HONEYWELL

MULTICS COBOL REFERENCE MANUAL

SOFTWARE

MULTICS COBOL REFERENCE MANUAL

SUBJECT

Description of Multics COBOL Language Rules, Syntax, Formats, and Usage

SPECIAL INSTRUCTIONS

This manual supersedes AS44, Revision 1, dated December 1976, and its addenda, AS44-01A, dated October 1977, AS44-01B, dated January 1979, AS44-01C, dated May 1979, AS44-01D, dated September 1979, AS44-01E, dated December 1979, and AS44-01F, dated July 1981. Refer to the Preface for "Significant Changes."

The manual has been extensively revised and reorganized. Throughout the manual, change bars in the margins indicate technical additions and asterisks denote deletions.

SOFTWARE SUPPORTED

Multics Software Release 10.2

ORDER NUMBER

AS44-02

December 1983

Honeywell

PREFACE

This manual is published as a reference guide for programmers and systems analysts whose programs written in the COBOL language are to be compiled and executed in the Multics environment. It is designed to provide a formal definition of the COBOL language as specified in American National Standard COBOL X3.23-1974 and implemented for the Honeywell Multics computer systems. It contains formats, syntax rules, and general rules for the construction of a working COBOL source program. The manual is organized on the basis of a functional processing concept in a manner similar to the standard specifications from which it is derived.

> Areas in this manual which are enclosed in boxes indicate Multics extensions to the American National Standard COBOL. Use of these extensions in COBOL source programs intended to be compiled interchangeably by one or more COBOL compilers may lead to different compilation and/or execution results.

Section 1 describes the syntax rules used in this manual. Section 2 presents a general treatment of all COBOL language concepts. Section 3 provides an overall outline of the COBOL divisions and their uses. Section 4 explains the reference format used in COBOL programming.

Section 5 describes the Control Division, a Honeywell extension that allows the user to specify default values for the current compilation of the source program. Section 6 describes the Identification, Environment, and Data Divisions in the nucleus; the Procedure Division in the nucleus is described in Section 7.

The standard defines a Nucleus and eleven functional processing modules: Table Handling, Sequential I-O, Relative I-O, Indexed I-O, Sort-Merge, Report Writer, Segmentation, Library, Debug, Inter-Program Communication, and Communication. Each module contains either one or two levels. In all cases the lower level is a proper subset of the higher level. The Nucleus contains language elements that are necessary for internal processing.

Section 8 describes the Table Handling module which contains the language elements necessary for: (1) the definition of tables, (2) the identification, manipulation and use of indices, and (3) reference to items within tables (two levels are provided).

File No.: 1L23 AS44-02

Section 9 describes the Sequential, Relative, and Indexed I-O modules. The Sequential I-O module contains the language elements necessary for the definition and access of sequentially organized files (two levels are provided). The Relative I-O module provides the capability of defining and accessing mass storage files in which records are identified by relative record numbers (two levels are provided). The Indexed I-O module provides the capability of defining mass storage files in which records are identified by the value of a key and accessed through an index (two levels are provided).

Section 10 describes the Sort-Merge module that allows for the inclusion of one or more sorts in a COBOL program (two levels are provided).

Section 11 describes the Segmentation module which provides for the overlaying at object time of Procedure Division sections (two levels are provided).

Section 12 describes the Library module which provides for the inclusion into a program of predefined COBOL text (two levels are provided).

Section 13 describes the Debug module which provides a means by which the user can specify a debugging algorithm--the conditions under which data and procedure items are monitored during execution of the program (two levels are provided).

Section 14 describes the Inter-Program Communication module which provides a facility by which a program can communicate with one or more other programs (two levels are provided).

Section 15 describes the Communication module which provides the ability to access, process, and create messages or portions thereof, and to communicate through a Message Control System with local communication devices (two levels are provided).

Section 16 describes the Report Writer module which provides for the semi-automatic production of printed reports (two levels are provided).

Appendix A contains a list of reserved words that may not be employed as user-defined words.

Appendix B is a glossary of terms used in this manual. Many terms, such as data-name and identifier, have similar meanings but very specific differences in the technical sense, of which the user must be aware. The exact meaning of all such terms, which is the intended meaning each time the word is used in this manual, is contained in the glossary.

Appendix C contains ASCII and EBCDIC collating sequence and bit configuration charts.

Appendix D describes FIPS leveling.

Other manuals that contain related supporting information are:

Multics COBOL Users' Guide, Order Number AS43

Multics Programmer's Reference Manual, Order Number AG91

Multics Commands and Active Functions, Order Number AG92

Significant Changes In AS44-02

1. Material which constitutes an extension to the COBOL Standard is enclosed in boxes.

2. The data alignment rules have been rewritten.

3. Delimited scope statements (a feature defined in the new COBOL standard) were added to Multics COBOL.

   Delimited scope statements are derived from existing statements (such as ADD) which may be followed by conditional phrases (such as ON SIZE ERROR) and which become conditional/imperative statements accordingly as the conditional phrase is present/absent. These statements become delimited scope statements if they are terminated by explicit scope delimiters (in the case of the ADD) statement this consists of the reserved word END-ADD). For example:

```
ADD A TO B
ON SIZE ERROR DISPLAY "FOOEY"
NOT ON SIZE ERROR DISPLAY "GOODY" END-ADD
```

   A delimited scope statement is an imperative statement. It may contain or be contained in other delimited scope statements.

   The table which follows summarizes the delimited scope statements, which were added to Multics COBOL:

Statement	Conditional Phrase	Scope Delimiters
ADD	ON SIZE ERROR	END-ADD
SUBTRACT	ON SIZE ERROR	END-SUBTRACT
MULTIPLY	ON SIZE ERROR	END-MULTIPLY
DIVIDE	ON SIZE ERROR	END-DIVIDE
COMPUTE	ON SIZE ERROR	END-COMPUTE
READ	AT END	END-READ
READ	INVALID KEY	END-READ
WRITE	INVALID KEY	END-WRITE
WRITE	AT END-OF-PAGE	END-WRITE
REWRITE	INVALID KEY	END-REWRITE
DELETE	INVALID KEY	END-DELETE
START	INVALID KEY	END-START
RECEIVE	NO DATA	END-RECEIVE
RETURN	AT END	END-RETURN
STRING	ON OVERFLOW	END-STRING
UNSTRING	ON OVERFLOW	END-UNSTRING
PERFORM		END-PERFORM
IF		END-IF
SEARCH		END-SEARCH

ACKNOWLEDGMENT

This acknowledgment has been reproduced from the CODASYL COBOL Journal of Development 1981, as requested in that publication, prepared and published by the CODASYL Programming Language Committee.

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

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

No warranty, expressed or implied, is made by any contributor or by the CODASYL 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

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. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1

NOTATION USED IN FORMATS AND RULES

This section contains descriptions of the rules of syntax applied in the language specifications of this manual.

DEFINITION OF A GENERAL FORMAT

A general format is the specific arrangement of the elements of a clause or a statement. A clause or a statement consists of elements as defined below. In this manual, a format is shown adjacent to information that defines the clause or statement. If more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the general formats. (If used, optional clauses must appear in the sequence shown.) In certain cases, stated explicitly in the rules associated with a given format, clauses can appear in sequences other than those shown. Applications, requirements, or restrictions are shown as rules.

SYNTAX RULES

A syntax rule is one that defines or clarifies the order in which words or elements are arranged to form larger elements such as phrases, clauses, or statements. Syntax rules also impose restrictions on individual words or elements.

Syntax rules are used to define or clarify the exact manner in which the statement must be written; i.e., the order of the elements of the statement and the restrictions on what each element may represent.

GENERAL RULES

A general rule is one that defines or clarifies the meaning or relationship of meanings of an element or set of elements. It is used to define or clarify the semantics of the statement and the effect that it has on either compilation or execution.

ELEMENTS

Elements which make up a clause or a statement consist of uppercase words, lowercase words, level numbers, brackets, braces, connectives, and special characters.

Words

Underlined uppercase words are called key words and are required when the functions of which they are a part are used. Uppercase words that are not underlined are optional and may be included in the source program at the discretion of the user. Uppercase words, whether underlined or not, must be spelled correctly.

In a general format, lowercase words are generic terms used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be supplied by the user. If such generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term in a subsequent explanation or discussion.

Level-Numbers

Specific level-numbers appearing in data description entry formats are required when such entries are used in a COBOL program. In this manual, the convention 01, 02, ..., 09 indicates level-numbers 1 through 9.

Brackets, Braces, and Choice Indicators

When brackets, [], enclose a portion of a general format, the user may include or omit the enclosed portion. When braces, {}, enclose a portion of a general format, the user must select one of the enclosed options. When choice indicators, {¦¦}, enclose a portion of a general format, the user must select one or more of the unique options, but any option may be specified only once.

Options are indicated in a general format by vertically stacking alternative possibilities. When braces, brackets, or choice indicators enclose a portion of a format, but only one possibility is shown, its function is to delimit that portion of a format to which the following ellipsis applies.

Ellipsis

In the general format, the ellipsis (...) represents the position at which the user elects repetition of a format. The portion of the format that may be repeated is determined as follows:

1. Given an ellipsis (...) in a clause or statement format, scanning right to left, determine the right bracket (]), right brace (}), or right choice indicator (¦) immediately to the left of the ellipsis. Continue scanning from right to left and determine the logically matching left bracket, brace, or choice indicator. The ellipsis applies to the material contained within the delimiters.

2. If an ellipsis is immediately preceded by a comma (,...) or a semicolon (;...) then instances of the material may be optionally separated by separator commas or separator semicolons.

Format Punctuation Characters

The punctuation characters comma and semicolon are shown in some formats. Where shown in the formats, they are optional and may be included or omitted in the coding. In the source program, these two punctuation characters are interchangeable and either one may be used when one of them is shown in the formats. Neither character may immediately precede the first clause of an entry or paragraph.

A comma or semicolon can be used to separate statements in the Procedure Division.

Paragraphs within the Identification and Procedure Divisions and entries within the Environment and Data Divisions must be terminated by the separator period.

Special Characters

When the characters + - < > and = appear in formats, they are required when such formats are used, even though they are not underlined.

SECTION 2

LANGUAGE CONCEPTS

CHARACTER SET

The basic and indivisible unit of the language is the character. The character set used to form COBOL character-strings and separators consists of 85 characters, including both uppercase and lowercase letters, digits, and special characters. In the case of nonnumeric literals, comment-entries, and comment lines, the character set is expanded to include the computer's entire character set. The characters allowable in each type of character-string and as separators are defined in this section and in Appendix B.

Individual characters of the language are joined to form character-strings and separators. A separator may be concatenated with another separator or with a character-string. A character-string may be concatenated only with a separator. The concatenation of character-strings and separators forms the text of a source program.

Uppercase and Lowercase Letters

In Multics COBOL, the user can employ both lowercase and uppercase letters to form character-strings wherever the rules for formation of character-strings allow the usage of uppercase letters.

Two characters form a corresponding uppercase and lowercase pair if they differ only in case (for example, A and a). The following rules define the conditions under which two characters of a corresponding uppercase and lowercase pair are treated as two distinct characters or as the same character.

Within PICTURE character-strings, in user words, and in reserved words, the two characters of a corresponding uppercase and lowercase pair are treated as specifying the same character. (For example, PIC X and pic x both describe a one-character alphanumeric item. Further, MOVE, move, and Move are all considered correct usages of the reserved word MOVE.)

The two characters of a corresponding uppercase and lowercase pair are treated as two distinct characters in the following cases:

- Within nonnumeric literals and comment-entries. For example, these two statements are different:

  IF "A" EQUALS TEMP ...

  IF "a" EQUALS TEMP ...

- Within internal file names (in the SELECT clause)
- Within program-id (in the Identification Division)
- Within text-name (in the COPY statement)

Separators

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

1. The punctuation character space is a separator. Wherever a space is used as a separator in a program, more than one space may be used.

2. The punctuation characters comma, semicolon, and period, when immediately followed by a space, are separators. These separators may appear in a COBOL source program wherever explicitly permitted by the general formats, by format punctuation rules, by statement and sentence structure definitions, or by reference format rules. (Refer to Section 4.)

The separator comma and the separator semicolon may appear anywhere that a separator space may appear.

3. The punctuation characters right parenthesis and left parenthesis are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, indexes, arithmetic expressions, or conditions.

4. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or a left parenthesis; a closing quotation mark must be immediately followed by one of the separators: space, comma, semicolon, period, or right parenthesis.

   Quotation marks may appear only in balanced pairs delimiting nonnumeric literals, except when the literal is continued. (Refer to "Continuation of Lines" in Section 4.)

5. Pseudo-text delimiters are separators which consist of two contiguous equal sign (==) characters on the same line. An opening pseudo-text delimiter must be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space,

6. The separator space may immediately precede all separators except:

   a. As specified by the reference format rules.

   b. The closing quotation mark, in which case a preceding space is considered as part of the nonnumeric literal and not as a separator.

   c. As the opening pseudo-text delimiter, where the preceding space is required.

7. The separator space may immediately follow any separator except the opening quotation mark, in which case a following space is considered as part of the nonnumeric literal and not as a separator.

Any punctuation character that appears as part of the specification of a PICTURE character-string or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

Rules established for the formation of separators do not apply to the characters that make up the contents of nonnumeric literals, comment-entries, or comment lines.

Character-Strings

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

COBOL WORDS

A COBOL word is a character-string of not more than 30 characters that forms a user-defined word, a system-name, or a reserved word. Within a given source program, these classes form disjoint sets; a COBOL word may belong to only one class.

User-Defined Words

A user-defined word is a COBOL word that the user must specify to satisfy the format of a clause or statement. Each character of a user-defined word is selected from the following set of characters:

- A, B, C, ..., Z
- a, b, c, ..., z
- 0, 1, 2, ..., 9
- - (hyphen)

NOTE: The hyphen may not appear as the first or last character.

There are 16 types of user-defined words:

alphabet-name	mnemonic-name
cd-name	paragraph-name
condition-name	program-name
data-name	record-name
file-name	report-name
index-name	section-name
level-number	segment-number
library-name	test-name

Within a given source program, 14 of these 16 types of user-defined words are grouped into the following 12 disjoint sets:

alphabet-names
cd-names
condition-names, data-names, and record-names
file-names
index-names
library-names
mnemonic-names
paragraph-names
program-names
report-names
section-names
text-names

All user-defined words, except segment-numbers and level-numbers, can belong to only one disjoint set. Further, all user-defined words within a given disjoint set must be unique, either because no other user-defined word in the same source program has identical spelling or punctuation, or because uniqueness can be ensured by qualification. (Refer to "Uniqueness of Reference" in this section.)

With the exception of paragraph-name, section-name, level-number, and segment-number, all user-defined words must contain at least one alphabetic character. Segment-numbers and level-numbers need not be unique; a given specification of a segment-number or level-number may be identical to any other segment-number or level-number and may even be identical to a paragraph-name or section-name.

CONDITION-NAME

A condition-name is a name that 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 within the Environment Division where a condition-name must be assigned to the ON STATUS or OFF STATUS, or both, of SWITCH-n.

A condition-name is used only as an abbreviation for the relation condition; this relation condition stipulates that the associated conditional variable is equal to one of the set of values to which that condition-name is assigned.

MNEMONIC-NAME

A mnemonic-name assigns a user-defined word to a hardware or operating system feature. These associations are established in the SPECIAL-NAMES paragraph of the Environment Division.

PARAGRAPH-NAME

A paragraph-name is a word that names a paragraph in the Procedure Division. Paragraph-names are equivalent only if they are composed of the same sequence of the same number of digits and/or characters.

SECTION-NAME

A section-name is a word that names a section in the Procedure Division. Section-names are equivalent only if they are composed of the same sequence of the same number of digits and/or characters.

OTHER USER-DEFINED WORDS

Refer to Appendix B for definitions of other types of user-defined words.

System-Names

A system-name is a COBOL word that is used to communicate with the operating environment. System-names are used to refer to a specific feature of the hardware/software environment, such as SYSOUT, SYSIN, etc.

Reserved Words

A reserved word is a COBOL word that is one of a specified list of words that may be used in COBOL source programs but must not appear in the programs as user-defined words. Reserved words can be used only as indicated in the general formats. (Refer to Appendix A for a listing of the reserved words.)

The seven types of reserved words are:

- Key words
- Optional words
- Connectives
- Special registers
- Figurative constants
- Special-character words
- System-names

KEY WORDS

A key word is a word whose presence is required when the format in which the word appears is used in a source program. Within each format, such words are uppercase and underlined. The three types of key words are:

- Verbs, such as ADD and READ
- Required words, which appear in statement and entry formats
- Words that have a specific functional meaning, such as NEGATIVE and SECTION

OPTIONAL WORDS

Within each format, uppercase words that are not underlined are called optional words and may be specified at the discretion of the user. The presence or absence of an optional word does not alter the semantics of the COBOL program in which it appears.

CONNECTIVES

The three types of connectives are:

- Qualifier connectives that are used to associate a data-name, condition-name, text-name, or paragraph-name with its qualifier: OF, IN
- Series connectives that link two or more consecutive operands: , (separator comma) or ; (separator semicolon)
- Logical connectives used in the formation of conditions: AND, OR

SPECIAL REGISTERS

Certain reserved words are used to name and reference special registers. Special registers are designated compiler-generated memory areas whose primary use is to store information produced in conjunction with specific COBOL features. The special registers are LINAGE-COUNTER, LINE-COUNTER, PAGE-COUNTER, and DEBUG-ITEM.

FIGURATIVE CONSTANTS

Figurative constants are reserved words used to name and reference specific compiler-generated constant values. (Refer to "Figurative Constant Values" later in this section.)

SPECIAL-CHARACTER WORDS

Arithmetic operators, e.g., + and -, and relational characters, e.g., >, <, and =, are reserved words. (Refer to Appendix B for definitions.)

SYSTEM-NAMES

System-names are used to refer to a specific feature of the hardware/software environment, such as SYSOUT, SWITCH-n, CATALOG-NAME, etc.

