript-1 [, subscript-2 [, subscript-3]])
```

5.4.8.3 Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY phrase in the definition of a table. 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 either a SET, a SEARCH ALL, or a 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 by the operator + or -, followed by an unsigned integer numeric literal, all delimited by the balanced pair of separators left parenthesis and right parenthesis, 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:

```
{data-name     } ( {index-name-1 [{+} literal-2]}
{condition-name}   {literal-1     [{-}         ]}

    [ , {index-name-2 [{+} literal-4]} [ , {index-name-3 [{+} literal-6]} ] ] )
        {literal-3    [{-}         ]}       {literal-5    [{-}         ]}
```

5.4.8.4 Internal Formats of Subscripts, Index-names and Index-data-items

1. Subscripts are stored as either COMP or DISPLAY numeric integers with a size that may vary from 1 to 18 digits. They may 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 2 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 PIC 9(4). Their value is always positive.

   Index data items are implicitly SYNCHRONIZED; thus, when they are described within record descriptions they may 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 indices 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       [ { - }           ] }   ]
```

Restrictions on qualification, subscripting and indexing follow:

1. A data-name must not itself be subscripted nor indexed when that data-name is being used as an index, subscript or qualifier.

2. Indexing is not permitted where subscripting is not permitted.

3. An index name may 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, 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 may 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 that of '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 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. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control when the statement is executed in a called program.

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 may be implicitly or explicitly specified. Any attribute which has been explicitly specified is called an explicit attribute. If an attribute has not been specified explicitly, then the attribute takes on the default specification. Such an attribute 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 may be performed.

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 may 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 may 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 used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation 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 may 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.

c. The term variable indicates an identifier or literal.

Table 5-1
Combination of Symbols in Arithmetic Expression

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

5. Arithmetic expressions allow the user 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 may 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 format's 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:

```
{identifier-1            }   {IS [NOT] GREATER THAN}   {identifier-2            }
{literal-1               }   {IS [NOT] LESS THAN   }   {literal-2               }
{arithmetic-expression-1 }   {IS [NOT] EQUAL TO    }   {arithmetic-expression-2 }
                             {IS [NOT] >           }
                             {IS [NOT] <           }
                             {is [NOT] =           }
```

NOTE

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 may not 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:

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

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 4.13.7, Concepts of 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 one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters (See Section 3.4.2, The OBJECT-COMPUTER paragraph). If one of the operands is specified as numeric, it must be an integer data item 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, The MOVE Statement and Section 4.20, The 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, The MOVE Statement and Section 4.20, The 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 in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same.

There are two cases to consider: operands of equal size and operands of unequal size.

1. Operands of equal size.

   If the operands are 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 that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

2. Operands of unequal size.

   If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands of equal size.

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

Relation tests may 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.

4. Index data items may not 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:

$$\text{identifier IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \end{array} \right\}$$

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 4.23, The 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 an arithmetic expression 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, that is, 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 the 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 is 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 (Section 5.6.1) is negated through the use of the logical operator NOT. The negated simple condition affects the opposite truth value for a 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 { { AND } condition } ...
{ { OR }

where condition may 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 never be used when either AND or OR (but not both) is used exclusively in a combined condition, parentheses may be used to effect a final truth value when a mixture of AND, OR and NOT is used. (See Table 5-2, Combinations of Conditions, Logical Operators, and Parentheses, and Section 5.14 Condition Evaluation Rules.

Table 5-2 indicates the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is 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, may be immediately preceded by only:	Element when not last, may be immediately followed by only:
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 Relation Conditions

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 may 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:

```
relation-condition { { AND } [NOT] [relational-operator] object } ...
                   { { OR  }                                      }
```

Within a sequence of relation conditions both of the above forms of abbreviation may 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.

Abbreviated Combined Relation 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 may 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 (following the expansion of any abbreviated relation condition)
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign
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 The 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 The 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 The CORRESPONDING Phrase

For the purpose of this discussion, d_1 and d_2 must each be identifiers that refer to group items. A pair of data items, one from d_1 and one from d_2 correspond if the following conditions exist:

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

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 d_1 and d_2 must not contain level-number 66, 77, or 88 or the USAGE IS INDEX clause.

4. A data item that is subordinate to d_1 or d_2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, d_1 and d_2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses. (See Section 4.19, The OCCURS Clause.)

5.7.4 The 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 may 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 the 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 is equal to 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.

ACCEPT

5.8 THE ACCEPT STATEMENT

Function

The ACCEPT statement causes low volume data to be made available to the specified data item.

General Format

Format 1

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

Format 2

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

Syntax Rules

1. 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 the information requested to be transferred to the data item specified by the identifier, with no editing or conversion.

Format 2

3. 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.

4. DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes shall be from high order to low order (left to right), that is, year of century, month of year, and day of month. Therefore, July 4, 1976 would be 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 in length.

5. DAY is composed of the data elements year of century and day of year. The sequence of the data element codes shall be from high order to low order (left to right). That is, year of century, day of year. Therefore, July 4, 1976 would be 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 in length.

6. TIME is composed 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 in length. The minimum value of TIME is 00000000; the maximum value of TIME is 23595999.

ADD

5.9 THE ADD STATEMENT

Function

The ADD statement causes two or more numeric operands to be added together and the result to be stored.

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, The 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. See Section 5.7.1, The ROUNDED Phrase; Section 5.7.2, The SIZE ERROR Phrase; Section 5.7.3, The CORRESPONDING Phrase; Section 5.7.4, The Arithmetic Statements; Section 5.7.6, Overlapping Operands; and Section 5.7.5, Multiple Results in Arithmetic Statements.

2. 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 the word TO.

3. 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, ...,.

4. If Format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

5. The compiler ensures that enough places are carried so that no significant digits are lost during execution.

ALTER

5.10 THE 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 Rules

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

5.11 THE CALL STATEMENT

Function

The CALL statement causes control to be transferred from one object program to another within the run unit.

General Format

CALL literal-1[USING data-name-1[,data-name-2]...]

Syntax Rules

1. Literal-1 must be a nonnumeric literal, consisting of the characters 0-9 and A-Z, and must be six or less characters in length. Literal-1 is the entry point in the called subprogram. For COBOL subprograms, literal-1 is the first six characters of the called programs PROGRAM-ID.

2. The USING phrase is included in the CALL statement only if there is a non-empty USING phrase in the Procedure Division header of the called program. The number of operands in each USING phrase must be identical.

3. Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, or Linkage Section. Data-name-1, data-name-2, ..., may be qualified and/or subscripted.

General Rules

1. The program whose name is specified by the value of literal-1 is the called program; the program in which the CALL statement appears is the calling program.

2. The execution of a CALL statement causes control to pass to the called program.

3. A called program is in its initial state the first time it is called within a run unit.

   On all other 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.

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

5. The order of appearance of the data-names in the USING phrase of the CALL statement and the USING phrase in the PROCEDURE DIVISION header is critical. Corresponding data-names refer to a single set of data that is available to the called and calling program. The correspondence is positional, not by

name. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

6. The CALL statement may appear anywhere within the PROCEDURE DIVISION of a program. The implemention will provide all controls necessary to insure that the proper logic flow is maintained when segmentation of user code causes overlaying.

5.12 THE 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

```
CLOSE file-name-1 [ {REEL}  [WITH NO REWIND]  ]
                  [ {UNIT}  [FOR REMOVAL      ]  ]
                  [ WITH  {NO REWIND}           ]
                  [       {LOCK     }           ]

      [ , file-name-2 [ {REEL}  [WITH NO REWIND]  ]  ]...
      [               [ {UNIT}  [FOR REMOVAL      ]  ]
      [               [ WITH  {NO REWIND}           ]
      [               [       {LOCK     }           ]
```

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 analogous sequential media.

1. A CLOSE statement may 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 have 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, Relationship of Categories of Files and the Formats of the CLOSE Statement.

Table 5-3
Relationship of Categories of Files and the Formats of the CLOSE Statement

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 may 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

A technique is supplied to ensure that this file cannot be opened again during this execution of this 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 in the open mode when a STOP RUN statement is executed or when the object program execution is prematurely terminated, the file will be 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 media 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.

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.

CLOSE

5.13 THE CLOSE STATEMENT (INDEXED & 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 Rules

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

General Rules

1. A CLOSE statement may only be executed for a file in an open mode.

2. The results of executing each type of CLOSE for a relative or indexed file are summarized below.

   Close File

   a. Input Files and Input-Output Files (Sequential Access Mode):

      If the file is positioned at its end, the labels are processed according to the standard label convention. 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.

   b. Input Files and Input-Output Files (Random or Dynamic Access Mode); Output Files (Random, Dynamic, or Sequential Access Mode):

      Labels are processed according to the standard label convention. Closing operations specified by the Record Management Services are executed.

   File Lock

      A technique is supplied to ensure that this file cannot be opened again during this execution of the program.

3. If a file is in the open mode when a STOP RUN statement is executed or when the object program execution is prematurely terminated, the file will be 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.

COMPUTE

5.14 THE COMPUTE STATEMENT

Function

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

General Format

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

Syntax Rules

1. 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. See Section 5.7.1, The ROUNDED Phrase; Section 5.7.2, The SIZE ERROR Phrase; Section 5.7.4, The Arithmetic Statement; Section 5.7.6 Overlaying Operands; and 5.7.5, Multiple Results in Arithmetic Statements.

2. 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.)

3. If more than one identifier is specified for the result of the operation, that is preceding =, 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.

4. The COMPUTE statement allows the user 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.

5.15 THE DELETE STATEMENT (INDEXED & 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 which is 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 at the time of the execution of this statement.

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 may 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 on a record that is being simultaneously accessed by another task. 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.

DISPLAY

5.16 THE DISPLAY STATEMENT

Function

The DISPLAY statement causes low volume data to be transferred 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 may be any figurative constant except ALL.

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 and a carriage return character are 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 line-feed and carriage return characters are not appended to the sending item. If the device handler allows it, 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 console.