LITERALS

A literal is a character-string whose value is implied by an ordered set of characters of which the literal is composed or by specification of a reserved word that references a figurative constant. Literals are either nonnumeric or numeric.

Nonnumeric Literals

A nonnumeric literal is a character-string delimited on both ends by quotation marks and consisting of any ASCII characters. A nonnumeric literal may be up to 256 characters in length. To represent a quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used. The value of a nonnumeric literal in the object program is the string of characters itself, except that:

- Delimiting quotation marks are excluded.
- Each embedded pair of contiguous quotation marks represents a single quotation mark character.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals are categorized as alphanumeric. (Refer to "PICTURE Clause" in Section 6.)

Nonprinting Characters

Any character within a nonnumeric literal can be represented by a string of digit pairs enclosed within quotation marks. In the following example of such a string, the leftmost digit in a pair represents the left five bits of the character and the rightmost digit represents the right four bits of the character:

"AX4"3BHC"YZ"

Thus, AX4 and YZ represent normal ASCII characters, and 3BHC represents the following bit pattern (see Table 2-1 for digit values):

"00011 1011 10001 1100"

Note that spaces (<SP>) are included in the above example only to indicate blank characters and to improve its readability.

Binary values for these digit pairs are listed in Table 2-1. Digits on the left side of the table can be used as either the leftmost or rightmost digit of a pair. However, digits on the right side of the table can be used only as the leftmost digit.

Table 2-1. Values of Digit Pairs in Nonnumeric Literals

Left or Right Digit	Binary Value	Left Digit Only	Binary Value
0	0000	G	10000
1	0001	H	10001
2	0010	I	10010
3	0011	J	10011
4	0100	K	10100
5	0101	L	10101
6	0110	M	10110
7	0111	N	10111
8	1000	O	11000
9	1001	P	11001
A	1010	Q	11010
B	1011	R	11011
C	1100	S	11100
D	1101	T	11101
E	1110	U	11110
F	1111	V	11111

NOTE: The specification of nonprinting characters may change in subsequent releases of Multics COBOL. The Multics COBOL user should be prepared to modify programs that contain nonprinting characters.

Numeric Literals

A numeric literal is a character-string whose characters are selected from the digits 0 through 9, the plus sign, the minus sign, and the decimal point. A numeric literal may be from one to 18 digits in length. Rules for forming numeric literals are:

1. A literal must contain at least one digit.

2. A literal must contain no more than one sign character. If used, a sign character must appear as the leftmost character of the literal. If unsigned, the literal is positive.

3. A literal must contain no more than one decimal point. The decimal point is treated as an assumed decimal point and may appear in any location within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

   If a literal conforms to the rules for the formation of numeric literals but is enclosed in quotation marks, it is a nonnumeric 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. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user. (Refer to "PICTURE Clause" in Section 6 for additional information.)

Figurative Constant Values

Figurative constant values are generated by the compiler and are referenced by using the reserved words given below. These words must not be delimited by quotation marks when they are used as figurative constants. The singular and plural forms of figurative constants are equivalent and may be used interchangeably. Figurative constant values and reserved words used to reference them are:

ZERO/ZEROS/ZEROES	Represents the value '0', or one or more of the character '0', depending on the context in which it appears.
SPACE/SPACES	Represents one or more of the character space (octal 040).
HIGH-VALUE/HIGH-VALUES	Represents one or more of the character that has the highest ordinal position in the program collating sequence.

If a PROGRAM COLLATING SEQUENCE clause is not present, the character associated with HIGH-VALUE is the character represented by octal 177. If the program contains a PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph, the actual ASCII character associated with HIGH-VALUE depends upon the collating sequence specified by that clause. If the alphabet-name specified is EBCDIC, that character is represented by octal 377. If the alphabet-name is user-defined, that value corresponds to the position of the highest explicitly or implicitly specified character in the alphabet specified.

LOW-VALUE/LOW-VALUES — Represents one or more of the character that has the lowest ordinal position in the program collating sequence.

If a PROGRAM COLLATING SEQUENCE clause is not present, the character associated with LOW-VALUE is the character represented by octal 000. If the program contains a PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph, the octal ASCII character associated with LOW-VALUE depends upon the collating sequence specified by that clause. If the alphabet-name specified is EBCDIC, that character is represented by octal 000. If the alphabet-name is user-defined, that value corresponds to the position of the lowest explicitly or implicitly specified character in the alphabet specified.

QUOTE/QUOTES — Represents one or more of the quotation mark character ("). The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to delimit a nonnumeric literal. For example, the phrase QUOTE ABD QUOTE is incorrect as a method of stating the nonnumeric literal "ABD".

ALL literal — Represents one or more characters of the string of characters that make up the literal. The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is included in the source program only for readability.

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, such as 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 on the right until the size of the

resultant string is equal to the size in characters of the associated data item. This is accomplished 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, such as when the figurative constant appears in a DISPLAY, STRING, STOP, or UNSTRING 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 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.

PICTURE CHARACTER-STRINGS

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols.

Any punctuation character that appears as part of the specification of a PICTURE character-string is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string. (Refer to "PICTURE Clause" in Section 6 for a discussion of the PICTURE character-string and for the rules governing its use.)

COMMENT-ENTRIES

A comment-entry is an entry in the Identification Division that may be any combination of characters. A comment-entry has no effect on the operation of a COBOL program.

COMPUTER INDEPENDENT DATA DESCRIPTION

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 to an equipment-oriented format. This standard data format is oriented to general data processing applications and uses the decimal system to represent numbers and the remaining characters in the COBOL character set to describe nonnumeric data items.

Logical Records and Files

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

PHYSICAL ASPECTS OF A FILE

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

- The grouping of logical records within the physical limitations of the file medium
- The means by which the file can be identified

CONCEPTUAL CHARACTERISTICS OF A FILE

The conceptual characteristics of a file are the explicit definitions of each logical entity within the file itself. In a COBOL program, input and output statements refer to one logical record. A 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 are convenient to a particular input or output device for the storage of data.

Several source language methods are available for describing the relationship of logical records and physical records. When a permissible relationship has been established, control of the accessibility of logical records as related to the physical unit must be provided by the interaction of the object program on the hardware and/or software system. In this manual, references to records mean to logical records, unless physical record is specifically stated.

The concept of a logical record 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.

RECORD CONCEPTS

The record description consists of a set of data description entries that describe 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.

Concept of Levels

A record may need to be subdivided for convenient data reference. The smallest subdivision is called an elementary item. A record can consist of a sequence of elementary items, or the record itself may be an elementary item.

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

A system of level-numbers indicates 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 (though not necessarily successive) level-numbers not greater in value than 49. (Special level-numbers 66, 77, and 88 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 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 have no true level:

- Entries that specify elementary items or groups introduced by a RENAMES clause. Entries that describe items using RENAMES clauses for the purpose of regrouping data items have been assigned the special level-number 66.
- Entries that specify noncontiguous working-storage, constant, and linkage data items. Entries that specify noncontiguous 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. Entries that specify condition-names, to be associated with particular values of a conditional variable, have been assigned the special level-number 88.

Classes of Data

The five categories of data items are grouped into three classes: alphabetic, numeric, and alphanumeric. (Refer to "PICTURE Clause.") 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 also to one of the categories. The class of a group item at object program execution is treated as alphanumeric, regardless of the class of elementary items subordinate to that group item. Table 2-2 depicts the relationship of the class and categories of data items.

Table 2-2. Data Item Classes and Categories

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

Selection of Character Representation and Radix

A data item can be represented internally in either character or binary form, depending upon the usage of the item. (Refer to the "USAGE Clause" in Section 6 and the "USAGE IS INDEX Clause" in Section 8.)

The size of an elementary data item or a group item is the number of characters in standard data format of the item. Synchronization and usage may cause a difference between this size and the actual number of characters required for the internal representation of data.

Algebraic Signs

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

The SIGN clause allows the location of the operational sign to be explicitly stated in a numeric DISPLAY item. The clause is optional; if it is not used, operational signs will be represented as defined by the USAGE clause of the Data Division.

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

Standard Alignment Rules

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 with zero-fill or truncation to the left, as required.

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 alignment rules are modified. (Refer to "JUSTIFIED Clause" in Section 6.)

Item Alignment for Increased Object Code Efficiency

Some computer memories are organized with natural addressing boundaries in the computer memory (e.g., word boundaries, half-word boundaries, byte boundaries, digit boundaries). The way in which data is stored is determined by the object program and need not respect these natural boundaries.

Certain uses of data (e.g., in arithmetic operations or in subscripting) may be facilitated if the data is stored so as to be aligned on these natural boundaries. Additional machine instructions may be required for accessing and storing data if it does not appear between those natural boundaries.

Data items which are aligned on their natural boundaries are said to be synchronized. A synchronized data item is assumed to be introduced and carried in that form; conversion to synchronized form occurs only during the execution of a procedure (other than READ or WRITE) which stores data in the item.

Synchronization can be accomplished in two ways:

1. By use of the SYNCHRONIZED clause.

2. By recognizing the appropriate natural boundaries and organizing the data suitably without the use of the SYNCHRONIZED clause. This technique gives rise to data allocation rules which are specific to the implementor.

The data allocation rules used by Multics COBOL are the same as those used by GCOS COBOL 74 and GCOS COBOL8X except where noted. An algorithm is defined which may be applied to a level 01 or level 77 data item and which determines the OFFSET and SIZE of every data sub-item contained within the data item. Alignment is forced by inserting unused space (called FILLER SPACE) before or after a data item.

Multics COBOL Data Allocation Rules

Major data items (i.e., level 01 or level 77 data items) are allocated either on word or double-word boundaries.

The following four cases arise: elementary data items, group data items, repeated group data items, and data items containing a REDEFINES clause.

CASE I

An elementary data item (EDI) is allocated according to the following rules:

1. The REQUIRED BOUNDARY for the EDI is determined using the table:

Data Type	Boundary
DISPLAY	1 byte
COMP	1 byte (packed decimal)
COMP-5	1 byte (packed decimal)
COMP-6	4 word (long binary)
COMP-7	2 half-word (short binary)
COMP-8	0 digit (packed decimal)
INDEX	8 double-word

   The SYNCHRONIZED clause applies only to COMP-7 and COMP-8 data. For COMP-7 data, a full-word is allocated and the data occupies only the rightmost 18 bits. This is similar to the PL/I data type fixed bin aligned. For COMP-8 data, the data is allocated starting on a byte boundary and occupies an integral number of bytes.

2. If the CURRENT OFFSET does not specify a boundary of the required type then FILLER SPACE (called type 2 FILLER SPACE) is allocated up to the next boundary of the required type.

3. The PICTURE, SIGN, and USAGE clauses are used to determine the size of the EDI and the required space is allocated.

A repeated elementary data item is allocated according to the rules:

1. A single instance of the EDI is allocated as described in the preceding section.

2. The size of the EDI equals the size of a single instance of the EDI times the number of occurrences specified.

CASE II

A group data item is allocated according to the rules:

1. The group parameters are determined by:

   a. The REQUIRED BOUNDARY for the GDI is "byte" or the REQUIRED BOUNDARY of the first EDI contained within the GDI, whichever is larger.

   b. The MAXIMUM REQUIRED BOUNDARY for the GDI is "byte" or the largest REQUIRED BOUNDARY for the EDIs contained within the GDI, whichever is larger.

2. If the CURRENT OFFSET is not a multiple of the REQUIRED BOUNDARY, type 2 FILLER SPACE is added.

3. The members of the GDI are allocated in the order in which they appear in the source program.

4. The OFFSET of the GDI equals the OFFSET of the first EDI contained within the GDI.

   Note: In Multics COBOL the OFFSET of the GDI is the CURRENT OFFSET. The SIZE of the GDI includes any type 2 FILLER SPACE which precedes the first EDI contained within the GDI.

5. The SIZE of the GDI equals the sum of the sizes of its members plus any FILLER SPACE inserted between its members. If the size of the GDI is not an integral number of bytes, then FILLER SPACE (called type 3 FILLER SPACE) is added.

CASE III

A repeated group data item is allocated according to the rules:

1. The OFFSET and SIZE of an instance of the GDI are determined as described in case II.

2. If the size of the instance of the GDI is not a multiple of its MAXIMUM REQUIRED BOUNDARY, FILLER SPACE (called type 1 FILLER SPACE) is added.

3. The OFFSET of the repeated GDI equals the OFFSET of the first instance of the repeated GDI.

4. The SIZE of the repeated GDI is equal to the size of an instance of the GDI times the number of occurrences specified.

Note: In Multics COBOL the SIZE of the repeated GDI does not include the type 1 FILLER SPACE appended to the last repetition of the GDI.

CASE IV

A data item which contains a REDEFINES clause is allocated according to the rules:

1. The SAVED OFFSET is set equal to the CURRENT OFFSET.

2. The CURRENT OFFSET is set equal to the OFFSET of the OBJECT data item (i.e., the data item which is being redefined).

3. The allocation of the data item takes place as previously defined.

4. If the CURRENT OFFSET exceeds the SAVED OFFSET, the SAVED OFFSET is set equal to the CURRENT OFFSET.

5. If the next data item does not specify another redefinition of the OBJECT data item, the CURRENT OFFSET is set equal to the SAVED OFFSET.

An example using the allocation algorithm is:

```
01 a.
02  b     pic x.
02  c     occurs 2 times.
03   d    pic x.
03   e    usage comp-6
02  f     occurs 2 times.
03   g    usage comp-8 pic s99.
02  h     occurs 3 times comp-8 pic s99.
02  i     comp-8 pic s99.
02  j     pic x.
02  k     occurs 2 times.
03   l    comp-6.
02  m     pic x.
02  n     pic x(8).
02  o     redefines n comp-6.
02  p.
03   q    comp-8 pic s99.
03   r    comp-8 pic s9.
02  s     comp pic s9.
```

The following table gives the offsets and sizes of the identifiers. An offset is given as a six-digit octal word offset (starting at 000002), possibly followed by a decimal bit offset (0,1,...), enclosed in parentheses.

Identifier	Offset	Usage/Class	Size
a	000002	GROUP alphanum	X(53)
b	000002	DSPLY alphanum	X(1)
c	000002(9)	GROUP alphanum	X(7)
d	000002(9)	DSPLY alphanum	X(1)
e	000003	COMP6 numeric	S9(11) bin(35)
f	000006	GROUP alphanum	X(2)
g	000006	COMP8 numeric	S9(2)
h	000007	COMP8 numeric	S9(2)
i	000010(5)	COMP8 numeric	S9(2)
j	000010(18)	DSPLY alphanum	X(1)
k	000010(27)	GROUP alphanum	X(5)
l	000011	COMP6 numeric	S9(11) bin(35)
m	000014	DSPLY alphanum	X(1)
n	000014(9)	DSPLY alphanum	X(8)
o	000015	COMP6 numeric	S9(11) bin(35)
p	000016(9)	GROUP alphanum	X(3)
q	000016(9)	COMP8 numeric	S9(2)
r	000016(23)	COMP8 numeric	S9(1)
s	000017	COMP5 numeric	S9(1)

Uniqueness of Reference

QUALIFICATION

Every user-specified name that defines an element in a COBOL source program must be unique. The name can be considered unique because no other name has identical spelling and hyphenation or because references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy in which the name exists. 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. Two identical data-names must not appear as entries subordinate to a group item unless they can be made unique through qualification. In the Procedure Division, two identical paragraph-names must not appear in the same section, if they are explicitly referenced.

In the hierarchy of qualification, names associated with a level indicator are the most significant, then those names associated with level-number 01, then names associated with level-number 02...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 a procedure-name.

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

The general formats for qualification are:

Format 1:

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

Format 2:

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

Format 3:

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

Format 4:

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

Format 5:

```
{PAGE-COUNTER} {OF} report-name
{LINE-COUNTER} {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 twice in a qualified reference.
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 Environment, Data, and Procedure Divisions (except in a REDEFINES clause, where qualification is unnecessary and must not be used).
4. A paragraph-name must not be duplicated within a section, if it is explicitly referenced. 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 being used as a qualifier cannot be subscripted.
6. A name can be qualified even though it does not need qualification; if more than one combination of qualifiers ensures uniqueness, 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.

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. (Refer to "OCCURS" Clause" and to "Table Handling" in Section 8.)

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. In the Report Section, neither a sum counter nor the special registers LINE-COUNTER and PAGE-COUNTER can be used as a subscript.

The subscript can be signed. 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, etc. 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 identifies the table element is delimited by the balanced pair of parentheses following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization.

One, two, or three subcript levels are allowed. The format is:

```
{ data-name      } ({ data-name } ...)
{ condition-name }  { literal   }
```

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 operators + 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 (when 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.

When a statement that refers to an indexed table element is executed, the value contained in the index referenced by 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 resulting from relative indexing.

One, two, or three subscript levels are allowed. The general format for indexing is:

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

IDENTIFIER

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

The general formats for identifiers are:

Format 1:

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

[( {data-name-3} ... )]
   {literal-1   }
```

Format 2:

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

[( {index-name-1 [{+} literal-2]} ... )]
   {literal-1    [{-}          ]}
```

The restrictions on qualification, subscripting, and indexing follow:

- A data-name must not be subscripted or indexed when it is being used as an index, subscript, or qualifier.
- Indexing is not permitted where subscripting is not permitted.
- An index 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. Such data items are called index data items.
- Literal-1 in the above format must be a positive numeric integer. Literal-2 must be an unsigned numeric integer.

CONDITION-NAME

Each condition-name must be unique; it may be made unique through qualification and/or indexing or through 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 the same as those for "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.

EXPLICIT AND IMPLICIT SPECIFICATIONS

Four types of explicit and implicit specifications occur in COBOL source programs:

- Explicit and implicit Procedure Division references
- Explicit and implicit transfers of control
- Explicit and implicit attributes
- Explicit and implicit scope terminators

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. An implicit reference also occurs during the execution of a PERFORM statement when the index or data item referenced by the index-name or identifier specified in the VARYING, AFTER, or UNTIL phrase is initialized, modified, or evaluated by the control mechanism associated with that PERFORM statement. Such an implicit reference occurs only if the data item contributes to the execution of the statement.

Transfers of Control

The mechanism that controls program flow transfers control from statement to statement in the sequence in which they were written in the source program unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control can be passed. The transfer of control from statement to statement occurs without specifying an explicit Procedure Division statement and, therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit methods 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, which override the statement-to-statement transfers of control:

- If a paragraph is being executed under control of another COBOL statement (e.g., PERFORM, USE, SORT, or MERGE) and the paragraph is the last paragraph in the range of the controlling statement, an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. In addition, if a paragraph is being executed under control of a PERFORM statement that causes iterative execution and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.
- When execution of any COBOL statement results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Another implicit transfer of control occurs after the declarative section is executed, as described above.
- When a SORT or a MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

An explicit transfer of control alters the implicit control transfer mechanism and can be caused only by the execution of a procedure branching or conditional statement. (Refer to "Categories of Statements" in Section 3.) Execution of the procedure branching ALTER statement 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 EXIT PROGRAM statement causes an explicit transfer of control when the statement is executed in a called program.

In this manual, the term "next executable statement" refers to the next COBOL statement to which control is transferred in accordance with the above rules and the rules associated with each language element in the Procedure Division.

No executable statement follows the last statement either in a declarative section or in a program when the paragraph in which it appears is not being executed under the control of some other COBOL statement.

Explicit and Implicit Attributes

Attributes may be specified either explicitly or implicitly. If an attribute has not been specified explicitly, it assumes the default specification and is considered as an implicit attribute. For example, the usage of a data item need not be specified. In this case, the usage of the data item is DISPLAY.

Explicit and Implicit Scope Terminators

Scope terminators serve to delimit the scope of certain statements. Scope terminators are of two types: explicit and implicit.

The explicit scope terminators are:

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

The implicit scope terminators are:

1. At the end of any sentence, the separator period which terminates the scope of all previous statements not yet terminated.

2. Within any statement containing another statement, the next phrase of the containing statement following the contained statement, terminates the scope of any unterminated contained statement (e.g., ELSE, THEN, etc.).

EXTERNAL SWITCH

An external switch is a software device which is used to indicate that one of two alternate states exists. These alternate states are referred to as the ON status and the OFF status of the associated external switch.

The status of an external switch may be interrogated by testing condition-names associated with that switch. The association of a condition-name with an external switch and the association of a user-specified mnemonic-name with the external switch is established in the SPECIAL NAMES paragraph of the Environment Division.

The user may establish a value for an external switch every time a program is executed. (See the run_cobol command described in the Multics COBOL User's Guide, (Order No. AS43)).

The status of certain external switches may be altered by the SET statement (see the SET statement, Section 7).

SECTION 3

OVERALL OUTLINE OF THE COBOL DIVISIONS

GENERAL DESCRIPTION OF COBOL

COBOL is a programming language used throughout the world for programming business data processing applications. The COBOL language was developed by a group of computer users and manufacturers, and first documentation was distributed in April 1960. Since then it has undergone many extensions and changes resulting from manufacturer experience with COBOL implementation and user experience with COBOL programming for computers of many sizes and configurations. The improvements are embodied in this version of the language termed COBOL-74.

From a computer users standpoint, COBOL offers several advantages:

- It provides a quick means of program implementation.
- It reduces costs of converting programs from one computer system to another.
- It reduces time and effort required for training programmers.
- It guarantees a measure of standard documentation.

COBOL allows the user to instruct computers in a language similar to English. Following conventions of a standard reference format, the user writes COBOL statements in sentence and paragraph form to describe data to be processed and to specify required procedures. The complete body of statements is called a source program.

The source program is always in a Multics segment created by a text editor, from punched cards, or from magnetic tape. The segment is submitted as input to the computer under control of a special program, called a compiler. The compiler produces an object program in a Multics segment that contains the actual sequence of machine instructions required to accomplish the functions specified in the source program. In addition, the compiler produces edited listings that include a printout of the source program.

Another important function of the compiler is to analyze the source program for correct COBOL syntax and to print error comments for any syntax errors that are detected. The computer's operation under control of the compiler is called compilation.

COBOL DIVISIONS

A Multics COBOL program comprises five divisions, each of which must be constructed according to specific rules. (Refer to Section 4.) A COBOL division is a set of zero, one, or more sections or paragraphs; together they constitute the division body. Paragraphs are likewise formed and combined in accordance with a specific set of rules.

The following divisions are used to form a COBOL program and must appear in the program in the following order:

- Control Division (optional)
- Identification Division
- Environment Division
- Data Division
- Procedure Division

Control Division

The optional Control Division contains the Default Section. When it is included in the source program, the Control Division must be the first division in the program, preceding the Identification Division. The Default Section allows compiler default conditions to be specified if other than standard defaults are required.

General Format:

```
[ CONTROL DIVISION.

[ DEFAULT SECTION.    [ default clauses ]  ... . ] ]
```

Identification Division

The Identification Division must be included in every COBOL source program. This division identifies the source program, the object program, and the resultant output listing. The user may also include the date on which the program is written, the date on which source program compilation is accomplished, and other information as desired.

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, the PROGRAM-ID paragraph. The other paragraphs may be included at the discretion of the user.

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 ]  ... ]
```

Environment Division

The Environment Division must be included in every COBOL source program. This division provides a standard method of expressing those aspects of a data processing problem that depend upon the physical characteristics of a specific computer. In this division, the compiling computer and the executing computer are specified. In addition, information relating to input/output control, special hardware or operating system characteristics, and control techniques can also be presented.

The Environment Division is divided into the Configuration Section and the Input-Output Section.

> The Configuration and Input-Output Sections are optional in Multics COBOL.

The Configuration Section provides program documentation for the hardware characteristics of the computer used for compilation and of the computer used to execute the object program. Provisions are included in this section for relating specific hardware and operating system features to user-specified mnemonic names.

The Configuration Section is subdivided into three paragraphs:

- SOURCE-COMPUTER paragraph, which identifies the computer on which the source program is to be compiled
- OBJECT-COMPUTER paragraph, which identifies the computer on which the object program produced by the compiler is to be executed

- SPECIAL-NAMES paragraph, which associates hardware names and operating system features with mnemonic-names used in the source program and relates alphabet-names to character sets and/or collating sequences

The Input-Output Section is concerned with information needed to control the transmission and handling of data between external devices and the object program; it is subdivided into the following two paragraphs:

- FILE-CONTROL paragraph, which names all files used in the program and associates them with internal file names, which are in turn associated with I/O switches by the operating system
- I-O-CONTROL paragraph, which defines special control techniques to be used in the object program

General Format:

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

Data Division

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:

- That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas
- That which is developed internally and placed into intermediate or working-storage, or placed into a specific format for reporting output

- Constants that are defined by the user

The Data Division is required in every COBOL program and is subdivided into the File Section, the Working-Storage Section, the Constant Section, the Linkage Section, the Communication Section, and the Report 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.

> The Constant Section describes data items whose values are assigned in the source program and do not change during the execution of the object program.

The Linkage Section appears 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 that of the Working-Storage Section. (Refer to Section 14 for Interprogram Communication information.)

The Communication Section describes the data item in the source program that serves as the interface between the COBOL Message Control System (CMCS) and the program. (Refer to Section 15, "Communication Facility.")

The Report Section describes the content and format of Reports that are to be generated.

General Format:

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

```
[ CONSTANT SECTION.
    { 77-level-description-entry
    { record-description-entry } ... ]
```

```
[ LINKAGE SECTION.

      { 77-level-description-entry   } ... ]
      { record-description-entry     }

[ COMMUNICATION SECTION.

      [ communication-description-entry [ record-description-entry ] ... ] ... ]

[ REPORT SECTION.

      [ report-description-entry ] ... ]
```

Procedure Division

The Procedure Division contains the procedures required to solve a given problem and must be included in every COBOL source program. This division may contain declarative and nondeclarative procedures.

PROCEDURE DIVISION HEADER

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

```
PROCEDURE DIVISION  [ USING { data-name } ... ] .
```

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 ] ... } ...
```

DECLARATIVES

Declarative sections must be grouped at the beginning of the Procedure Division, preceded by the key word DECLARATIVES, and followed by the key words END DECLARATIVES. (Refer to the USE statement applications in Section 9.)

PROCEDURES

A procedure is composed of a paragraph, or group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. 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 in which it occurs. It consists of a paragraph-name (which may be qualified) 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 procedures appear.

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

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

STATEMENTS AND SENTENCES

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.

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.

There are four types of statements: imperative statements, conditional statements. compiler-directing statements, and delimited-scope statements.

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

Conditional Statements and Sentences

A conditional sentence 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:

1. An IF, SEARCH, or RETURN statement.
2. A READ statement that specifies the AT END or INVALID KEY phrase.
3. A WRITE statement that specifies the INVALID KEY or END-OF-PAGE phrase.
4. A START, REWRITE, or DELETE statement that specifies the INVALID KEY phrase.
5. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.
6. A RECEIVE statement that specifies the NO DATA phrase.
7. A STRING or UNSTRING statement that specifies the ON OVERFLOW phrase.
8. A CALL statement that specifies the ON OVERFLOW phrase.

A conditional sentence is a conditional statement, optionally preceded by an imperative statement, terminated by a separator period.

Compiler-Directing Statements and Sentences

A compiler-directing statement consists of a compiler-directing verb and its operands. Compiler-directing verbs are COPY, REPLACE, and USE. A compiler-directing statement causes the compiler to take a specific action during compilation.

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

Imperative Statements and Sentences

An imperative statement may assume two forms:

1. An imperative statement may begin with an imperative verb which specifies an unconditional action to be taken by the object program.

2. An imperative statement may consist of a conditional statement delimited by its explicit-scope termination (delimited-scope statement).

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

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

Notes:
1. Without the optional SIZE ERROR phrase
2. Without the optional INVALID KEY phrase
3. Without the optional ON OVERFLOW phrase
4. Without the optional AT END or INVALID KEY phrase
5. Without the optional NO DATA phrase
6. Without the optional INVALID KEY or END-OF-PAGE phrase

When the term "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 or any phrase associated with a statement containing the "imperative statement."

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

> Delimited Scope Statement
>
> A delimited scope statement is any statement which includes its explicit scope terminator (see "Explicit and Implicit Scope Terminators").

Scope of Statements

Scope terminators delimit the scope of certain Procedure Division statements. Statements which include their explicit scope terminators are termed delimited scope statements. The scope of statements which are contained within statements (nested) may also be implicitly terminated.

When statements are nested within other statements which allow optional conditional phrases, any optional conditional phrase encountered is considered to be the next phrase of the nearest preceding unterminated statement with which that phrase is permitted to be associated, but with which no such phrase has already been associated.

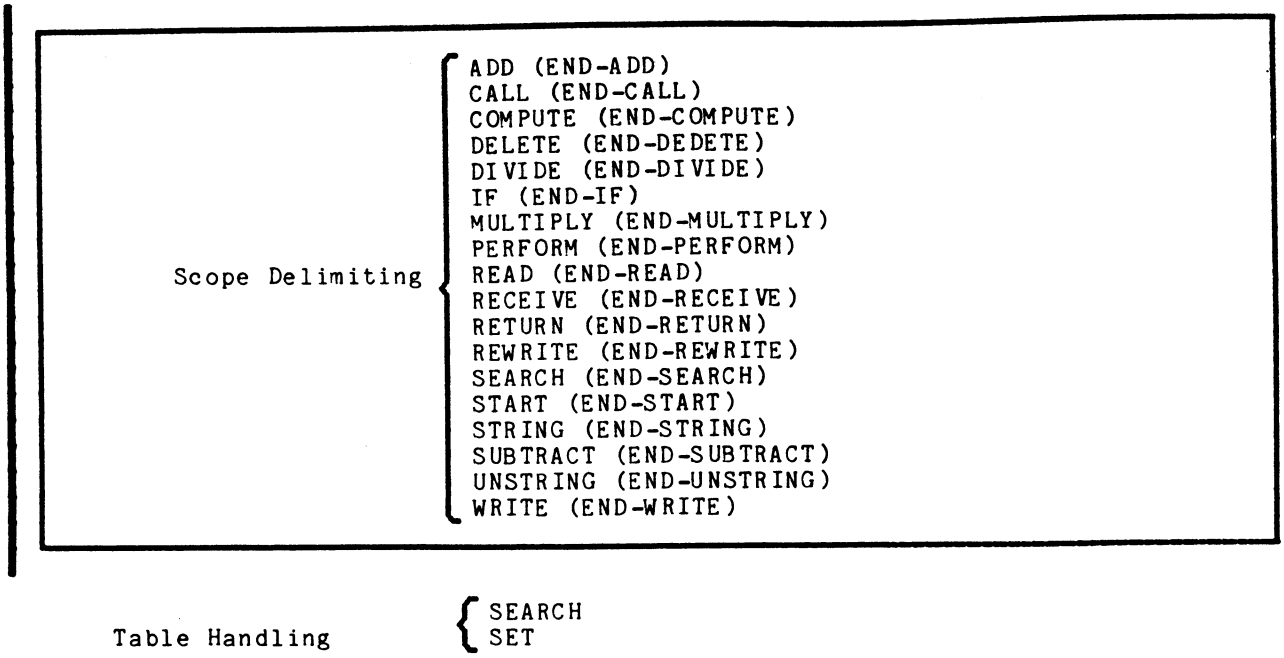
Categories of Statements

Verbs available in COBOL are listed below within their functional categories:

Category	Verbs
Arithmetic	ADD COMPUTE DIVIDE INSPECT (TALLYING) MULTIPLY SUBTRACT
Compiler-Directing	COPY REPLACE USE
Conditional	ADD (SIZE ERROR) CALL (OVERFLOW) COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DIVIDE (SIZE ERROR) IF MULTIPLY (SIZE ERROR) READ (AT END or INVALID KEY) RETURN RECEIVE (NO DATA) REWRITE (INVALID KEY) SEARCH (AT END) START (INVALID KEY) STRING (OVERFLOW) SUBTRACT (SIZE ERROR) UNSTRING (OVERFLOW) WRITE (INVALID KEY or END-OF-PAGE)

Data Movement	ACCEPT (DATE, DAY, or TIME) ACCEPT (MESSAGE COUNT) INSPECT (REPLACING) MOVE STRING UNSTRING
Ending	STOP
Input/Output	ACCEPT (identifier) CLOSE DELETE DISABLE DISPLAY ENABLE OPEN PURGE READ RECEIVE REWRITE SEND START STOP (literal) WRITE
Interprogram Communication	CALL CANCEL

Category	Verbs
No operation	EXIT
Ordering	MERGE RELEASE RETURN SORT
Procedure Branching	ALTER CALL EXIT PROGRAM GO TO PERFORM
Report Writer	GENERATE INITIATE TERMINATE SUPPRESS



Specific Statement Formats

The specific statement formats, together with a detailed discussion of the restrictions and limitations associated with each, appear in alphabetic sequence in the appropriate sections of this manual. (Refer to the index.)

SECTION 4

REFERENCE FORMAT

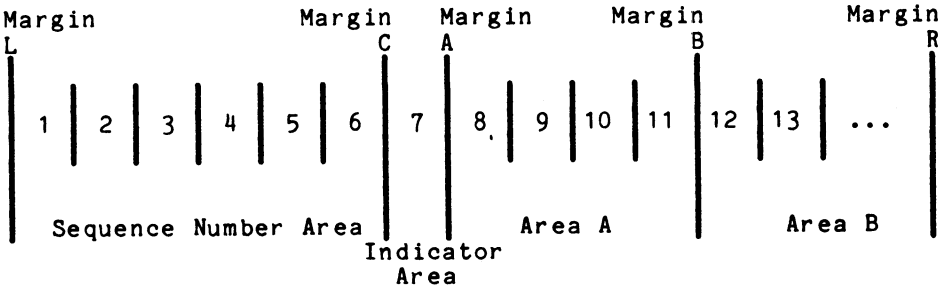
The reference format provides a standard method for describing COBOL source programs; it is defined in terms of character positions in a line. The COBOL compiler accepts source programs written in reference format and produces an output listing of the source program in reference format. Refer to the Multics COBOL Users' Guide for an alternate reference format for terminals.

The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

The divisions of a source program must be ordered as follows; the Control Division (optional), the Identification Division, then the Environment Division, then the Data Division, and then the Procedure Division. Each division must be written according to the rules specified for the reference format.

REFERENCE FORMAT REPRESENTATION

The reference format for a line (character position, not print position) is represented as follows:



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

Margin C is between character positions 6 and 7.

Margin A is between character positions 7 and 8.

Margin B is between character positions 11 and 12.

Margin R is immediately to the right of character position 256.

The sequence number area occupies six character positions (1-6) and is between margin L and margin C.

The indicator area is character position 7 of a line.

Area A occupies character positions 8, 9, 10, and 11 and is between margin A and margin B.

Area B begins immediately to the right of margin B and terminates immediately to the left of margin R.

Sequence Numbers

The sequence number can be used to label a source program line. The contents of the sequence number area are user-defined and can consist of any character in the host computer's character set.

> Sequence numbers are for documentation purposes only.

Continuation of Lines

Whenever a sentence, entry, phrase, or clause requires more than one line, it can be continued by starting subsequent lines in Area B. These subsequent lines are called continuation lines. The line being continued is called the continued line. Any word, literal, or PICTURE character-string can be broken in such a way that part of it appears on a continuation line.

A hyphen in the indicator area of a line indicates that the first nonblank character in area B of the current line is the successor of the last nonblank character of the preceding line, excluding intervening comment lines, with no intervening space. However, if the continued line contains a nonnumeric literal without a closing quotation mark, the first nonblank character in area B on the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continuation line must be blank.

If the indicator area of a continuation line contains no hyphen, the last character in the preceding line is assumed to be followed by a space.

Blank Lines

A blank line is one that is blank from margin C to margin R. A blank line can appear anywhere in the source program, except immediately before a continuation line. (Refer to "Continuation of Lines" above.)

DIVISION, SECTION, AND PARAGRAPH FORMATS

The division header must start in area A.

The section header must start in area A. A section consists of paragraphs in the Environment and Procedure Divisions, Control Division entries in the Control Division, and Data Division entries in the Data Division.

A paragraph consists of a paragraph-name followed by a period and a space and by zero, one, or more sentences, or a paragraph header followed by one or more entries. Comment lines may be included within a paragraph. The paragraph header or paragraph-name starts in area A of any line following the first line of a division or a section. The first sentence or entry in a paragraph begins either on the same line as the paragraph header or paragraph-name or in area B of the next nonblank line that is not a comment line. Successive sentences or entries begin either in area B of the same line as the preceding sentence or entry or in area B of the next nonblank line that is not a comment line. When the sentences or entries of a paragraph require more than one line, they may be continued as described in "Continuation of Lines" above.

DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or level-number followed by a space, followed by its associated name, followed by a sequence of independent descriptive clauses. Each clause, except the last clause of an entry, may be terminated by either the separator semicolon or the separator comma. The last clause is always terminated by a period followed by a space. There are two types of Data Division entries: those that begin with a level indicator and those that begin with a level-number.

Level indicators are FD, SD, RD, and CD. In 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 name and appropriate descriptive information.

> The associated name may also begin in area A.

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

> The associated record-name or item-name may also begin in area A.

Successive data description entries may have the same format as the first or may be indented according to level-number. 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 limited only by the length of a line.

DECLARATIVES

The key words DECLARATIVES and END DECLARATIVES precede and follow, respectively, the declarative portion of the Procedure Division. Each term must appear as the only entries on a line, must begin in area A, and must be followed by a period and a space.

COMMENT LINES

A comment line is any line with an asterisk in the continuation indicator area of the line. A comment line can appear as any line in a source program after the Identification Division header. Any combination of characters from the ASCII character set may be included in area A and area B of that line. The asterisk and the characters in area A and area B will be produced on the output listing but serve as documentation only.

A special form of comment line represented by a stroke (/) in the indicator area of the line causes page ejection prior to printing the comment.

Successive comment lines are allowed.

There may be comment lines between a continued line and the related continuation line.

UPPERCASE/LOWERCASE OUTPUT

Uppercase and lowercase letters do not affect the source listing. The user receives the source information exactly as written. However, in the cross-reference listing all user-words are translated to lowercase letters.

The source input to the compiler is from a Multics source segment. The segment is created by standard Multics facilities (the text editor edm or qedx commands, or from a card deck). It must have a name in the form name.cobol.

Each line has a maximum length of 256 characters and is terminated by a newline character.

A newline character is invalid in a literal. If detected in a literal, it is treated as an end of line, and the next line must be a continuation line.

SECTION 5

CONTROL DIVISION

DESCRIPTION OF THE CONTROL DIVISION

The Control Division is optional and consists of the Default Section. When it is included in the source program, the Control Division must be the first division in the program, preceding the Identification Division.

The Control Division consists of a single section, the Default Section. The Default Section allows the specification of certain defaults other than the standard compiler defaults.

STRUCTURE OF THE CONTROL DIVISION

The general format of the sections and clauses in the Control Division and their order of presentation in the source program is given below.

General Format:

```
[ CONTROL  DIVISION.

[ DEFAULT  SECTION.  [ default clauses ]  ...  . ] ]
```

The Default Section is optional and is included only if the standard compiler default values require changing.

DISPLAY SIGN

DISPLAY SIGN Clause

The DISPLAY SIGN clause specifies the representation and position of the sign on each DISPLAY data item without requiring the explicit use of a SIGN clause.

General Format:

```
[ DEFAULT FOR ] { DISPLAY SIGN IS { LEADING  } [ SEPARATE CHARACTER ] }
                                  { TRAILING }
```

Syntax Rules:

1. The DISPLAY SIGN clause applies only to numeric data description entries whose PICTURE contains an 'S' and whose usage is DISPLAY, explicitly or implicitly.
2. The DISPLAY SIGN clause is not applicable to a numeric data description entry that has an implicit (at the group entry) or explicit SIGN clause associated with it.

General Rules:

1. The DISPLAY SIGN clause specifies the default sign convention for signed numeric DISPLAY data items to which no SIGN clause applies.
2. If this clause is present, all rules specified under the SIGN clause in the Data Division apply to the explicit or implicit default option.
3. If this clause is not specified, the sign is trailing and not separate.

SECTION 6

NUCLEUS OF MULTICS COBOL

The nucleus of Multics COBOL provides the basic language capability for the internal processing of data. This section describes the Identification, Environment, and Data Divisions in the nucleus. The Procedure Division in the nucleus is contained in Section 7.

IDENTIFICATION DIVISION FOR THE NUCLEUS

The Identification Division must be included in every COBOL source program. This division identifies both the source program and the resultant output listing.

Paragraph headers identify the type of information contained in the paragraph. The program-name must be given in the PROGRAM-ID paragraph. The other paragraphs are optional and may be included at the discretion of the user, in the order of presentation shown in the general format.

Structure of the Identification Division

The general format of paragraphs in the Identification Division and their order of presentation in the source program are given below.

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] ... ]
```

PROGRAM-ID Paragraph

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

General Format:

PROGRAM-ID. program-name.

Syntax Rules:

1. The program-name must conform to the rules for formation of a user-defined word. (Refer to "User-Defined Words" in Section 2.)

General Rules:

1. The PROGRAM-ID paragraph must contain the name of the program and must be present in every program.
2. The program-name identifies the listings and the object program.

3. In Multics COBOL, program-name is an entry name. When referenced in a CALL or when invoked from a terminal, the program is referenced as object-name$program-name, where object-name is identical to the name of the source segment without the suffix ".cobol." If object-name and program-name are identical, the reference may be abbreviated to program-name; if they are different, the reference must include both names. For example, if the name of the source segment is Joe.cobol and the program-name is Sam, the program must referenced by the name Joe$Sam.

DATE-COMPILED

DATE-COMPILED Paragraph

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

General Format:

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

Syntax Rules:

1. The comment-entry can be any combination of characters from the computer's character set. The comment-entry may be contained on multiple lines; however, use of a hyphen in the indicator area to indicate continuation of the comment-entry is not permitted.

General Rules:

1. The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a DATE-COMPILED paragraph is included in the source program, it is replaced during compilation with a paragraph of the form:

   DATE-COMPILED. mm/dd/yy

   where: mm represents the month

   dd represents the day of the month

   yy represents the year

ENVIRONMENT DIVISION FOR THE NUCLEUS

The Environment Division provides a standard means for expressing those aspects of a data processing problem that depend upon the physical characteristics of a specific computer. This division must be included in every COBOL source program to specify the compiling and executing computers, must follow the Identification Division, and must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

The Configuration Section in the Nucleus describes source and object computer characteristics and is subdivided into three paragraphs:

- SOURCE-COMPUTER paragraph, which identifies the computer on which the source program is to be compiled.
- OBJECT-COMPUTER paragraph, which identifies the computer on which the object program produced by the compiler is to be executed.
- SPECIAL-NAMES paragraph, which associates hardware names and operating system features with the mnemonic-names used in the source program.

> The Environment Division and the sections and paragraphs within the Environment Division are optional.

Structure of the Environment Division

The general format of sections and paragraphs in the Environment Division and their order of presentation in the source program are given below.

General Format:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.    source-computer-entry
OBJECT-COMPUTER.    object-computer-entry
[SPECIAL-NAMES.     special-names-entry]
```

Configuration Section

The Configuration Section provides program documentation for the hardware characteristics of the computer used for compilation and of the computer used to execute the object program. Provisions are included in this section for relating specific hardware and operating system features to user-specified mnemonic-names.

Syntax Rules:

1. The Configuration Section must be included, must follow the Environment Division header, and must begin with the section-name CONFIGURATION SECTION followed by a period and a space.

SOURCE-COMPUTER PARAGRAPH

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

General Format:

```
SOURCE-COMPUTER. [HIS-SERIES-60] { MULTICS   } [WITH DEBUGGING MODE]
                                 { LEVEL-68  }
```

Syntax Rules:

1. The SOURCE-COMPUTER paragraph must begin with the paragraph-name SOURCE-COMPUTER followed by a period and a space.

General Rules:

1. This paragraph provides program documentation only and has no effect on compilation.

2. Refer to Section 13 for a description of the WITH DEBUGGING MODE clause.

OBJECT-COMPUTER

OBJECT-COMPUTER PARAGRAPH

The OBJECT-COMPUTER paragraph in the Configuration Section identifies the computer on which the program is to be executed, and specifies the collating sequence of the object program.

General Format:

```
OBJECT-COMPUTER. [HIS-SERIES-60] {MULTICS   }
                                 {LEVEL-68  }

    [ MEMORY SIZE integer {WORDS     } ]
                          {CHARACTERS}
                          {MODULES   }
    [ PROGRAM COLLATING SEQUENCE IS alphabet-name ]
```

Syntax Rules:

1. The OBJECT-COMPUTER paragraph must begin with the paragraph-name OBJECT-COMPUTER followed by a period and a space.

2. If the alphabet-name option of the PROGRAM COLLATING SEQUENCE phrase is specified, alphhabet-name must be defined in the SPECIAL-NAMES paragraph.

OBJECT-COMPUTER

General Rules:

1. The MEMORY SIZE phrase is used for program documentation only and has no effect on the object program.

2. If the PROGRAM COLLATING SEQUENCE phrase is specified, the collating sequence associated with alphabet-name is used to determine the truth value of any nonnumeric comparisons explicitly specified in relation conditions or in condition-name conditions, and the value of the figurative constants HIGH-VALUE and LOW-VALUE.

3. If the PROGRAM COLLATING SEQUENCE phrase is not specified, the native (ASCII) collating sequence is used.

4. If the PROGRAM COLLATING SEQUENCE phrase is specified, the collating sequence is that collating sequence associated with alphabet-name specified in that phrase.

5. The PROGRAM COLLATING SEQUENCE phrase is also applied to any nonnumeric sort or merge keys unless the COLLATING SEQUENCE phrase of the SORT or MERGE statement is specified.

6. The PROGRAM COLLATING SEQUENCE phrase applies only to the program in which it is specified.

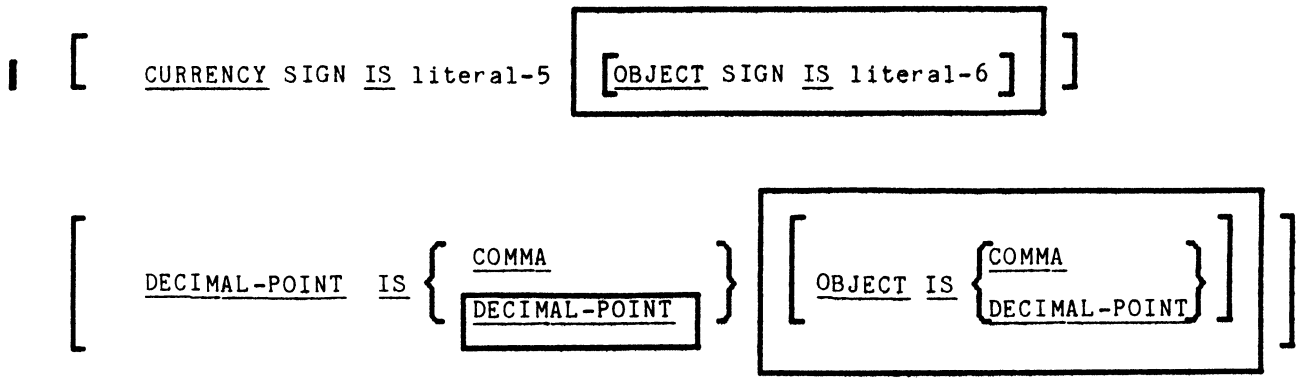
SPECIAL-NAMES

SPECIAL-NAMES PARAGRAPH

The SPECIAL-NAMES paragraph in the Configuration Section provides a means of relating specific hardware and operating system features to user-specified mnemonic-names and of relating alphabet-names to character sets and/or collating sequences.

General Format:

```
SPECIAL-NAMES.
  [ SYSIN IS mnemonic-name-1 ]   [ SYSOUT IS mnemonic-name-2 ]
  [ CONSOLE IS mnemonic-name-3 ]   [ HOF IS mnemonic-name-4 ]
  [ CHANNEL-m IS mnemonic-name-5 ]

  [
    SWITCH-n
                                                                    |
      { IS mnemonic-name-6 [ { ON STATUS IS condition-name-1 } ] }  |
      {                      { OFF STATUS IS condition-name-2 } }  |
      { { ON STATUS IS condition-name-1 }                           |
      { { OFF STATUS IS condition-name-2 }                          |
  ]

  [
    alphabet-name IS
                                                                    |
      { STANDARD-1                                                  |
      { NATIVE                                                      |
      { ASCII                                                       |
      { EBCDIC                                                      |
      { literal-1 [ {THROUGH} literal-2                         ] } |
      {           [ {THRU   }                                   ] } |
      {           [ ALSO literal-3 [ ALSO literal-4 ]... ]      ] } |
  ]
```



Syntax Rules:

1. Literals specified in the literal phrase of the alphabet-name clause:
   a. If numeric, must be unsigned integers and must be have a value within the range 1 through 128, the maximum number of characters in the ASCII character set.
   b. If nonnumeric and associated with a THROUGH or ALSO phrase, must each be one character in length.
2. If the literal phrase of the alphabet-name clause is specified, a given character must not be specified more than once in an alphabet-name clause.
3. The words THROUGH and THRU are equivalent.
4. Literal-5 must be a one-character nonnumeric literal.
5. Literal-6 must be a nonnumeric literal less than four characters in length.
6. In SWITCH-n, the value n must be in the range and must be 1 through 8.
7. In CHANNEL-m, the value m must be in the range and must be 1 through 16.

SPECIAL-NAMES

General Rules:

1. Mnemonic-name-1, associated with SYSIN, can be used only in an ACCEPT statement, in which case it specifies that data is to be accepted (read) from the I/O switch user_input, usually attached to the user's terminal.

2. Mnemonic-name-2, associated with SYSOUT, can be used only in a DISPLAY statement, in which case it specifies that data is to be displayed (written) on the I/O switch user_output, usually attached to the user's terminal.

3. Mnemonic-name-3, associated with CONSOLE, can be used only in an ACCEPT or DISPLAY statement, in which case it specifies that data is either to be accepted (read) from the I/O switch user_input or displayed (written) to the I/O switch error_output; both switches are usually attached to the user's terminal.

4. Mnemonic-name-4, associated with HOF, and mnemonic-name-5, associated with CHANNEL-m, can be used only in a WRITE or SEND statement within the BEFORE or AFTER phrase. CHANNEL-m indicates one of the printer control channels, where m may be a value from 1 to 16. The meaning of the various channels is site defined.

5. The external switches are represented by the specification of SWITCH-n, where n may be a value from 1 to 8. If SWITCH-n is specified, at least one condition-name must be associated with it. The status of the switch is specified by condition-names and interrogated by testing the condition-names. (Refer to "Switch-Status Condition" in Section 7.)

6. Mnemonic-name-6 is associated with the name of an external switch (not the condition of that switch). It can be set only by the SET statement. (Refer to "SET Statement" in Section 7.)

7. 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 (refer to "OBJECT-COMPUTER Paragraph" in this section) or the COLLATING SEQUENCE phrase of a SORT statement (refer to Section 10), the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry (refer to "File Description - Complete Entry Skeleton" in Section 9), 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 X3.4-1968, Code for Information Interchange.

   b. NATIVE and ASCII are synonymous with STANDARD-1.

   c. If the EBCDIC phrase is specified, the character code set or collating sequence identified is that defined by EBCDIC.

SPECIAL-NAMES

d. If the literal phrase is specified, the alphabet-name may not be referenced in the CODE-SET clause. (Refer to "CODE-SET Clause" in Section 9.) The collating sequence identified is that defined according to the following rules:

1) The value of each literal specifies:

- The ordinal number of a character within the ASCII character set, if the literal is numeric. This value must not exceed 128, the value of the number of printable ASCII characters.

- The actual character within the ASCII character set, if the literal is nonnumeric. If the value of the nonnumeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

2) The order in which the literals appear in the alphabet-name clause specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified.

3) Any characters within the ASCII collating sequence, which are not explicitly specified in the literal phrase, assume a position, in the collating sequence being specified, greater than any of the explicitly specified characters. The relative order within the set of these unspecified characters is unchanged from the ASCII collating sequence.

4) If the THROUGH phrase is specified, the set of contiguous characters in the ASCII character set beginning with the character specified by the value of literal-1, and ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of contiguous characters specified by a given THROUGH phrase may specify characters of the ASCII character set in either ascending or descending sequence.

5) If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1, literal-3, literal-4, etc. are assigned to the same position in the collating sequence being specified.

8. The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position in the program collating sequence, the last character specified is associated with the figurative constant HIGH-VALUE.

9. The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position in the program collating sequence, the first character specified is associated with the figurative constant LOW-VALUE.

10. Literal-5 which 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:

    a. Digits 0 through 9

    b. Alphabetic characters A, B, C, D, L, P, R, S, V, X, Z, a thru z, or the space. The characters must be in uppercase, even though corresponding characters in the picture can be in either uppercase or lowercase.

    c. Special characters * + - , . ; ( ) " / or =

11. The DECIMAL-POINT IS COMMA clause indicates that the functions of the comma and the period are exchanged in the character-string of the PICTURE clause and in numeric literals. For example, the symbol used for the decimal point is the comma; otherwise, the period is used. (Refer to "Numeric Literals" in Section 2 and to "PICTURE Clause" in this section for further information.)

12. Literal-6 which appears in the OBJECT SIGN IS literal clause, is used at object time, while editing, to represent the currency symbol. When the OBJECT SIGN clause is not present, literal-5 is used instead of literal-6. If neither clause is present the dollar sign ($) is used.

13. The clause DECIMAL-POINT IS DECIMAL-POINT means that the function of the comma and period are those specified by default. This clause is meaningful when the clause OBJECT IS COMMA is also used.

14. The clause OBJECT IS COMMA means that the comma is used at object time, while editing, to represent the decimal point, and the period to represent the fixed insertion comma. The clause OBJECT IS DECIMAL-POINT means that the period is used at object time, while editing, to represent the decimal point, and the comma to represent the fixed insertion comma. When neither clause is used, the same characters are used at object time as those specified in the PICTURE clause.

DATA DIVISION FOR THE NUCLEUS

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output.

The Working-Storage Section in the Nucleus describes records and noncontiguous data items that are not part of external data files but are developed and processed internally.

> The Constant Section is used to describe data items whose values are assigned in the source program and do not change during the execution of the object program.

Structure of the Data Division

The Data Division is prepared in accordance with the reference format described in Section 4. The Data Division is identified by and must begin with the division header DATA DIVISION followed by a period and a space. The general format of the Data Division in the Nucleus is given below.

General Format:

```
DATA DIVISION.
[
 WORKING-STORAGE SECTION.
[ 77-Level-description-entry ]
[ record-description-entry   ] ...