6. If the UPON phrase is not used, the data is written on the user's standard display device.

5.17 THE DIVIDE STATEMENT

Function

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal 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]
```

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. See Section 5.7.1, The ROUNDED Phrase; Section 5.7.2, The SIZE ERROR Phrase; Section 5.7.4, The Arithmetic Statements; and Section 5.7.6, Overlapping Operands; and Section 5.7.5, Multiple Results in Arithmetic Statements; for a description of these functions. See also, general rules 5 through 7 below for a discussion of the ROUNDED phrase and the SIZE ERROR phrase as they pertain to Formats 4 and 5.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

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

   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, the user will have to do his own analysis to recognize which situation has actually occurred.

EXIT

5.18 THE 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. An 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 THE GO TO STATEMENT

Function

The GO TO statement causes control to be transferred 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.

IF

5.20 THE IF STATEMENT

Function

The IF statement causes a condition (see Section 5.6, Conditional Expressions) 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 may be followed by a conditional statement.

2. The ELSE NEXT SENTENCE phrase may 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. (See Section 5.4.6, Imperative Sentences.) 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. (See Section 5.4.6, Imperative Sentences.) 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 may contain an IF statement. In this case the IF statement is said to be nested.

   IF statements within IF statements may 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.

INSPECT

5.21 THE INSPECT STATEMENT

Function

The INSPECT statement provides the ability to count (Format 1), replace (Format 2), or count and replace (Format 3) occurrences of single characters or groups of 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 may 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 may 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 will be treated as follows:

   a. If any of the identifiers (identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7) is described as alphanumeric, the INSPECT statement treats the contents of each identifier as a character-string.

   b. If any of the identifiers (identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7) is described as alphanumeric edited, numeric edited, or 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 (identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7) 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 5.16, The 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 may 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 (1) 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 (1) 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 (1) 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 five 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 B B 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 THE 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, The CORRESPONDING Phrase. The results are the same as if the user 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. 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 well 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 takes place as defined under Standard Alignment Rules, Section 4.13.10. 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 (see Section 4.23, the SIGN Clause), 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 takes place as defined under the Standard Alignment Rules, Section 4.13.10, 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 4.23, The 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 takes place as defined under the Standard Alignment Rules, Section 4.13.10. 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 move that is not an 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 will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area.

6. The following chart summarizes the legality of the various types of MOVE statements. The reference to general rules (for example /2a) indicates the rule (above) that prohibits the move or the behavior of a legal move.

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 THE MULTIPLY STATEMENT

Function

The MULTIPLY statement causes numeric data items to be multiplied 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-2    }

    [,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. See Section 5.7.1, The ROUNDED Phrase; Section 5.7.2, The SIZE ERROR Phrase; Section 5.7.4, The Arithmetic Statements; Section 5.7.6, Overlapping Operands; and Section 5.7.5, Multiple Results in Arithmetic Statements.

2. 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 applies for identifier-1 or literal-1 and identifier-3, etc.

3. 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 THE 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 with 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 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 refers to that file, either explicitly or implicitly.
4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 5-4, Permissible Statements, X at an intersection indicates that the specified statement, used in the sequential access mode, may be used with the sequential file organization and open mode given at the top of the column.

Table 5-4
Permissible Statements

Statement	Open Mode			
	Input	Output	Input-Output	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

5. A file may 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, the 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, the 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 this 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. If the file is not present, the first READ statement for this file causes the AT END condition to occur. (See section 5.27, The READ Statement.)

10. The NO REWIND phrase can be used only with sequential single reel/unit files.

11. The WITH NO REWIND phrase will be ignored if it does not apply to the storage media 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. The beginning file labels are processed only in the case of a single reel/unit file.

    b. The beginning reel/unit labels on the last existing reel/unit are processed as though the file was being opened with the INPUT phrase.

    c. The 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 initially 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. The labels are checked in accordance with the specified conventions for input-output label checking.

    b. The 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. The execution of an EXTEND statement will fail for any of the following reasons:

    NOTE

    The value in parentheses, following 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 issued to a file that is already opened for exclusive access by another task. (91)

    b. An OPEN statement issued to a device that has no available file space. (95)

    c. An OPEN statement to a file that shares buffer space with an already opened file. (96)

    d. An OPEN statement issued to a file that cannot be found on its associated I/O device. (97)

5.25 THE OPEN STATEMENT (INDEXED & RELATIVE)

Function

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

General Format

```
OPEN    { INPUT  file-name-1  [, file-name-2] ... }
        { OUTPUT file-name-3  [, file-name-4] ... }  ...
        { I-O    file-name-5  [, file-name-6] ... }
```

Syntax Rules

1. 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 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 references that file, either explicitly or implicitly.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 5-5, Permissible Statements, X at an intersection indicates that the specified statement used in the access mode given for that row may be used with indexed or relative file organizations and the open mode given at the top of the column.

Table 5-5
Permissible Statments

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 may 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 labels at the beginning of the file are processed as follows:

   a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with Record Management Services conventions for input label checking.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with Record Management Services conventions for output label writing. The behavior of the OPEN statement when label records are specified but not present is undefined.

8. 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.

9. 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 will result in an AT END condition.

10. The I-O phrase permits the opening of a file for both input and output operations. Because this phrase implies the existence of the file, it cannot be used if the file is being initially created.

11 When the I-O phrase is specified, the execution of the OPEN statement includes the following steps:

    a. The labels are checked in accordance with the standard conventions for input-output label checking.

    b. The new labels are written in accordance with the standard conventions for input-output label writing.

12. 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.

13. The execution of an OPEN statement will fail for any of the following reasons:

NOTE

The value in parentheses, following 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 issued to a file that is already opened for exclusive access by another task. (91)

    b. An OPEN statement issued to a device that has no available file space. (95)

    c. An OPEN statement to a file that shares buffer space with an already opened file. (96)

    d. An OPEN statement issued to a file that cannot be found on its associated I/O device. (97)

PERFORM

5.26 THE 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}
    {integer     } TIMES
```

Format 3

```
PERFORM procedure-name-1 [ {THROUGH} procedure-name-2 ] UNTIL condition-1
                         {THRU   }
```

Format 4

```
PERFORM procedure-name-1 [ {THROUGH} procedure-name-2 ]
                         {THRU   }
      VARYING {identifier-2} FROM {identifier-3}
              {index-name-1}      {index-name-2}
                                  {literal-1   }
                BY {identifier-4} UNTIL condition-1
                   {literal-2   }
    [ AFTER {identifier-5} FROM {identifier-6}
            {index-name-3}      {index-name-4}
                                {literal-3   }
                BY {identifier-7} UNTIL condition-2
                   {literal-4   }
      [ AFTER {identifier-8} FROM {identifier-9}
              {index-name-5}      {index-name-6}
                                  {literal-5   }
                BY {identifier-10} UNTIL condition-3 ] ]
                   {literal-6    }
```

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 may 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, and 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 may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement to where all of these paths must lead.