[ CONSTANT SECTION.
[ 77-level-description-entry ]
[ record-description-entry   ] ... ]
```

Working-Storage Section

The Working-Storage Section is composed of the section header, followed by data description entries for noncontiguous data items or record description entries. Each Working-Storage Section record-name and noncontiguous item name must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

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 to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number 77.

The following data clauses are required in each data description entry:

- Level-number 77
- Data-name
- The PICTURE clause or the USAGE IS INDEX, COMP-6 or COMP-7 clause.

Other data description clauses are optional and can be used to complete the description of the item, if necessary.

WORKING-STORAGE RECORDS

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

WORKING-STORAGE INITIAL VALUES

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

Constant Section

The Constant Section is like the Working-Storage Section except that in the Constant Section:

- All data must have a VALUE clause.

  NOTE: Because of the REDEFINES and RENAMES clauses, a VALUE clause must be used with every data item, not with every data description.

- The items can be referenced only where literals can be referenced; that is, their contents cannot be altered during program execution.
- The USAGE IS INDEX clause may not be used.
- The OCCURS clause may not be used.

DATA DESCRIPTION SKELETON

Data Description - Complete Entry Skeleton

A data description entry specifies the characteristics of a particular item of data. A detailed data description consists of a set of entries and each entry defines the characteristics of a particular unit of data. The general formats of the detailed data description entry, syntax rules, and general rules follow. The individual clauses are described later in this section.

Format 1:

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

    [ REDEFINES data-name-2 ]

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

    [                 { DISPLAY               } ]
    [                 { COMPUTATIONAL         } ]
    [                 { COMP                  } ]
    [                 {                       } ]
    [ [ USAGE IS ]    { COMPUTATIONAL-5       } ]
    [                 { COMP-5                } ]
    [                 { COMPUTATIONAL-6       } ]
    [                 { COMP-6                } ]
    [                 { COMPUTATIONAL-7       } ]
    [                 { COMP-7                } ]
    [                 { COMPUTATIONAL-8       } ]
    [                 { COMP-8                } ]

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

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

    [ { JUSTIFIED } RIGHT ]
    [ { JUST      }       ]

    [ BLANK WHEN ZERO ]
    [ VALUE IS literal ] .
```

DATA DESCRIPTION SKELETON

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    }
```

Syntax Rules:

1. The level-number in Format 1 can be any number from 01 to 49 or 77.
2. The clauses can 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 data item except an index data item, in which case the use of this clause is prohibited.
4. The words THRU and THROUGH are equivalent.

DATA DESCRIPTION SKELETON

General Rules:

1. The SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO clauses must not be specified except for an elementary data item.

2. 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 that contains a level-number except the following:

   - Another condition-name
   - A level 66 item
   - A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY)
   - An index data item (Refer to "USAGE Clause" in Section 8.)

BLANK WHEN ZERO

BLANK WHEN ZERO CLAUSE

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

General Format:

BLANK WHEN ZERO

Syntax Rules:

1. The BLANK WHEN ZERO clause can be used only for an elementary data item whose PICTURE is specified as numeric or numeric edited.

2. This clause cannot be used with variable-length items.

General Rules:

1. When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

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 or FILLER

DATA-NAME OR FILLER CLAUSE

A data-name clause specifies the name of the data being described. The FILLER clause specifies an elementary item of the logical record that cannot be referred to explicitly.

General Format:

level-number { data-name / FILLER }

Syntax Rules:

1. In the File Section, Working-Storage Section, Linkage Section, Communication Section, and Constant Section, 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 reference to the FILLER data item, but to its value.

In Multics COBOL the key word FILLER may be used to name a non-elementary item in a record.

JUSTIFIED CLAUSE

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

General Format:

```
{JUSTIFIED}
{JUST     }  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.
4. The JUSTIFIED clause must not be specified for an index data item.

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. Refer to "Standard Alignment Rules" in Section 2.

LEVEL-NUMBER

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working-storage data items, linkage data items, Constant Section 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, SD, or CD entry must have level-numbers with the values 01 through 49, 66, or 88.
3. Data description entries in the Working-Storage Section, Constant Section, and Linkage Section must have level-numbers with the values 01 through 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 and Constant Section data items, noncontiguous linkage data items, and must be used only as described in Format 1 of the data description skeleton.
   b. Level-number 66 is assigned to identify RENAMES entries and must be used only as described in Format 2 of the data description skeleton.
   c. Level-number 88 is assigned to entries which define condition-names associated with a conditional variable and must be used only as described in Format 3 of the data description skeleton.
3. Multiple level 01 entries that are subordinate to any given level indicator represent implicit redefinitions of the same area.

PICTURE

PICTURE CLAUSE

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

General Format:

{PICTURE / PIC} IS character-string

Syntax Rules:

1. A PICTURE clause may be specified only at the elementary item level.
2. The 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 data item except an index data item, or a data item with USAGE IS COMP-6 or COMP-7, in which case use of this 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. Five categories of data may be described with a PICTURE clause; alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.
   - To define an item as alphabetic:
     a. Its PICTURE character-string can contain only the symbols 'A', 'B'; and
     b. Its contents when represented in standard data format must be one or more alphabetic characters.

- To define an item as numeric:

  a. Its PICTURE character-string can contain only the symbols '9', 'P', 'S', and 'V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive; and

  b. If unsigned, its contents when represented in standard data format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9'; if signed, the item can also contain a '+', '-', or other representation of an operational sign (see the SIGN clause).

- 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; and

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

- 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

     (1) The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X' or at least one '/' (stroke) and at least one 'X'; or

     (2) The character-string must contain at least one '0' (zero) and at least one 'A' or at least one '/' (stroke) and at least one 'A'; and

  b. The contents when represented in standard data format are allowable characters in the computer's character set.

- To define an item as numeric edited:

  a. Its PICTURE character-string is restricted to certain combinations of the 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; and

     (1) The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive; and

     (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 these symbols that are allowed to represent a digit in standard data format must be one of the numerals.

2. 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 which is 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 can appear only once in a given PICTURE: 'S', 'V', '.', 'CR', and 'DB'.

3. 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 which 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 and 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 'P's within a PICTURE description. Since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE characters and to the right if 'P's 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 '.' (period) 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 zero, 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 not the position of an operational sign; it must be written as the leftmost character in the PICTURE. The 'S' is counted in determining the size (in terms of standard data format characters) of the elementary item.

V The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear 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 which contains any allowable character from the computer's character set.

Z Each 'Z' in a character-string can only be used to represent the leftmost leading numeric character positions which will be replaced by a space character when the contents of that character position are zero. Each 'Z' is counted in the size of the item.

9 Each '9' in the character-string represents a character position which 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 zero 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 which 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. Both CR and DB, after editing, will appear in uppercase letters in the receiving data item.

* Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position are zero. Each '*' is counted in the size of the item.

PICTURE

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.

4. Any character appearing in a picture, that is either not one of the above-defined symbols or is in violation of precedence rules (see below) is treated as a simple insertion editing character. For example,

```
01 x pic 99:99.

move 123 to x.

--> 01:23.
```

Editing Rules:

1. Two general methods are used to perform editing in the PICTURE clause, either by insertion or by suppression and replacement. The four types of insertion editing are:

   - Simple insertion
   - Special insertion
   - Fixed insertion
   - Floating insertion

   The two types of suppression and replacement editing are:

   - Zero suppression and replacement with spaces
   - Zero suppression and replacement with asterisks

2. The type of editing that may be performed upon an item depends on the category to which the item belongs. The following list indicates 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 the restrictions of 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 is inserted. Any other character not having special meaning in the PICTURE character-string is treated as a simple insertion character. An observation diagnostic is issued in this case, since this is an extension of ANS COBOL-74.

5. Special Insertion Editing.

The '.' (period) is used as the insertion character and 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 not allowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

6. Fixed Insertion Editing.

The currency symbol and the editing sign control symbols '+', '-', 'CR', '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 symbol '+' or a '-', 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 may 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 results shown in Table 6-1, depending on the data item value.

Table 6-1. Results of Fixed Insertion Editing Symbols

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

7. Floating Insertion Editing.

The currency symbol and editing sign control symbols '+' or '-' are the floating insertion characters and as such 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 can 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. Nonzero 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 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 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 nonzero 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 zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point in the PICTURE character-string.

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 fixed insertion characters being edited into the receiving data item, plus one for the floating insertion character.

8. Zero Suppression Editing.

The suppression of leading zeros 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; 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 which 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 is to represent any or all of the leading numeric

PICTURE

character positions to the left of the decimal point by suppression symbols. The other 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 zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero 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 zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is 'Z', the entire data item will be spaces. If the value is zero and the suppression symbol is '*', the data item will be all '*' except for the actual decimal point.

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 chart in Table 6-2 shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbols at the top of the column may precede, in a given character-string, the symbols at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by '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.

Nonfloating insertion symbols '+' and '-', floating insertion symbols 'Z', '*', '+', '-', and 'cs', and other symbol 'P' appear twice in the PICTURE character precedence chart. The leftmost column and uppermost row for each symbol represents 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.

Table 6-2. 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

REDEFINES CLAUSE

The REDEFINES clause allows the same memory 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 included 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.

3. The REDEFINES clause must not be used in level 01 entries in the File Section. Refer to the DATA RECORDS clause.

4. This clause must not be used in level 01 entries in the Communication Section.

5. The data description entry for data-name-2 cannot contain a REDEFINES clause; however, data-name-2 may be subordinate to an entry that 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 cannot be subscripted or indexed. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS clause.

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

REDEFINES

General Rules:

1. Redefinition begins at 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.

> In Multics COBOL the data description entry defined by data-name-1 need not be the same size as the data descripion entry referenced by data-name-2.

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 VALUE clauses, except in condition-name entries.

5. Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area.

RENAMES CLAUSE

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 included 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 level 66 entry cannot rename another level 66 entry nor can it rename a level 01, 77, or 88 entry.

3. Data-name-1 must not be used as a qualifier, and can be qualified only by the names of the associated level 01, FD, CD, or SD entries.

4. 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.

5. 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.

6. Data-name-2 and data-name-3 can be qualified.

7. The words THRU and THROUGH are equivalent.

8. 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.

RENAMES

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 CLAUSE

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:

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

Syntax Rules:

1. The SIGN clause can be specified only for a numeric data description entry whose PICTURE contains the character 'S', or for 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. If no USAGE is specified, DISPLAY is assumed.

3. Only one SIGN clause can apply to any given numeric data description entry.

4. If the CODE-SET clause is specified, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

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 not the position of the operational sign.

2. A numeric data description entry whose picture contains the character 'S', but not the SIGN IS SEPARATE clause, has an operational sign. If the LEADING or TRAILING phrase is specified the sign is associated with the high-order or low-order digit, respectively, of the data item. If neither is specified, the sign is associated with the low-order digit of the data item.

SIGN

3. If SEPARATE CHARACTER is specified,

   a. The operational sign is assumed to be 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.

4. If SEPARATE CHARACTER is not specified,

   a. The operational sign is assumed to be associated with the leading or trailing digit of the elementary numeric data item.

   b. The letter 'S' in a PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

5. Every numeric data description entry whose PICTURE contains the character 'S' is a signed numeric data description entry. If conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

6. Signs resulting from arithmetic operations and initial values are:

   \+ = 53v8^ (graphic +)

   \- = 55v8^ (graphic -)

SYNCHRONIZED

SYNCHRONIZED CLAUSE

The SYNCHRONIZED clause specifies the alignment of an elementary data item on the natural boundaries of the computer memory.

General Format:

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

Syntax Rules:

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

3. LEFT and RIGHT are accepted and treated as documentation only.

General Rules:

1. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries delimiting this data item. If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions (or portions thereof) must not be 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. The SYNCHRONIZED clause only has meaning for COMP-7 and COMP-8 data. For COMP-7, a full word is allocated with the data occupying the rightmost 18 bits. For COMP-8, data is allocated on a byte boundary left adjusted and occupies an integral number of bytes.

3. 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.

4. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data

SYNCHRONIZED

description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:

a. Each occurrence of the data item is SYNCHRONIZED.

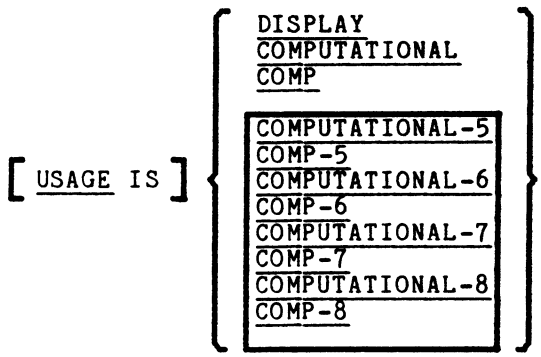
b. Any implicit FILLER generated for other data items within that same table are generated for each occurrence of those data items.

5. See "Standard Alignment Rules" in Section 2 for additional details.

USAGE CLAUSE

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

General Format:



Syntax Rules:

1. The PICTURE character-string of a COMPUTATIONAL data item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', and one or more 'P's.

2. COMP is an abbreviation for COMPUTATIONAL.

COMP-n is an abbreviation for COMPUTATIONAL-n.

3. COMP and COMP-5 have the same meaning.

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. 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).

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

USAGE

4. DISPLAY data items are ASCII characters. Signed numeric data has the sign associated with the rightmost position of the data item unless LEADING is specified in the SIGN clause. In this case, the sign is associated with the leftmost position of the data item. (Refer to the SIGN clause.) A plus sign combined with the digits 1 through 9 is represented by the characters A through I respectively. A plus sign combined with the digit 0 is represented by the character left brace ([) (octal 173). A minus sign combined with the digits 1 through 9 is represented by the characters J through R respectively. A minus sign combined with 0 is represented by the character right brace (]) (octal 175).

5. COMP, COMP-5, COMPUTATIONAL, and COMPUTATIONAL-5 data items are packed decimal. A digit is represented by 4 bits, right-justified in a byte. A byte may represent 2 digits. The leftmost bit of byte is always 0. If the data item is signed, the sign is in right half byte of the rightmost byte; i.e., it is a trailing sign. The plus sign is octal 13 ("1011"b). The minus sign is octal 15 ("1101"b).

6. COMP-6 is a fixed twos complement binary integer of 36 bits. A PICTURE clause cannot be specified for this data item.

7. COMP-7 is a fixed twos complement binary integer of 18 bits. A PICTURE clause cannot be specified for this data item.

8. COMP-8 data items are packed decimal. A digit is represented by four bits; however, such items need not start on a byte boundary. A byte may represent two digits. The leftmost bit of the byte is always 0. If the data item is signed, the sign occupies the leftmost half byte of data; i.e., it is a leading sign. The plus sign is octal 13 ("1011"b). The minus sign is octal 15 ("1101"b).

9. Refer to the Multics COBOL Users' Guide for additional information.

10. Refer to Section 2, "Standard Alignment Rules."

11. For COMP-7, and COMP-8 see Standard Alignment Rules when synchronization is of importance.

VALUE

VALUE CLAUSE

The VALUE clause defines the value of constants, the initial value of working-storage data items, and the values associated with a condition-name.

Format 1:

```
VALUE IS literal
```

Format 2:

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

Syntax Rules

1. The words THRU and THROUGH are equivalent.
2. A signed numeric literal must be associated with a signed numeric PICTURE character-string.
3. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric 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 data item, the literal is aligned in the data item according to the standard alignment rules (see Section 2).
   b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. Editing characters in the PICTURE clause are included in determining the size of the data item but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.
   c. Initialization takes place independent of any BLANK WHEN ZERO clause or JUSTIFIED clause that may be specified.

2. A figurative constant may be substituted in both 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. Wherever the THRU phrase is used, literal-1 must be less than literal-2, 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 in the Linkage Section, the VALUE clause can be used only in condition-name entries.

   b. In the Working-Storage Section and in the Communication Section, the VALUE clause must be used in condition-name entries. The VALUE clause can also be used to specify the initial value of any other data item; 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's description, the initial value is undefined.

   c. In the Constant Section, the VALUE clause must be used for every data item defined, but not necessarily for every data description, due to redefines and renames capabilities.

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.

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 nonnumeric 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 the JUSTIFIED, SYNCHRONIZED, or USAGE clauses (other than USAGE IS DISPLAY).

SECTION 7

PROCEDURE DIVISION FOR THE NUCLEUS

PROCEDURE DIVISION FUNCTIONS

The Procedure Division contains the procedures required to solve a given problem and must be included in every Multics COBOL source program. The Procedure Division is written as statements, combined to form sentences, combined to form paragraphs under paragraph-names, which in turn can be combined into sections under section-names.

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 operators, and parentheses are given below in Table 7-1.

Identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

Arithmetic Operators

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

Formation and Evaluation Rules

Rules for the formation and evaluation of arithmetic expressions are:

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

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

2. Parentheses are 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 some deviation from the normal precedence is required. 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. Methods in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in Table!7-1.

4. An arithmetic expression can begin only with the symbols (, +, or -, or a variable, and can end only with a) or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression, so that each left parenthesis is to the left of its corresponding right parenthesis.

5. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands or receiving data items. (Refer to "Arithmetic Statements" later in this section.)