5. If control passes to these procedures via other than a PERFORM statement, control will pass 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 then control 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

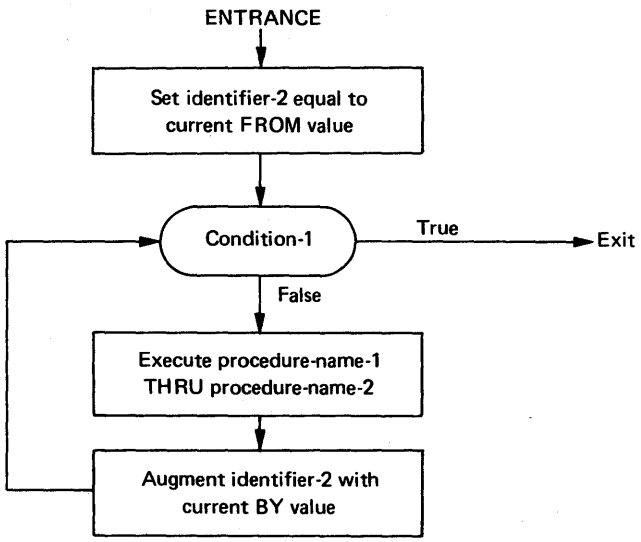
item referenced by identifier-1 is equal to 0 or is negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the next executable statement following the PERFORM statement.

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 phrase, identifier, when it appears in an associated VARYING of 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.

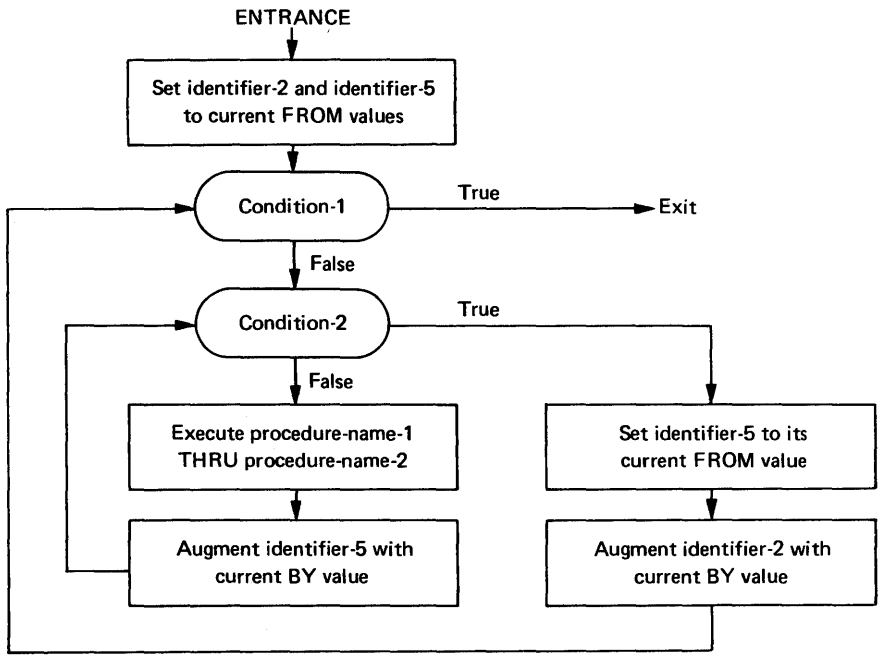


Flowchart for the VARYING Phrase of a PERFORM Statement Having One Condition

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.

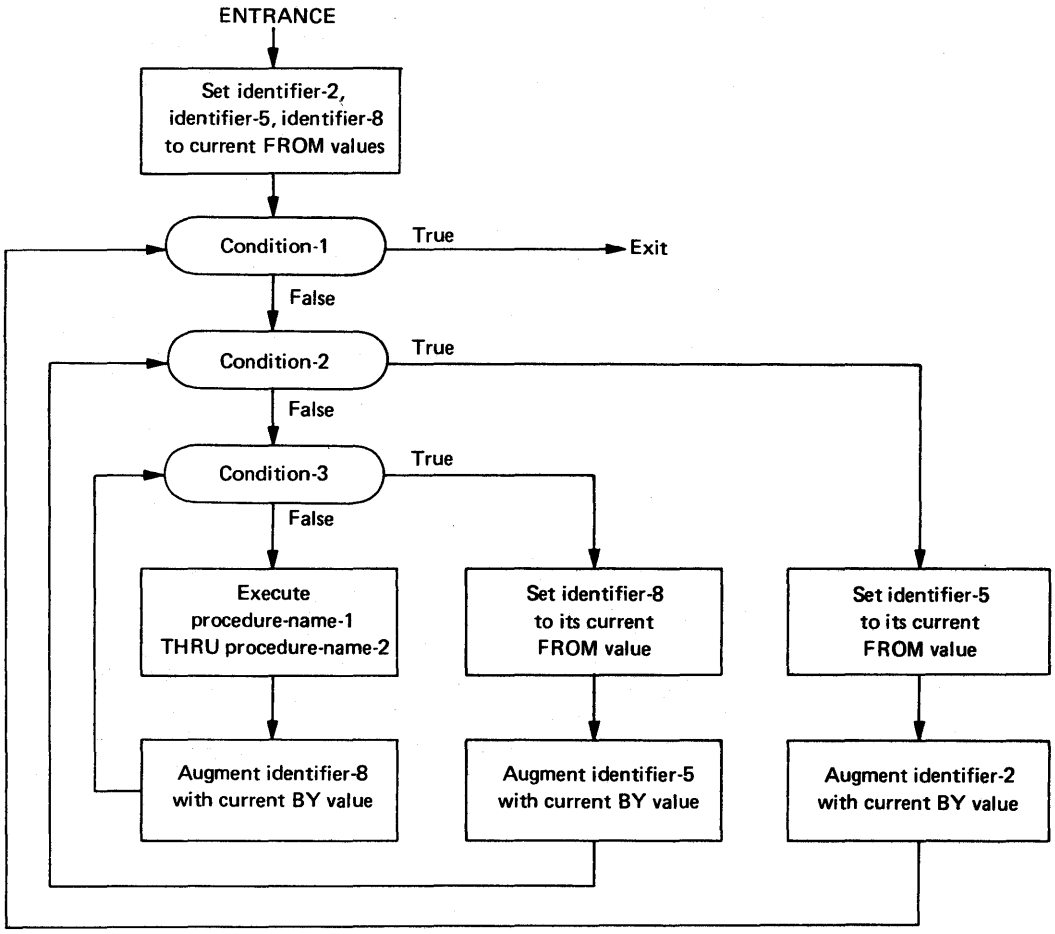


Flowchart for the VARYING Phrase of a PERFORM Statement Having Two Conditions

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.

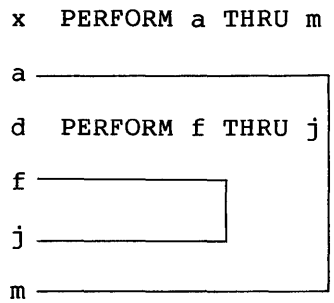
For three identifiers, the mechanism is the same as for two identifiers except that identifier-8 goes through a complete cycle each time that identifier-5 is augmented by identifier-7 or literal-4, which in turn goes through a complete cycle each time identifier-2 is varied.

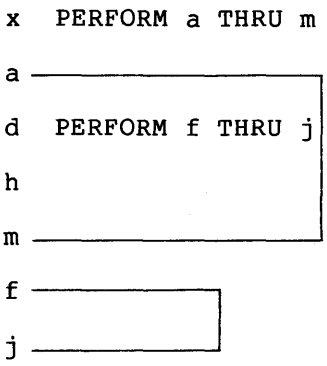


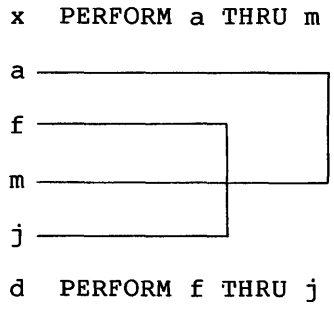
Flowchart for the VARYING Phrase of a PERFORM Statement Having Three Conditions.

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 may not 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 THE 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 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.

When the AT END condition occurs, execution of the input-output statement that caused the condition is unsuccessful.

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.

READ

5.28 THE 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 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.

When the AT END condition occurs, execution of the input-output statement that caused the condition is unsuccessful.

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 may arise; the READ is considered unsuccessful 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 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 will 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.

READ

5.29 THE 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 2

```
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 storage area.

2. Data-name must be the name of a data item specified as a record key associated with file-name.

3. Data-name may 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.

   When the AT END condition occurs, execution of the input-output statement that caused the condition is unsuccessful.

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 will 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.

REWRITE

5.30 THE 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 may 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.

REWRITE

5.31 THE 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 are 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 may 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 may 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.

REWRITE

5.32 THE 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 are 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 may 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 Rule 7b and c)

9. The contents of alternate record key data items of the record being rewritten may 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 Rule 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 THE 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           } }
            { data-name-1      { IS =        } { literal-1              } }
            { data-name-1      { IS =        } { arithmetic-expression-1} }
            { condition-name-1                                            }

   [ AND    { data-name-2      { IS EQUAL TO } { identifier-4           } } ]
            { data-name-2      { IS =        } { literal-2              } }
            { data-name-2      { IS =        } { arithmetic-expression-2} }
            { condition-name-2                                            }   ....

  { imperative-statement-2 }
  { NEXT SENTENCE          }
```

NOTE

The required relational character = 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., may 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 indices 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 type of 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 4.19, The 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 4.19, The 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 are 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 above. If one of the conditions is satisfied upon its 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 type of search operation 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. (For further information, see the PDP-11 COBOL User's Guide.) The length of the table is discussed in the OCCURS clause. (See Section 4.19, The 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.

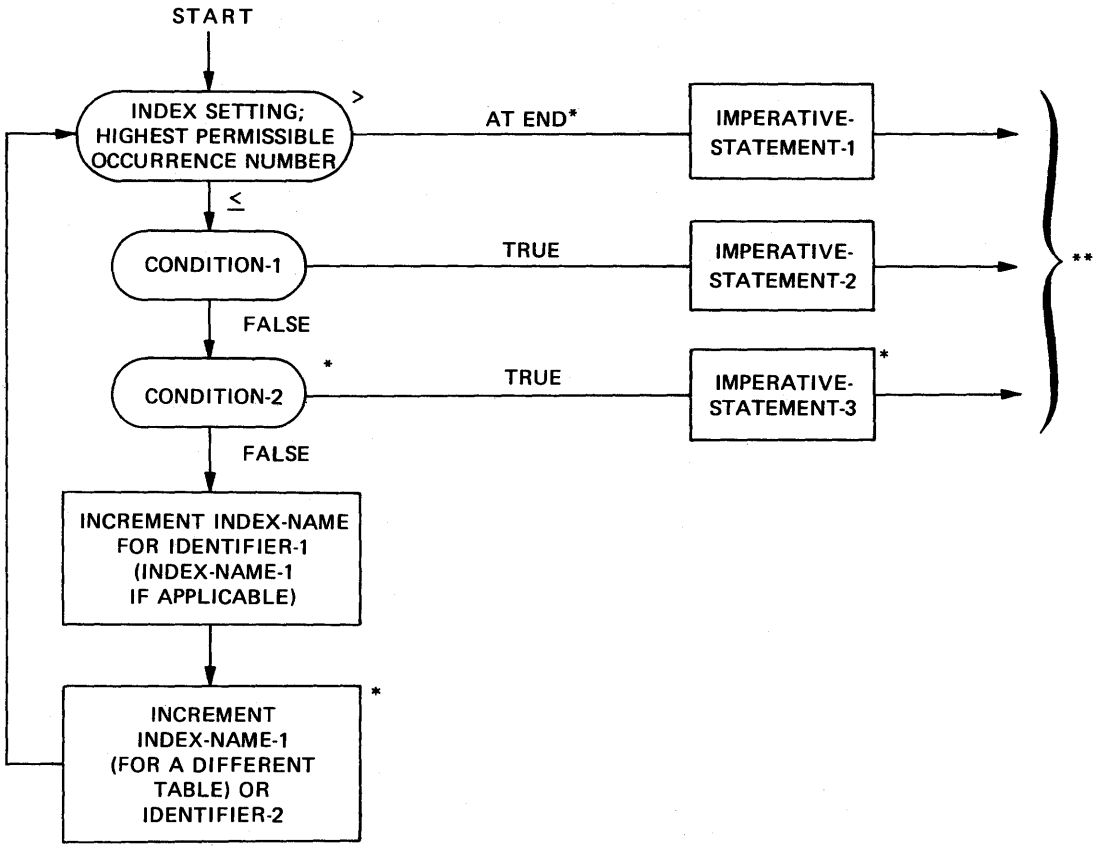
7. In Format 1, if the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY phrase of identifier-1, that index-name is used for this search. If this is not the case, or if the VARYING identifier-2 phrase is specified, the first (or only) index-name given in the INDEXED 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:



*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.

SET

5.34 THE SET STATEMENT

Function

The SET statement is used to establish 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 may 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 PDP-11 COBOL 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 action occurs:

   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. A new value of the index value portion of index-name-1 is always computed.

   b. If identifier-1 is an index data item, it may 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 may 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 chart represents the validity of various operand combinations in the SET statement. The references to the preceding general rules (/3b) indicates the applicable general rule.

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

START

5.35 THE START STATEMENT (RELATIVE)

Function

The START statement provides a means of logical positioning within a relative file for subsequent sequential retrieval of records.

General Format

```
START file-name [ KEY { IS EQUAL TO
                      { IS =
                      { IS GREATER THAN
                      { IS >
                      { IS NOT LESS THAN
                      { IS NOT <          } data-name ]
    [; INVALID KEY imperative-statement]
```

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. Data-name, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

4. Data-name may be qualified.

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 is 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 will be unsuccessful. 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.

START

5.36 THE 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

```
START file-name [ KEY { IS EQUAL TO
                      { IS =
                      { IS GREATER THAN
                      { IS >
                      { IS NOT LESS THAN
                      { IS NOT <          } data-name ]
[; INVALID KEY imperative-statement]
```

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 may reference a data item specified as a record key associated with file-name, or it may 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.

4. Data-name may be qualified.

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 statement 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 will be unsuccessful. 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.

STOP

5.37 THE 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 may be numeric or nonnumeric or may be 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 the object program execution is terminated.

2. If STOP literal is specified, the literal is displayed on the user's standard display device. The execution of the object program is temporarily suspended awaiting an ACCEPT from the console device. Typing a carriage return causes the program execution to continue. Continuation of the object program begins with the execution of the next executable statement in sequence.

STRING

5.38 THE 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 may 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 may not 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. The 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 the programmer, and he is responsible for setting its initial value. The initial value must not be less than one.

7. If the POINTER phrase is not specified, the following general rules 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 above are encountered, control passes to the next executable statement.

SUBTRACT

5.39 THE SUBTRACT STATEMENT

Function

The SUBTRACT statement is used to subtract one or the sum of two or more numeric data items from one or more items and set the value of one or more items 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, The 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. See Section 5.7.1, The ROUNDED phrase; Section 5.7.2, The SIZE ERROR Phrase; Section 5.7.3, The CORRESPONDING Phrase; Section 5.7.4, The Arithmetic Statements; Section 5.7.5, Multiple Results in Arithmetic Statements; and Section 5.7.6, Overlapping Operands.

2. 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 the word FROM.

3. 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.

4. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

5. The compiler insures enough places are carried so as not to lose significant digits during execution.

UNSTRING

5.40 THE 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 may be described as either alphabetic (except that the symbol B may not be used in the PICTURE character-string), alphanumeric, or numeric (except that the symbol P may not 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 may not be used in the PICTURE character-string).

5. No identifier may name a level 88 entry.

6. The DELIMITER IN phrase and the COUNT IN phrase may 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 full character set.

11. Each literal-1 or the 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, The 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, The 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. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

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.

USE

5.41 THE USE STATEMENT

Function

The USE statement specifies procedures for input-output error handling that are in addition to the standard procedures provided by the File Control System.

General Format

```
USE AFTER STANDARD {EXCEPTION} PROCEDURE ON { file-name-1 [, filename-2] ... }
                   {ERROR     }             { INPUT                        }
                                            { OUTPUT                       }
                                            { I-O                          }
                                            { EXTEND                       }
```

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 may 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 non-declarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may 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 THE 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

```
WRITE record-name [FROM identifier-1]
  [ {BEFORE} ADVANCING {identifier-2} [LINE ]  ]
  [ {AFTER }           {integer      } [LINES] ]
  [                    [PAGE]                  ]
  [ ; AT {END-OF-PAGE} imperative-statement    ]
         {EOP        }
```

Syntax Rules

1. Record-name and identifier-1 must not refer to data that is allocated to the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division; record-name may 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 may 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 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 file Record Management Services.

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 File Control System standard. 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 media. 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 THE 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 are allocated to the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division; record-name may 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 media 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 the Record Management Services.

8. When a file is opened in the output mode, records may 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 the file 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 may arise; the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the following actions take place.

    If the access mode is sequential, a boundary violation may 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.

    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.

    If the access mode is random or dynamic and the contents of the RELATIVE KEY data item does not lie in the range of key values associated with the file, a boundary violation may 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.

WRITE

5.44 THE 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. The 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 are 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

SEGMENTATION

The COBOL segmentation facility allows you to communicate object program overlay requirements to the compiler. COBOL segmentation deals only with the segmentation of procedures. Therefore, only the Procedure Division is considered in determining segmentation requirements.