Table 7-1. Combination of Symbols in Arithmetic Expressions

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

NOTES:
1. The letter "P" indicates a permissible pair of symbols.
2. The character "-" represents an invalid pair.
3. "Variable" represents an identifier or literal.

CONDITIONAL EXPRESSIONS

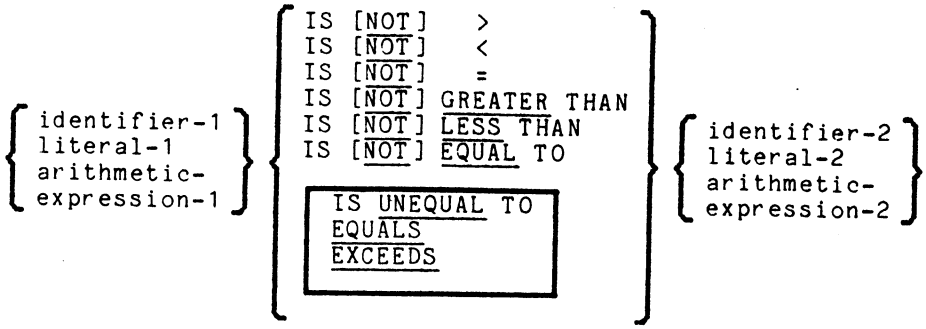
Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF, PERFORM, and SEARCH statements. Two categories of conditions are associated with conditional expressions: simple conditions and complex conditions. Each may be enclosed within any number of paired parentheses, in which case its category is not changed.

Simple Conditions

Simple conditions are the relation, class, condition-name, switch-status, and sign conditions. A simple condition has a truth value of true or false. When simple conditions are included within parentheses, the simple truth value does not change.

A relation condition causes a comparison of two operands, each of which can be the data item referenced by an identifier, a literal, or the value resulting from an arithmetic expression. A relation condition has a truth value of true if the relation exists between the operands. Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

The general format of a relation condition is:



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 relation condition must contain at least one reference to a variable.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word of which the relational operator is composed. 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; that is, NOT EQUAL is a truth test for an unequal comparison and NOT GREATER is a truth test for an equal or less comparison. The meaning of the relational operators is:

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

Comparison of Numeric Operands

For operands whose class is numeric, 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 represented, 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.

Comparison of Nonnumeric Operands

For two nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. Refer to the PROGRAM COLLATING SEQUENCE phrase of the OBJECT-COMPUTER paragraph. If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

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. (Refer to the MOVE statement and the character P in 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.

A noninteger numeric operand cannot be compared to a nonnumeric operand.

Comparison of Operands of Equal or Unequal Sizes

The size of an operand is the total number of standard data format characters in the operand. Numeric and nonnumeric operands can be compared only when their usage is the same. There are two cases to consider:!!operands of equal size and operands of unequal 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.

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

CLASS CONDITION

The class condition determines whether the operand is numeric, consisting entirely of the characters 0 through 9 (with or without the operational sign), or alphabetic, consisting entirely of the characters A through Z, a through z, and the space. The general format for the class condition is:

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

The usage of the operand being tested must be described as DISPLAY. When used, NOT and the next key word specify one class condition that defines the class test to be executed for truth value; that is, NOT NUMERIC is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational signs. 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.

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, a through z, and the space.

CONDITION-NAME CONDITION

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:

```
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.

SWITCH-STATUS CONDITION

A switch-status condition determines the "on" or "off" status of an implementor-defined switch. The implementor-name 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 of the status-switch 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.

SIGN CONDITION

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero. The general format for a sign condition is:

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

When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value; for example, NOT ZERO is a truth test for a nonzero (positive or negative) value. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. The arithmetic expression must contain at least one reference to a variable.

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions, 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.

NEGATED SIMPLE CONDITIONS

A simple condition is negated by using the logical operator NOT. The negated simple condition results in the opposite truth value for a simple condition. Thus, the truth value of a negated simple condition is true only if the truth value of the simple condition is false; the truth value of a negated simple condition is false only if the truth value of the simple condition is true. If a negated simple condition is included within parentheses, the truth value does not change. The general format for a negated simple condition is:

NOT simple-condition

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 / OR } condition } ...

where "condition" can be:

- A simple condition
- A negated simple condition
- A combined condition
- A negated combined condition; that is, the NOT logical operator followed by a combined condition enclosed within parentheses
- Combinations of the above, stated according to the rules specified in Table 7-2

Although parentheses need never be used when either AND or OR (but not both) is used exclusively in a combined condition, parentheses can be used to effect a final truth value when a mixture of AND, OR, and NOT is used. (Refer to "Condition Evaluation Rules" below.)

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 7-2. Combinations of Conditions, Logical Operators, and Parentheses

Given the following element	Location in conditional expression		In a left-to-right sequence of elements:	
	First	Last	Element, when not first, can be immediately preceded by only:	Element, when not last, can 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.

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 is 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:

- Omitting the subject of the relation condition
- Omitting the subject and relational operator of the relation condition

The format for an abbreviated combined relation condition is:

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

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 is inserted in place of the omitted subject, and the last stated relational operator is inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules in Table 7-2. This insertion of an omitted subject and/or relational operator terminates after 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 or symbol immediately following NOT is GREATER, >, LESS, <, 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))))

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. (Refer to "Formation and Evaluation Rules" above.

2. Truth values for simple conditions are established in the following order:

   - Relation (following the expansion of any abbreviated relation condition)
   - Class
   - Condition-Name
   - 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.

COMMON PHRASES IN STATEMENT FORMATS

In the statement descriptions of the Procedure Division, several phrases appear frequently: ROUNDED, SIZE ERROR, and CORRESPONDING.

In the following paragraphs, the term "resultant-identifier" is that identifier associated with a result of an arithmetic operation.

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 one (1) in the least significant digit whenever the most significant digit of the excess is greater than or equal to five (5).

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

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 zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and not to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case 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 values of those resultant-identifiers affected are undefined. Values of resultant-identifiers for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifiers during execution of this operation. Normally, when a size error occurs and the SIZE ERROR phrase is not specified, program execution terminates, and control returns to command level.

2. If the SIZE ERROR phrase is specified and a size error condition occurs, the values of resultant-identifiers affected by the size errors are not altered. Values of resultant-identifiers for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifiers 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 produce a size error condition, the imperative statement in the SIZE ERROR phrase is not executed until all individual additions or subtractions are completed.

CORRESPONDING Phrase

For the purpose of this discussion, d_1 and d_2 represent 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_1 and a data item in d_2 are not designated by the key word FILLER and have the same data-name and the same qualifiers up to, but not including, d_1 and d_2.

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

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; data items subordinate to the data item that contains such a clause are also ignored. However, d_1 and d_2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

5. If no data item corresponds (i.e., no ADD, SUBTRACT, or MOVE takes place), a fatal diagnostic is produced at compilation time.

ARITHMETIC STATEMENTS

The arithmetic statements are ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT. 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.

3. Computations in the object program must frequently use data items to hold intermediate values. These items, called temporary operands, are 30 digits in size, are invisible to the programmer, and hold any interim values generated by an arithmetic operation. A number of most significant nonzero digits, which can be set by the NUMERIC-LIMIT clause, are retained at all times. Unless otherwise specified, the default is 30. In an arithmetic operation, when the size of the value exceeds that number of digits, the least significant (rightmost) digits are truncated.

4. A size error can occur if the final results of an arithmetic operation are too large for the receiving field. The size error occurs when an attempt is made to store the intermediate data item, not during actual computation. During computation, a size error can occur only under certain conditions, such as when the user attempts to divide by zero or violates a rule of exponentiation. (Refer to "Exponentiation in Arithmetic Expressions" below.)

Overlapping Operands

When sending and receiving items in an INSPECT, SET, STRING, or UNSTRING statement share a part of their storage areas, the result of the execution of such a statement is undefined.

When sending and receiving items in an arithmetic statement or a MOVE statement share a part of their storage areas, the statement operates as if the sending operand is moved to a temporary operand and then the temporary operand participates in the operation.

Multiple Results in Arithmetic Statements

Arithmetic statements can have multiple results. Such statements behave as though they are written in the following way:

1. A statement which 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 provided by the compiler.

Exponentiation in Arithmetic Expressions

The rules that apply to exponentiation in an arithmetic expression are as follows:

1. An exponent is an arithmetic expression.
2. When an expression to be raised to a power has the value of zero, the exponent must have a positive value greater than zero.
3. When an expression to be raised to a power has a negative value, the exponent must have an integral value.
4. If a Multics COBOL user violates either rule 2 or 3, a size error condition occurs at execution time. (Refer to "SIZE ERROR Phrase" above.)

INCOMPATIBLE DATA

Except for the class condition, when the contents of a data item are referenced in the Procedure Division and the contents of that data item are incompatible with the class specified for that data item by its PICTURE clause, the result of such a reference is undefined.

NON-INPUT/OUTPUT ERRORS

Non-input/output (I/O) errors can be detected by the processing equipment, the Multics operating system, or by instructions generated by the compiler. An out-of-range subscript is an example of a non-input/output error. The following conditions occur as a result:

1. If the compiler can anticipate the error (for example, an out of range subscript), a COBOL error message describing the error in COBOL terms is issued. It gives the program-name, the external line number, and the address (segment number and offset). These error messages are described in the Multics COBOL Users' Guide. The error condition is then signalled.
2. If the compiler cannot anticipate the error (for example, record quota overflow) the error condition is signalled. If a default handler is not provided for the error, Multics issues an error message and goes to command level. Refer to the MPM Reference Guide for descriptions of signalling error conditions, default handlers, and command level.
3. Error messages are written on the I/O module error_output, which is normally attached to the user's terminal.

ACCEPT STATEMENT

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

Format 1:

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

Format 2:

```
ACCEPT identifier FROM { DATE
                       { DAY
                       { TIME
                       { DAY-OF-WEEK
```

Syntax Rules:

1. Mnemonic-name must be specified in the SPECIAL-NAMES paragraph (Section 6) of the Environment Division and must be associated with either SYSIN or CONSOLE.

General Rules:

Format 1:

1. The ACCEPT statement causes data to be transferred from the hardware device or system file specified by the FROM phrase. This data replaces the contents of the data item named by the identifier.

2. The FROM phrase is for documentation only. Data is input from the I/O switch user_input, which is normally attached to the user's terminal.

3. The transfer of data is terminated when a newline character is encountered in the data. The newline character is not stored in the identifier.

4. If the size of the receiving data item exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item with blank-fill on the right.

5. If the size of the transferred data exceeds the size of the receiving data item, only the leftmost characters of the transferred data are stored in the receiving data item. The remaining characters of the transferred data which do not fit into the receiving data item are discarded.

6. No conversion takes place when storing data in the identifier. Data is stored in memory exactly as it is received.

Format 2:

1. The ACCEPT statement causes the information requested to be transferred to the data item specified by 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.

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

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

4. TIME is composed of the data elements hours, minutes, seconds, and hundredths of a second. TIME is based on eln elapsed time after midnight on a 24-hour clock basis; thus, 2:41 P.M. is expressed as 14410000. TIME, when accessed by a COBOL program, reacts as if it is described in the 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.

5. DAY-OF-WEEK is composed of a single data element whose content represents the day of the week. When accessed by a COBOL program, DAY-OF-WEEK behaves as if it had been described in the COBOL program as an unsigned elementary numeric integer data item one digit in length. In DAY-OF-WEEK, the value 1 represents Monday, 2 represents Tuesday, ..., 7 represents Sunday.

ADD STATEMENT

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

Format 1:

```
ADD { identifier-1 } ... TO { identifier-m [ ROUNDED ] } ...
    { literal-1    }

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-ADD ]
```

Format 2:

```
ADD { identifier-1 } ... GIVING { identifier-m [ ROUNDED ] } ...
    { literal-1    }

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-ADD ]
```

Format 3:

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

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-ADD ]
```

ADD

Syntax Rules:

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 each identifier 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 decimal digits.

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

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

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

4. CORR is an abbreviation for CORRESPONDING.

General Rules:

1. If Format 1 is used, the values of the operands preceding the word TO are added together. That sum is then added to the current value of identifier-m, etc. and the result is stored immediately into identifier-m, etc.; this process is repeated respectively for each operand following the word TO.

2. If Format 2 is used, the values of the operands preceding the word GIVING are added together and this sum is stored as the new value of each identifier-m, etc., the resultant-identifiers.

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

4. For large operands, the compiler uses the temporary operand described under "Arithmetic Statements" earlier in this section.

5. Refer to "Common Phrases in Statement Formats" earlier in this section for an explanation of the uses of the ROUNDED, SIZE ERROR, and CORRESPONDING phrases and arithmetic statements.

ALTER STATEMENT

The ALTER statement modifies a predetermined sequence of operations.

General Format:

```
ALTER { procedure-name-1  TO  [ PROCEED TO ]  procedure-name-2 } ...
```

Syntax Rules:

1. The token(s), procedure-name-1, etc. is the name of a paragraph containing a single sentence consisting of a GO TO statement without the DEPENDING phrase.
2. The token(s), procedure-name-2, etc. is the name of a paragraph or section in the Procedure Division.

General Rules:

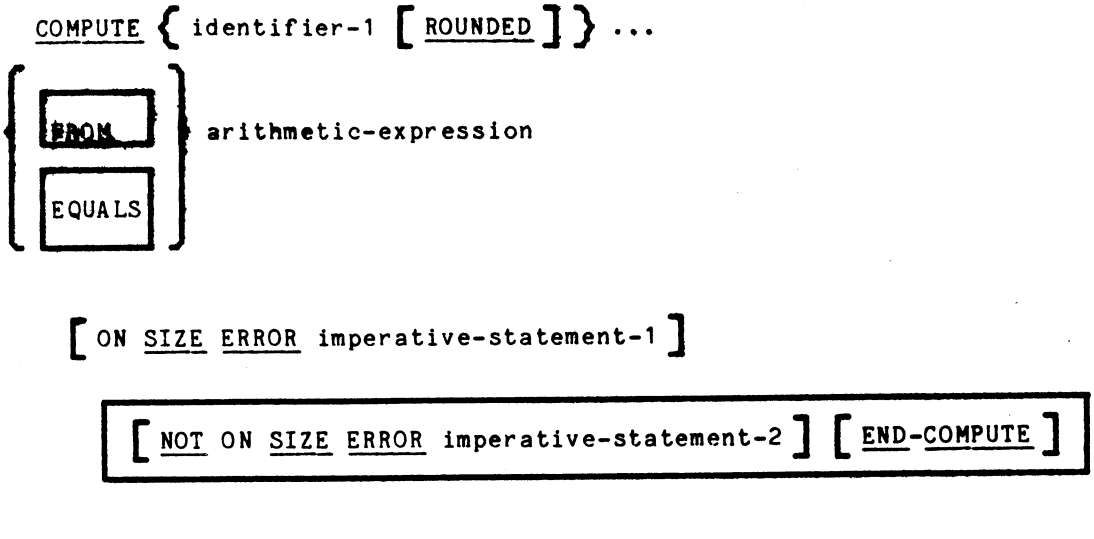
1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, etc. so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2, etc. respectively.
2. A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

3. Use of the "ALTER" statement should be avoided; use of the "GO TO...DEPENDING" statement is preferred.

COMPUTE STATEMENT

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

General Format:



Syntax Rules:

1. Identifiers that appear to the left of a FROM, EQUALS, or = must refer to either an elementary numeric item or an elementary numeric edited item.

General Rules:

1. An arithmetic expression consisting of a single identifier or literal provides a method of setting the values of identifier-1 etc., equal to the value of the single identifier or literal.

2. If more than one identifier is specified for the result of the operation, and precedes a FROM, EQUALS, or =, the value of the arithmetic expression is computed, and this value is stored as the new value of each of the data items referenced by identifier-1 etc., in turn.

3. The effect of the COMPUTE statement is the same as if a series of individual arithmetic operations are written. Refer to "Arithmetic Statements" earlier in this section for details of intermediate results and truncation.

4. Refer to "Common Phrases in Statement Formats" earlier in this section for an explanation of the uses of the ROUNDED and SIZE ERROR phrases and arithmetic statements.

CONTINUE STATEMENT

The CONTINUE statement is a no-operation statement. It indicates that no executable statement is present.

General Format:

CONTINUE

Syntax Rules:

1. The CONTINUE statement may be used anywhere a conditional statement or imperative statement is used.

General Rules:

1. The CONTINUE statement has no effect on the execution of the program.

DISPLAY STATEMENT

The DISPLAY statement causes a line or lines of characters to be transferred to a hardware device or system output file.

General Format:

```
DISPLAY { identifier-1 } ... [ UPON mnemonic-name ]
        { literal-1    }
```

Syntax Rules:

1. Mnemonic-name must be specified in the SPECIAL-NAMES paragraph (Section 6) of the Environment Division and must be associated with either SYSOUT or CONSOLE.
2. Each literal can 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 user I/O switch specified by the UPON phrase in the order listed.
2. If the UPON phrase is not specified, SYSOUT is assumed.
3. If SYSOUT is implicit or is specified by mnemonic-name associated with SYSOUT, data is transferred to the I/O switch user_output, which is normally attached to the user's terminal.

   If CONSOLE is specified by a mnemonic-name associated with CONSOLE, data is transferred to the I/O switch error_output, which is normally attached to the user's terminal.
4. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.
5. When a DISPLAY statement contains more than one operand, the operands are concatenated without intervening spaces and the values of the operands are transferred in the sequence in which the operands are encountered.
6. A newline character is transferred following the last operand.
7. No conversion of data takes place before transfer to the device. (Refer to the Multics COBOL Users' Guide.)