6.1 ORGANIZATION

When segmentation is used, the Procedure Division for a source program must be written as a consecutive group of sections. Each section is composed of a series of closely related operations designed to collectively perform a particular function. Also, each section must be specified as belonging to the non-overlayable or overlayable portion of the program.

Using segmentation affects only the physical management of the object program during execution. It neither imposes any syntactic restrictions nor implies any semantic differences over the same program written without segmentation. The logical sequence of the program is the same as the physical sequence except for specific transfers of control. Flow of control from a non-overlayable segment to an overlayable segment, or from an overlayable segment to another overlayable segment is accomplished by the system.

6.1.1 Non-Overlayable vs. Overlayable Segments

A non-overlayable segment is that portion of the program that cannot be overlaid once it is loaded into memory. An overlayable segment, however, can overlay or be overlaid by any other overlayable segment.

6.2 USING THE SEGMENTATION FACILITY

The COBOL segmentation facility requires that you specify the SEGMENT-LIMIT clause (see Section 3.4.2) in the OBJECT-COMPUTER paragraph of the Environment Division, and that you assign segment numbers to each section of the Procedure Division.

6.2.1 The SEGMENT-LIMIT Clause

The SEGMENT-LIMIT clause has the following format:

SEGMENT-LIMIT IS segment-number

where segment-number is an integer ranging from 00 to 49.

The value specified by segment-number is used by the compiler to determine whether a segment is overlayable. That is, the value you specify in the SEGMENT-LIMIT clause is compared to the segment-number you assign to each section in the Procedure Division. Sections having segment-numbers that are less in value than the one specified by the SEGMENT-LIMIT clause are non-overlayable. Those having segment-number values that are greater than or equal to the segment limit are overlayable.

6.2.2 Segment Numbers

Sections within the Procedure Division are grouped into segments by means of a system of segment-numbers. A segment-number is included in the section header.

General Format

Section-name SECTION [segment-number].

Syntax Rules

1. The segment-number must be an integer ranging in value from 00 through 49.
2. If the section-number is omitted, 00 is assumed.

General Rules

1. All sections having the same segment-number constitute a program overlay. Sections with the same segment-numbers need not be physically contiguous in the source program.

2. Segments with segment-numbers less than the number specified in the SEGMENT-LIMIT clause belong to the non-overlayable portion of the program.

3. Segments with segment-numbers equal to or greater than the value specified in the SEGMENT-LIMIT clause belong to the overlayable portion of the program.

CHAPTER 7

THE LIBRARY MODULE

7.1 FUNCTION

The Library module provides a capability for specifying text that is to be copied from a library file.

COBOL library files contain source text that is available to the compiler for copying at compile time. The effect of the interpretation of the COPY statement is to insert the text of a file into the source program where it will be treated by the compiler as part of the source program.

7.2 THE COPY STATEMENT

Function

The COPY statement incorporates text into a COBOL source program.

General Format

```
COPY { text-name  }
     { literal-3  }

     [ REPLACING { , { literal-1 }   BY { literal-2 } } ... ]
                 {   { word-1    }     { word-2    } }
```

Syntax Rules

1. Each text-name must be unique within the file directories available to the compiler. (See Section 1.2.3.1, 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 "SY: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 in the following manner:

   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 into the source program unchanged.

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 PDP-11 COBOL extensions to the COBOL language. Words that are not reserved by the standard are indicated by an asterisk. A single asterisk indicates a word used in PDP-11 COBOL. A double asterisk indicates a word used in DECsystem-10 COBOL. All of the following words are reserved by PDP-11 COBOL and must not be used as user-created 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
COMPUTATIONAL
*COMPUTATIONAL-1
COMPUTE
*COMP-1
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
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
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
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

Row Value \ Column Value	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120
	000	010	020	030	040	050	060	070	100	110	120	130	140	150	160	170
0					space	(	0	8		H	P	X				
1		HT				)	1	9	A	I	Q	Y				
2					"	*	2		B	J	R	Z				
3						+	3	;	C	K	S					
4					$	,	4	<	D	L	T					
5						-	5	=	E	M	U					
6						.	6	>	F	N	V					
7						/	7		G	O	W					

Characters used to form words.

Row Value \ Column Value	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120
	000	010	020	030	040	050	060	070	100	110	120	130	140	150	160	170
0	NUL	BS	DLE	CAN	space	(	0	8	@	H	P	X	grave	h	p	x
1	SOH		DC1	EM	!	)	1	9	A	I	Q	Y	a	i	q	y
2	STX		DC2	SUB	"	*	2	:	B	J	R	Z	b	j	r	z
3	ETX		DC3	ESC	#	+	3	;	C	K	S	[	c	k	s	{
4	EOT		DC4	FS	$	,	4	<	D	L	T	\	d	l	t	|
5	ENQ		NAK	GS	%	-	5	=	E	M	U	]	e	m	u	}
6	ACK	SO	SYN	RS	&	.	6	>	F	N	V	(↑)	f	n	v	(ESC)
7	BEL	SI	ETB	US	apos	/	7	?	G	O	W	(←)	g	o	w	DEL

Characters that may appear within a non-numeric literal in the source text.

Row Value \ Column Value	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120
	000	010	020	030	040	050	060	070	100	110	120	130	140	150	160	170
0	NUL	BS	DLE	CAN	space	(	0	8	@	H	P	X	grave	h	p	x
1	SOH	HT	DC1	EM	!	)	1	9	A	I	Q	Y	a	i	q	y
2	STX	LF	DC2	SUB	"	*	2	:	B	J	R	Z	b	j	r	z
3	ETX	VT	DC3	ESC	#	+	3	;	C	K	S	[	c	k	s	{
4	EOT	FF	DC4	FS	$	,	4	<	D	L	T	\	d	l	t	|
5	ENQ	CR	NAK	GS	%	-	5	=	E	M	U	]	e	m	u	}
6	ACK	SO	SYN	RS	&	.	6	>	F	N	V	(↑)	f	n	v	(ESC)
7	BEL	SI	ETB	US	apos	/	7	?	G	O	W	(←)	g	o	w	DEL

Characters that may appear in an alphanumeric field in the object program.