DIVIDE STATEMENT

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

Format 1:

```
DIVIDE { identifier-1 } INTO { identifier-2 [ ROUNDED ] } ...
       { literal-1    }
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-DIVIDE ]
```

Format 2:

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING
       { literal-1    }      { literal-2    }
    { identifier-3 [ ROUNDED ] } ...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-DIVIDE ]
```

Format 3:

```
DIVIDE { identifier-1 } BY { identifier-2 } GIVING
       { literal-1    }    { literal-2    }
    { identifier-3 [ ROUNDED ] } ...
    [ ON SIZE ERROR imperative-statement-1 ]
    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-DIVIDE ]
```

DIVIDE

Format 4:

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING
       { literal-1    }      { literal-2    }
    identifier-3 [ ROUNDED ] REMAINDER identifier-4

  [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-DIVIDE ]
```

Format 5:

```
DIVIDE { identifier-1 } BY { identifier-2 } GIVING
       { literal-1    }    { literal-2    }
    identifier-3 [ ROUNDED ] REMAINDER identifier-4

  [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-DIVIDE ]
```

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.

DIVIDE

General Rules:

1. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2, etc.. The value of the dividends (identifier-2, etc.) is replaced by the quotients.

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

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

4. Formats 4 and 5 are used when a remainder from the division operation is desired, namely 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 which contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded.

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

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

   a. If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, 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, analysis must be performed to determine which condition has actually occurred.

7. Refer to "Common Phrases in Statement Formats" earlier in this section for an explanation of the uses of the ROUNDED and SIZE ERROR phrases and arithmetic statements.

EXIT

EXIT STATEMENT

The EXIT statement provides a common end point for a series of procedures.

General Format:

EXIT.

Syntax Rules:

1. The EXIT statement must appear in a sentence by itself.
2. The EXIT sentence must be the only sentence in the paragraph.

General Rules:

1. An EXIT statement serves only to assign a procedure-name to a given point in a program. It has no other effect on the compilation or execution of the program.

GO TO STATEMENT

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

Format 1:

```
GO TO [ procedure-name-1 ]
```

Format 2:

```
GO TO { procedure-name-1 } ... DEPENDING ON identifier
```

Syntax Rules:

1. A Format 1 GO TO statement, without procedure-name-1, can only appear in a single statement paragraph.
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. 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.
4. Identifier is the name of a numeric elementary item described with no positions to the right of the assumed decimal point.

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. If no such ALTER statement is executed, abnormal termination of the object program will result.
3. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1, etc., depending on the value of the identifier being 1, 2, etc. If the value of the identifier is anything other than the positive or unsigned integers 1, 2, etc., then no transfer occurs and control passes to the next statement in the normal sequence for execution.

IF

IF STATEMENT

The IF statement causes a condition to be evaluated. The subsequent action 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 [ END-IF ] }
             { NEXT SENTENCE }  { ELSE NEXT SENTENCE          }
                                { END-IF                      }
```

Syntax Rules:

1. Statement-1 and statement-2 represent either an imperative statement or a conditional statement, optionally preceded by an imperative statement.

2. The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the final period of the sentence.

3. If the END-IF phrase is specified, the NEXT SENTENCE phrase must not be specified.

General Rules:

1. The scope of an IF statement may be terminated by any of the following:

   a. An END-IF phrase at the same level of nesting.

   b. A separator period.

   c. If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.

2. When an IF statement is executed, the following transfers of control occur:

   a. If the condition is true and statement-1 is specified, control is transferred to the first statement of statement-1 and execution continues according to the rules for each statement specified in statement-1. If a procedure branching or conditional statement

IF

is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-1, the ELSE phrase, if specified, is ignored, and control passes to the end of the IF statement.

b. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.

c. If the condition is false and statement-2 is specified, statement-1 or its surrogate NEXT SENTENCE is ignored, control is transferred to the first statement of statement-2, and execution continues according to the rules for each statement specified in statement-2. If a procedure branching or conditional statement is executed which causes an explicit transfer of control, control is explicitly transferred in accordance with the rules of that statement. Upon completion of the execution of statement-2, control passes to the end of the IF statement.

d. If the condition is false and the ELSE phrase is not specified, statement-1 is ignored and control passes to the end of the IF statement.

e. If the condition is false and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

3. Statement-1 or statement-2 may contain an IF statement. In this case, the IF statement is considered to be nested.

   IF statements within IF statements can be considered as paired IF, ELSE, and END-IF combinations, proceeding from left to right. Thus, any ELSE or END-IF encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE or END-IF.

4. Refer to "Conditional Expressions" earlier in this section for a discussion of conditions.

INSPECT STATEMENT

The INSPECT statement provides the ability to tally (Format 1), replace (Format 2), or tally and replace (Format 3) occurrences of single characters or groups of characters in a data item.

Format 1:

```
INSPECT identifier-1 TALLYING
    [ identifier-2 FOR [ [ { ALL       } { identifier-3 } ]
                         [ { LEADING   } { literal-1   } ]
                         [ { CHARACTERS }               ]
      [ { BEFORE } INITIAL { identifier-4 } ] ] ... ] ...
      { AFTER  }         { literal-2    }
```

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 } ]
      [ { LEADING   } { literal-3   }    { literal-4   } ]
      [ { FIRST     }                                   ]
      [ { BEFORE } INITIAL { identifier-7 } ] ] ... ] ...
      { AFTER  }         { literal-5    }
```

Format 3:

```
INSPECT identifier-1 TALLYING

{ identifier-2 FOR { { { ALL       } { identifier-3 } } }
                   { { { LEADING   } { literal-1   } } }
                   { { { CHARACTERS }               } }

    [ { BEFORE } INITIAL { identifier-4 } ] } ... } ...
    [ { AFTER  }         { literal-2    } ] }

  REPLACING

{ CHARACTERS BY { identifier-6 } [ { BEFORE } INITIAL { identifier-7 } ]
                { literal-4    } [ { AFTER  }         { literal-5    } ]

  { { ALL     } { { identifier-5 } BY { identifier-6 } }
  { { LEADING } { { literal-3    }    { literal-4    } }
  { { FIRST   }

      [ { BEFORE } INITIAL { identifier-7 } ] } ... } ... }
      [ { AFTER  }         { literal-5    } ]
```

Syntax Rules:

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 etc. must reference either an elementary alphabetic, alphanumeric, or numeric item described (either implicitly or explicitly) as USAGE IS DISPLAY.

3. Each literal must be nonnumeric and can be any figurative constant, except ALL.

INSPECT

Formats 1 and 3 Only:

1. Identifier-2 must reference an elementary numeric data item.

2. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one-character data item.

Formats 2 and 3 Only:

1. 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.

2. 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.

3. 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:

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanical 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 identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

   b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are described as alphanumeric edited, numeric edited, or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (see 2a above) and the INSPECT statement had been written to reference the redefined data item.

   c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are 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 2b above had been applied. (Refer to the MOVE statement.)

3. In general rules 4 through 6, and the specific rules for Formats 1, 2, and 3 which follow, 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 the 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 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 the specific rules for Formats 1 and 2. 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 one-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.

INSPECT

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 certain comparison cycles, namely those which involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3, or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

   c. If the AFTER phrase is specified, the associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase can participate only in certain comparison cycles, namely those which involve that portion of the contents of the data item referenced by identifier-1 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:

1. The contents of the data item referenced by identifier-2 are not initialized by the execution of the INSPECT statement.

2. The rules for tallying are as follows:

   a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

   b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided 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 are incremented by one for each character matched, in the sense of general rule 5e, within the contents of the data item referenced by identifier-1.

   d. Overlapping operands are not permitted.

Format 2:

1. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

2. 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 the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 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:

1. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 are written. One of these statements is a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the other statement is a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and tallying apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement.

Examples of INSPECT Statement:

Following are six examples of the INSPECT statement:

Example 1:

```
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
```

Example 2:

```
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
```

Example 3:

```
INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X"

Where word = ARXAX, word = GRXAX
Where word = HANDAX, word = HGNDGX
```

Example 4:

```
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
```

Example 5:

```
INSPECT word REPLACING ALL "X" BY "Y", "B, "B" BY "Z", "W" BY "Q" AFTER
INITIAL "R"

Where word = RXXBQWY, word = RYYZQQY
Where word = YZACDWBR, word = YZACDWBR
Where word = RAWRXEB, word = RAQRYEZ
```

INSPECT

Example 6:

```
INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A"

word before     1  2     X  Z  A  B  C  D
word after      BBBBBBBBBBBBBBA  B  C  D
```

MOVE

MOVE STATEMENT

The MOVE statement transfers data, in accordance with the rules for editing, to one or more data areas.

Format 1:

```
MOVE { identifier-1 } TO identifier-2 ...
     { literal      }
```

Format 2:

```
MOVE { CORR          } identifier-1 TO identifier-2
     { CORRESPONDING }
```

Syntax Rules:

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3, etc. 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 specified, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given for the CORRESPONDING phrase in this section. The results are the same as if the user refers 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, etc. The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, etc. 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. The result of the statement

```
MOVE  a (b) TO  b, c  (b)
```

is equivalent to:

```
MOVE  a (b) TO  temporary operand
MOVE  temporary operand  TO  b
MOVE  temporary operand  TO  c  (b)
```

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 category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

   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 noninteger numeric literal or a noninteger 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 procedures given in general rule 4 below.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with 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" in Section 2. 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 is not moved; if the operational sign occupied a separate character position (refer to the SIGN clause), that character is not 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 "Standard Alignment Rules," except where zeros are replaced because of editing requirements.

- When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. 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.
- 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.
- 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 "Standard Alignment Rules." If the size of the sending item is greater than the size of the receiving item, 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 is filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in general rule 4 of the OCCURS clause.

6. Table 7-3 summarizes the validity of the various types of MOVE statements. The reference to a general rule indicates the rule that either prohibits the move or describes the actions of a valid move.

Table 7-3. Valid Types of MOVE Statements

Category of Sending Data Item	Category of Receiving Data Item		
	Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Numeric Noninteger Numeric Edited
Alphabetic	Valid/4c	Valid/4a	Invalid/3a
Alphanumeric	Valid/4c	Valid/4a	Valid/4b
Alphanumeric Edited	Valid/4c	Valid/4a	Invalid/3a
Numeric { Integer	Invalid/3b	Valid/4a	Valid/4b
Numeric { Noninteger	Invalid/3b	Invalid/3c	Valid/4b
Numeric Edited	Invalid/3b	Valid/4a	Invalid/3a

MULTIPLY

MULTIPLY STATEMENT

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

Format 1:

```
MULTIPLY { identifier-1 } BY { identifier-2 [ ROUNDED ] } ...
         { literal-1    }

    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-MULTIPLY ]
```

Format 2:

```
MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }    { literal-2    }
GIVING { identifier-3 [ ROUNDED ] } ...
    [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-MULTIPLY ]
```

Syntax Rules:

1. Each identifier must refer to a numeric elementary item, except that in Format 2 each 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 18 digits.

MULTIPLY

General Rules:

1. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2, etc. The values of the multipliers (identifier-2, etc.) are replaced by the products.

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

3. Refer to "Common Phrases in Statement Formats" earlier in this section for an explanation of the uses of the ROUNDED and SIZE ERROR phrases and arithmetic statements.

PERFORM STATEMENT

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.

Format 1:

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

        [ imperative-statement END-PERFORM ]
```

Format 2:

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

        { identifier-1 } TIMES [ imperative-statement END-PERFORM ]
        { integer-1    }
```

Format 3:

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

            UNTIL condition-1 [ imperative-statement END-PERFORM ]
```

Format 4:

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

        VARYING { identifier-2 } FROM { identifier-3 }
                { index-name-1 }      { index-name-2 }
                                      { literal-1    }

        BY      { identifier-4 } UNTIL condition-1
                { literal-2    }
```

PERFORM

```
[ 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     }
```

```
[ imperative-statement  END-PERFORM ]
```

Syntax Rules:

1. If procedure-name-1 is omitted, imperative-statement-1 and the END-PERFORM phrase must be specified. If procedure-name-1 is specified, imperative-statement-1 and the END-PERFORM phrase must not be specified.

2. In format 4, if procedure-name-1 is omitted, the AFTER phrase must not be specified.

3. Each identifier represents a numeric elementary item described in the Data Division. In Format 2, identifier-1 must be described as a numeric integer.

4. Each literal represents a numeric literal.

5. The words THRU and THROUGH are equivalent.

6. 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 nonzero integer.

7. 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.

8. A literal in the BY phrase must not be zero.

9. Condition-1, condition-2, condition-3 can be any conditional expression as described under "Conditional Expressions" earlier in this section.

10. 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, both must be procedure-names in the same declarative section.

General Rules:

1. 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 procedure-name-1 is specified, the PERFORM statement is referred to as an "out-of-line" PERFORM statement; when procedure-name-1 is omitted, the PERFORM statement is referred to as an "in-line" PERFORM statement.

4. The statements contained within the range of procedure-1 (through procedure-name-2 if specified) for an out-of-line PERFORM statement, or contained within the PERFORM statement itself for an in-line PERFORM statement, are referred to as the "specified set of statements".

5. The END-PERFORM phrase delimits the scope of the in-line PERFORM statement.

6. An in-line PERFORM statement functions according to the following General Rules for an otherwise identical out-of-line PERFORM statement. The exception is that the statements contained within the in-line PERFORM statement are executed in place of the statements contained within the range of procedure-name-1 (through procedure-name-2 if specified). Unless specifically qualified by the word "in-line" or "out-of-line", all the General rules which apply to the out-of-line PERFORM statement also apply to the in-line PERFORM statement.

7. 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 10b, 10c, and 10d). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

PERFORM

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.

e. If an in-line PERFORM statement is specified, an execution of the PERFORM statement is completed after the last statement contained within it has been executed.

8. There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements can occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 can be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

9. If control passes to these procedures by means 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.

10. PERFORM statements operate as follows:

a. Format 1 is the basic PERFORM statement. The specified set of statements referenced by this type of PERFORM statement is executed once and then control passes to the end of the PERFORM statement.

b. Format 2 is the PERFORM...TIMES. The specified set of statements is performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the end of the PERFORM statement. Following the execution of the specified set of statements the specified number of times, control is transferred to the end of the PERFORM statement.

During execution of the PERFORM statement, references to identifier-1 cannot alter the number of times the specified set of statements is to be executed from that indicated by the initial value of identifier-1.

c. Format 3 is the PERFORM...UNTIL. The specified set of statements is performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the end of the PERFORM statement. If the condition is true when the PERFORM statement is entered, no transfer to the specified set of

PERFORM

statements takes place, and control is passed to the end of 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 or AFTER phrase, is initialized according to the rules of the SET statement; subsequent augmentation is as described in Figures 7-1, 7-2, and 7-3.

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 specified set of statements 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 end of the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the end of the PERFORM statement.

```
Set identifier-2 to current FROM value

PERFORM WHILE (condition-1)

   Execute specified set of statements

   Augment identifier-2 with current BY value

END PERFORM
```

Figure 7-1. Equivalent Program 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 end of the PERFORM statement; if false, condition-2 is evaluated. If condition-2 is false, the specified set of statements is 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 changes 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) are taken into consideration and affect the operation of the PERFORM statement.

```
Set identifier-2 to current FROM value
Set identifier-5 to current FROM value

PERFORM WHILE (condition-1)

   PERFORM WHILE (condition-2)

      Execute specified set of statements

      Augment identifier-5 with current BY value

   END-PERFORM

   Set identifier-5 to current FROM value
   Augment identifier-2 with current BY value

END-PERFORM
```

Figure 7-2. Equivalent Program 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, in which case 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.

PERFORM

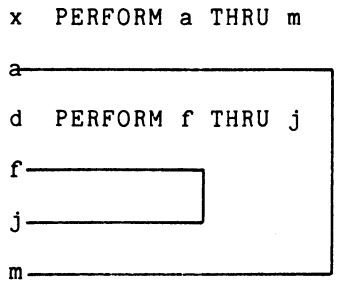
```
Set identifier-2 to current FROM value
Set identifier-5 to current FROM value
Set identifier-8 to current FROM value

PERFORM while (condition-1)

   PERFORM WHILE (condition-2)

      PERFORM WHILE (condition-3)

         Execute specified set of statements

         Augment identifier-8 with current BY value

      END-PERFORM

      Set identifier-8 to current FROM value
      Augment identifier-5 with current BY value

END-PERFORM

      Set identifier-5 to current FROM value
      Augment identifier-2 with current BY values

END-PERFORM
```

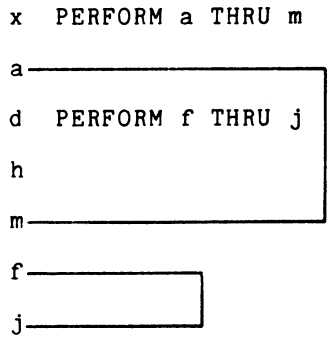
Figure 7-3. Equivalent Program for the VARYING Phrase of a PERFORM Statement Having Three Conditions

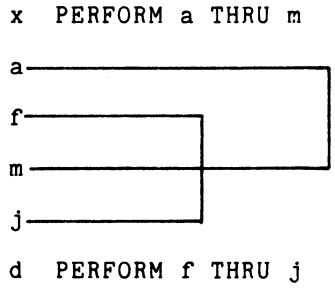
After the completion of a Format 4 PERFORM statement, identifier-5 and identifier-8 contain the current value of identifier-6 and identifier-9, respectively. Identifier-2 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-2 contains the current value of identifier-3.

PERFORM

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 following illustrations.







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 nonindependent 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 nonindependent segments

   b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement

SET STATEMENT

The SET statement allows the user to modify the value of external switches.

General Format:

```
SET { { mnemonic-name-1 } ... TO { ON  } } ...
                                 { OFF }
```

Syntax Rules:

1. Each mnemonic-name must be associated with an external-switch, the status of which can be altered by the SWITCH-n phrase in the SPECIAL-NAMES paragraph.

General Rules:

1. The status of each external-switch associated with the specified mnemonic-name is modified such that the truth value resultant from the evaluation of a condition-name associated with that switch will reflect an ON status if the ON phrase is specified, or an OFF status if the OFF phrase is specified. (Refer to "Switch-Status Condition" in this section.)

STOP STATEMENT

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

General Format:

```
STOP { RUN     }
     { literal }
```

Syntax Rules:

1. The literal can be numeric or nonnumeric or can be any figurative constant, except ALL.
2. 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 object program is brought to normal termination.
2. If the STOP literal is specified, the literal and the program-name are communicated to the user's terminal, and a new level of the command processor is established. The user can resume execution of the program at the next executable statement by using the Multics start command.