Row Value \ Column Value	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120
	000	010	020	030	040	050	060	070	100	110	120	130	140	150	160	170
0																
1																
2		LF														
3		VT														
4		FF														
5		CR														
6																
7																

Characters used to delineate lines of the source text.

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 of a situation, by the operating system, that prevents further successful execution of that program.

ACCESS MODE
How records are to be operated upon in a file.

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

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 letters (A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z) and the space.

ALPHANUMERIC CHARACTER
Any character in the computer character set.

ALTERNATE RECORD KEY
A key, other than the Prime Record Key, whose contents identify a record within 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 belongs to the following set:

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

ASCENDING KEY

A key upon whose values data are 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 not involving the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

AT END CONDITION

A condition caused:

1. During 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. During execution of a RETURN statement when no next logical record exists for the associated sort or merge file.

3. During 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 may 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 which follows the page body.

CALLED PROGRAM
A program that is the object of a CALL statement combined at object time with the calling program to produce a run unit.

CALLING PROGRAM
A program which executes a CALL to another program.

CHARACTER
The basic indivisible unit of the language.

CHARACTER DATA ITEM
A data item consisting entirely of Standard Data Format characters.

CHARACTER POSITION
A character position is the amount of physical storage required to store a single Standard Data Format character whose usage is DISPLAY.

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

COBOL CHARACTER SET

The complete COBOL character set, except for the contents of nonnumeric literals, comment-entries, and comment lines, consists of the 59 characters listed below:

Character	Meaning
0, 1, . . , 9	digit
A, B, . . , Z	letter
	space
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than
<	less than
!	exclamation point
#	number sign
%	percent
&	ampersand
'	apostrophe
:	colon
?	question mark
@	at sign

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 the OR logical operator.

COMMENT-ENTRY
An entry in the Identification Division that may be any combination of characters from the computer character set.

COMMENT LINE
A source program line represented by an asterisk in the Indicator Area of the line, and any characters from the computer character set in Areas A and B of that line. The comment line serves only for program documentation. A special form of comment line represented by a stroke (/) in the Indicator Area of the line and any characters from the computer character set in Areas A and B of that line causes page ejection prior to printing the comment.

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

COMPILER DIRECTING STATEMENT
A statement beginning with a compiler directing verb that causes the compiler to take a specific action during compilation.

COMPLEX CONDITION
A condition where one or more logical operators act upon one or more conditions. (See NEGATED SIMPLE CONDITION; COMBINED CONDITION; NEGATED COMBINED CONDITION.)

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

CONCURRENT RUN UNIT
A run unit, other than this run unit, that has been initiated but not terminated during the time in which this 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 'condition' (condition-1, condition-2, . . .) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

CONDITION-NAME
Either a user-defined word that assigns a name to a subset of values that a conditional variable may 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 indices, 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 conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program depends on this 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 used to:

1. Associate a data-name, paragraph-name, condition-name or text-name with its qualifier.
2. Link two or more operands written in a series.
3. Form 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 then followed by a set of data clauses, as required.

DATA ITEM
A unit of data (excluding literals) defined by the COBOL program.

DATA ITEM VALIDATION
Verification of the proper contents of data items as they are accessed from, or stored in, the data base.

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 which 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 of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

DESCENDING KEY
A key upon whose values data are ordered, in accordance with the rules for comparing data items, starting with the highest value of key down to the lowest value of key.

DIGIT POSITION
A digit position is the amount of physical storage required to store a single digit. This amount may 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 (4) divisions in a PDP-11 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 PDP-11 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:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	stroke (virgule, slash)

ELEMENTARY ITEM
A data item 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 Control Division, 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.

File Description (FD)

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 indices, 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 may 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 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 integer 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 that format.

INTEGRITY
The ability to ensure that record relationships and data item contents are not adversely affected by concurrent processes.

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 a 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 within an indexed file.

KEY WORD
A reserved word needed when the format where 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-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 within the page body.

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

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

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 may 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 may include any character in the computer character set, 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 may 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 in which the computer environment within which the object program is executed is described.

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 a file is in between the time an OPEN statement is issued and the time a CLOSE statement is executed.

OPERAND
Although the general definition of operand is a component which is operated upon, in this publication 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 form a portion of a COBOL procedural statement or of a COBOL clause.

PHYSICAL RECORD
(See BLOCK.)

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

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

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

PROGRAM-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
a. A data-name which is used in a reference with another data-name at a lower level in the same hierarchy.

b. A section-name which is used in a reference with a paragraph-name specified in that section.

c. 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
IS UNEQUAL TO	Not equal to
EQUALS	Equal to
EXCEEDS	Greater than

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

RESOURCE
A facility or service controlled by the operating system that can be used by an executing program.

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 Division.

In the Environment and Data Division, 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 which names a section in the Procedure Division.

SECURITY

The ability to prohibit access to a data base record by unauthorized means.

SEGMENT-NUMBER

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

SEQUENTIAL PROCESSING
(See SYNCHRONOUS PROCESSING.)

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 in which the computer environment within which the source program is compiled is described.

SOURCE PROGRAM

Although it is recognized that a source program may be represented by other forms and symbols, in this report it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences 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, full stop)
"	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 which identifies library text.

TEXT-WORD
Any character-string or separator, except space, in a COBOL library.

TOP MARGIN
An empty area which 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
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.

UPDATE USAGE MODE
The state of a realm during which its record may be both accessed and modified.

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 which 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 character-string of not more than 30 characters that forms a user-defined word, a system-name, or a 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 of Working-Storage records or both.

INDEX

PDP-11 COBOL
Language Reference
Manual
AA-1749D-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

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
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.