3. Refer to the the Multics COBOL Users' Guide for additional information on the STOP statement.

STRING STATEMENT

The STRING statement provides juxtaposition of the partial or complete contents of two or more data items into a single data item.

General Format:

```
STRING { { identifier-1 } ... DELIMITED BY { identifier-2 } } ...
       { { literal-1    }                  { literal-2    }
                                           { SIZE         }

    INTO identifier-3 [ WITH POINTER identifier-4 ]

  [ ON OVERFLOW imperative-statement-1 ]

  [ NOT ON OVERFLOW imperative-statement-2 ] [ END-STRING ]
```

Syntax Rules:

1. Each literal can be any figurative constant without the optional word ALL.
2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-4, must be described explicitly or implicitly in the USAGE clause as DISPLAY.
3. Identifier-3 must represent an elementary alphanumeric data item without editing symbols.
4. Identifier-4 must represent an elementary numeric integer data item of sufficient size to contain a value equal to one greater than the size of the area referenced by identifier-3. The symbol P may not be used in the PICTURE character-string of identifier-4.
5. When identifier-1, or identifier-2 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 in this section to identifier-1, identifier-2, literal-1, and literal-2 apply equally to all recursions thereof.
2. Identifier-1 or literal-1 represent the sending items. Identifier-3 represents the receiving item.

3. Literal-2, identifier-2 indicate the characters delimiting the move. If the SIZE phrase is used, the complete data item defined by identifier-1 or literal-1 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 or literal-2, it refers to an implicit one-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 or from the contents of the data item referenced by identifier-1 are transferred to the contents of identifier-3 in accordance with the rules for alphanumeric-to-alphanumeric moves, except that no space-filling is provided. (Refer to "MOVE Statement.")

   b. If the DELIMITED BY phrase is specified without the SIZE phrase, the contents of the data item referenced by identifier-1 or the value of literal-1 is 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 characters specified by literal-2 or by the contents of identifier-2 are encountered. The characters specified by literal-2 or by the data item referenced by identifier-2 are not transferred.

   c. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1 or the contents of the data item referenced by identifier-1 is transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-3 until all data is transferred or the end of the data item referenced by identifier-3 is reached.

6. If the POINTER phrase is specified, identifier-4 is explicitly available to the user, who is responsible for setting its initial value. The initial value must be greater than zero.

7. If the POINTER phrase is not specified, the following general rules apply as if the user specified identifier-4 with an initial value of one.

8. When characters are transferred to the data item referenced by identifier-3, the characters are moved one at a time from the source into the character position of the data item referenced by identifier-3 designated by the value associated with Identifier-4. Identifier-4 is then increased by one prior to the move of the next character. The value associated with identifier-4 is changed during execution of the STRING statement only by the activity specified above.

9. At the end of the execution of the STRING statement, only the portion of the data item referenced by identifier-3 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-3 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 STRING statement is completed, the value associated with identifier-4 is either less than 1 or exceeds the number of character positions in the data item referenced by identifier-3, no further data is transferred to the data item referenced by identifier-3, and the imperative statement in the ON OVERFLOW phrase is executed, if specified.

STRING

11. If the ON OVERFLOW phrase is not specified when the conditions described in general rule 10 above are encountered, control is transferred to the next executable statement.

SUBTRACT STATEMENT

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 values of one or more items equal to the results.

Format 1:

```
SUBTRACT { literal-1    } ...
         { identifier-1 }
           FROM {identifier-m [ ROUNDED ]} ...
      [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-SUBTRACT ]
```

Format 2:

```
SUBTRACT { literal-1    } ... FROM { literal-2    }
         { identifier-1 }          { identifier-2 }
           GIVING {identifier-3 [ ROUNDED ]} ...
      [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-SUBTRACT ]
```

Format 3:

```
SUBTRACT { CORR          } identifier-1 FROM identifier-2 [ ROUNDED ]
         { CORRESPONDING }
      [ ON SIZE ERROR imperative-statement-1 ]

    [ NOT ON SIZE ERROR imperative-statement-2 ] [ END-SUBTRACT ]
```

Syntax Rules:

1. Each identifier must refer to a numeric elementary item except that:
   - In Format 2, each identifier 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 decimal digits.
   - In Format 1, the composite of operands is determined by using all of the operands in a given statement.
   - 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.
   - In Format 3, the composite of operands is determined separately for each pair of corresponding data items.
4. CORR is an abbreviation for CORRESPONDING.

General Rules:

1. In Format 1, all literals or identifiers preceding the word FROM are added together, this total is subtracted from the current value of identifier-m, and the result is stored immediately into identifier-m. This process is repeated, respectively, for each operand following the word FROM.
2. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-2 or identifier-2, and the result of the subtraction is stored as the new value of identifier-3, etc.
3. If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.
4. For large operands, the compiler uses the temporary operand described under "Arithmetic Statements" earlier in this section.
5. Refer to "Common Phrases in Statement Formats" earlier in this section for uses of the ROUNDED, SIZE ERROR, and CORRESPONDING phrases and multiple results.

UNSTRING

UNSTRING STATEMENT

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 ] } ...
[ WITH POINTER identifier-7 ] [ TALLYING IN identifier-8 ]

  [ ON OVERFLOW imperative-statement-1 ]

  [ NOT ON OVERFLOW imperative-statement-2 ] [ END-UNSTRING ]
```

Syntax Rules:

1. Each literal must be a nonnumeric literal. In addition, each literal can be any figurative constant without the optional word ALL.
2. Identifier-1, identifier-2, identifier-3, and identifier-5 must each be described, explicitly or implicitly, as an alphanumeric data item.
3. Identifier-4 can 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 cannot be used in the PICTURE character-string), and must be described in the USAGE clause as DISPLAY.
4. Identifier-6, identifier-7, and identifier-8 must be described as elementary numeric integer data items (except that the symbol P cannot be used in the PICTURE character-string).
5. An identifier must not name a level-number 88 entry.
6. The DELIMITER IN phrase and the COUNT IN phrase can be specified only if the DELIMITED BY phrase is specified.

General Rules:

1. All references in this section to identifier-2, literal-1, identifier-3, literal-2, and identifier-6 apply equally to 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 isolated by the delimiters for the move to identifier-4. This value does not include a count of the delimiter characters.

6. The data item referenced by identifier-7 contains a value that indicates a relative character position within the area defined by identifier-1.

7. The data item referenced by identifier-8 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 single-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 is only one occurrence, and this occurrence is moved to the receiving data item according to general rule 13d.

9. When any examination encounters two contiguous delimiters, the current receiving area is either space-filled or zero-filled according to the description of the receiving area, except when Rule 8 (the ALL phrase) applies.

10. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the computers 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 characters in the sending field are considered to be a single delimiter. No characters 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.

UNSTRING

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 contents of the data item referenced by identifier-7. 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. (Refer to 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 satisfied, the examination terminates with the last character examined.

    c. The characters thus examined (excluding the delimiting characters, 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.

    d. If the DELIMITER IN phrase is specified, the delimiting characters 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. 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 characters, 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 the next recurrence of identifier-4. The activity described in paragraphs 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.

UNSTRING

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-7 is incremented by 1 for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the contents of the data item referenced by identifier-7 contains 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-8 contains 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 statement is initiated, and the value in the data item referenced by identifier-7 is less than 1 or greater than the size of the data item referenced by identifier-1.

    b. If, during the execution of an UNSTRING statement, all data 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-7, identifier-8 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.

SECTION 8

TABLE HANDLING

DESCRIPTION OF TABLE HANDLING

The table handling feature provides a capability for defining tables of contiguous data items and for accessing an item relative to its position in the table.

Data items can be accessed in up to three-dimensional variable-length tables. Additional facilities are provided for specifying ascending or descending keys and for searching a dimension of a table for an item that satisfies a specific condition.

Tables composed of contiguous data items are defined in COBOL by including the OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table element, and its name and description apply to each repetition or occurrence. Since each occurrence of a does not have a unique data-name assigned to it, reference to a desired occurrence can be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. Subscripting and indexing are the two methods used to specify the occurrence number of a desired table element.

Table Definition

To define a one-dimensional table, an OCCURS clause is used as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items that contain the table element. Example 1 shows a one-dimensional table.

Example 1:

```
01  TABLE-1.
    02  TABLE-ELEMENT  OCCURS  20  TIMES.
        03  NAME  PIC X (15)
        03  ADDRESS  PIC X (25)
```

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table produces a two-dimensional table. To define a two-dimensional table, therefore, an OCCURS clause must appear in the data description of the element of the table and in the description of only one group item containing that table element. To define a three-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the description of two group items which contain the element. In COBOL, tables of up to three dimensions are permitted. Example 2 shows a table that has one dimension for CONTINENT-NAME, two dimensions for COUNTRY-NAME, and three dimensions for CITY-NAME and CITY-POPULATION. The table includes 100,510 data items: 10 for CONTINENT-NAME, 500 for COUNTRY-NAME, 50,000 for CITY-NAME, and 50,000 for CITY-POPULATION. Within the table there are 10 occurrences of CONTINENT-NAME. Within each CONTINENT-NAME there are 50 occurrences of COUNTRY-NAME, and within each COUNTRY-NAME there are 100 occurrences of CITY-NAME and CITY-POPULATION.

Example 2:

```
01  CENSUS-TABLE.
    05  CONTINENT-TABLE  OCCURS  10  TIMES.
        10  CONTINENT-NAME  PIC  XXXXXX.
        10  COUNTRY-TABLE  OCCURS  50  TIMES.
            15  COUNTRY-NAME  PIC  XXXXXXXX.
            15  CITY-TABLE  OCCURS  100  TIMES.
                20  CITY-NAME  PIC  XXXXXXXXXX.
                20  CITY-POPULATION  PIC 999999999999.
```

References to Table Items

Whenever the user refers to a table element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table, the occurrence number of the desired element provides complete information. For access to tables of more than one dimension, an occurrence number must be supplied for each dimension of the table accessed. In Example 2 then, a reference to the fourth CONTINENT-NAME would be complete, whereas a reference to the fourth COUNTRY-NAME would not. To refer to COUNTRY-NAME, which is an element of a two-dimensional table, the user must refer to the fourth COUNTRY-NAME within the sixth CONTINENT-TABLE.

One method by which occurrence numbers can be specified is to append one or more subscripts to the data-name. A subscript is an integer whose value specifies the occurrence number of an element. The subscript can be represented either by a literal that is an integer or by a data-name that is defined elsewhere as a numeric elementary item with no character positions to the right of the assumed decimal point. In either case, the subscript, enclosed in parentheses, is written immediately following the name of the table element. A table reference must include as many subscripts as there are dimensions in the table whose element is being referenced; i.e., there must be a subscript for each OCCURS clause in the hierarchy containing the data-name, including the data-name itself. In Example 2, references to CONTINENT-NAME require only one subscript, references to COUNTRY-NAME require two, and references to CITY-NAME and CITY-POPULATION require three.

When more than one subscript is required, they are written in order of successively less inclusive dimensions of the data organization. When a data-name is used as a subscript, it can be used to refer to items in many different tables. These tables need not have elements of the same size. The data-name can also appear as the only subscript with one item and as one of two or three subscripts with another item. Also, it is permissible to mix literal and data-name subscripts; e.g., CITY-POPULATION(10, NEWKEY, 42). Subscripts may be written as follows:

(1Δ2)
(Δ1Δ2) (1Δ2Δ)
(1,Δ2)
(Δ1, Δ2Δ)
(1,Δ2Δ)
(Δ1, Δ2)

NOTE: A comma used as a separator must be followed by a space.

Another method of referring to items in a table is indexing. To use this technique, one or more index-names are assigned to an item whose data description contains an OCCURS clause. There is no separate entry to describe the index-name, since its definition is completely hardware-oriented, and it is not considered data per se. At object program execution, the contents of the index-name correspond to an occurrence number for that specific dimension of the table to which the index-name is assigned. The initial value of an index-name at object program execution is not determinable, and the index-name must be initialized by the SET statement before use.

When a reference is made to a table element or to an item within a table element, and the name of the item is followed by its related index-name or names in parentheses, then each occurrence number required to complete the reference is obtained from the respective index-name. The index-name thus acts as a subscript whose value is used in any table reference that specifies indexing.

When a reference requires more than one occurrence number for completeness, a data-name subscript must not be used to indicate one occurrence number and an index-name to indicate another. Therefore, if indexing is to be used, each OCCURS clause within the hierarchy (each dimension of the table) must contain an INDEXED BY phrase. However, literals and index-names can be mixed within one reference, just as literals and data-name subscripts are mixed.

Table Searching

Data that has been arranged in the form of a table is often searched. In COBOL, the SEARCH statement, through its two options, provides facilities for producing serial and nonserial, e.g., binary, searches. When the SEARCH statement is used, an associated index-name or data-name can be varied. This statement also provides for the execution of imperative statements when certain conditions are true.

DATA DIVISION FOR TABLE HANDLING

OCCURS Clause

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

Format 1:

```
OCCURS integer-2 TIMES

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

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

Format 2:

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

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

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

Syntax Rules:

1. When both integer-1 and integer-2 are used (Format 2), the value of integer-1 must be less than the value of integer-2.

NOTE: Integer-1 can be zero.

2. Data-name-1 must describe a positive integer.
3. Data-name-1 and data-name-2 can be qualified.

OCCURS

4. The first recurrence of data-name-2 must be either the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause. Subsequent recurrences of data-name-2 must be subordinate to the entry containing the OCCURS clause.

5. Data-name-2 must be specified without the subscripting normally required.

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 since 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 can be followed, within that record description, only by data description entries that are subordinate to it.

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

   a. Has a 01, 66, 77, or 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 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 be no 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, etc. 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 this clause is used, the data-name that is the subject of this data description entry must be either subscripted or indexed, whenever it is referred to in a statement other than SEARCH. In addition, if the subject of this entry is the name of a group item, all data-names belonging to the group 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 entry clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described. (Refer to "VALUE Clause" in Section 6.)

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.

   Format 2 specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences, 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 referenced group item has an entry subordinate to it that specifies Format 2 of the OCCURS clause, 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, etc. The ascending or descending order is determined according to the rules for comparison of operands. (Refer to Comparison of Numeric and Nonnumeric Operands under "Relation Conditions" in Section 7.) The data-names are listed in their descending order of significance, from most significant to least significant.

USAGE Clause

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

General Format:

```
[ USAGE  IS ] INDEX
```

Syntax Rules:

1. 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.

2. The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SYNCHRONIZED, and VALUE 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. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value that must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. If a group item is described with 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 statements or in a relation condition.

3. 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.

4. The USAGE IS INDEX clause indicates that the data item contains two values that point to a particular element within a table. The two values are the occurrence index (occurrence number times the size of each occurrence) and the occurrence number. Index data items are represented by six bytes (nine-bit bytes with bit C of each byte unused). The first four bytes contain, in binary form, the occurrence index, and the last two bytes contain, in binary form, the occurrence number.

5. Refer to "Data Allocation Rules" in Section 2.

Relation Condition

Relation tests can be made between:

1. Two index-names. The result is the same as if the corresponding occurrence numbers were compared.
2. An index-name and a data item (other than 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-name or another index data item. The actual values are compared without conversion.

The result of the comparison of an index data item with any data item or literal not specified above will result in a compile time fatal diagnostic.

Overlapping Operands

When a sending and a receiving item in a SET statement share a part of their storage areas, the result of the execution of such a statement is undefined.

SEARCH Statement

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.

Format 1:

```
SEARCH identifier-1 [ VARYING { identifier-2 } ]
                              { index-name-1 }
   [ AT END imperative-statement-1 ]
     { WHEN condition-1 { imperative-statement-2 } } ...
                        { NEXT SENTENCE          }
   [ END-SEARCH ]
```

Format 2:

```
SEARCH ALL identifier-1 [ AT END imperative-statement-1 ]
        WHEN { data-name-1 { IS EQUAL TO } { identifier-3 }   }
             {             { EQUALS     } { literal-1    }   }
             {             { IS =       } { arithmetic-expression-1 } }
             { condition-name-1                               }
     [ AND { data-name-2 { IS EQUAL TO } { identifier-4 }   } ] ...
           {             { EQUALS     } { literal-2    }   }
           {             { IS =       } { arithmetic-expression-2 } }
           { condition-name-2                               }
       { imperative-statement-2 }
       { NEXT SENTENCE          }
   [ END-SEARCH ]
```

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 phrase. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

2. When specified, identifier-2 must be described as USAGE IS INDEX or as a numeric elementary item with no positions to the right of the assumed decimal point.

3. In Format 1, condition-1, etc., may be any condition as described under "Conditional Expressions" in Section 7.

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 phrase of identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2 must be indexed by the first index-name associated with identifier-1 along with other indexes or literals as required, and must be referenced in the KEY phrase of identifier-1. Identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2 must not be referenced in the KEY phrase 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 phrase of identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY phrase of identifier-1 is referenced, all preceding data-names in the KEY phrase of identifier-1 or their associated condition-names must also be referenced.

5. If the END-SEARCH phrase is specified, the NEXT SENTENCE phrase must not be specified.

SEARCH

General Rules:

1. If Format 1 of the SEARCH statement is used, a serial type of search operation takes place, starting with the current index setting.

   a. At the start of execution of the SEARCH statement, if 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. 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. At the start of execution of the SEARCH statement, if 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 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 which case the search terminates as indicated in a. 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 Format 2, the results of the SEARCH ALL operation are predictable only when:

   a. The data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY phrase associated with the description of identifier-1, and

   b. The contents of the keys referenced in the WHEN phrase are sufficient to identify a unique table element.

3. If Format 2 of SEARCH is used, the initial setting of the index-name for identifier-1 is ignored, and a binary search operation takes place. If any conditions specified in the WHEN phrase 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 conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to imperative-statement-2.

SEARCH

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 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 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 two- or three-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 two- or three-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.

9. A flowchart of the Format 1 SEARCH operation containing two WHEN phrases appears in Figure 8-1.

SEARCH

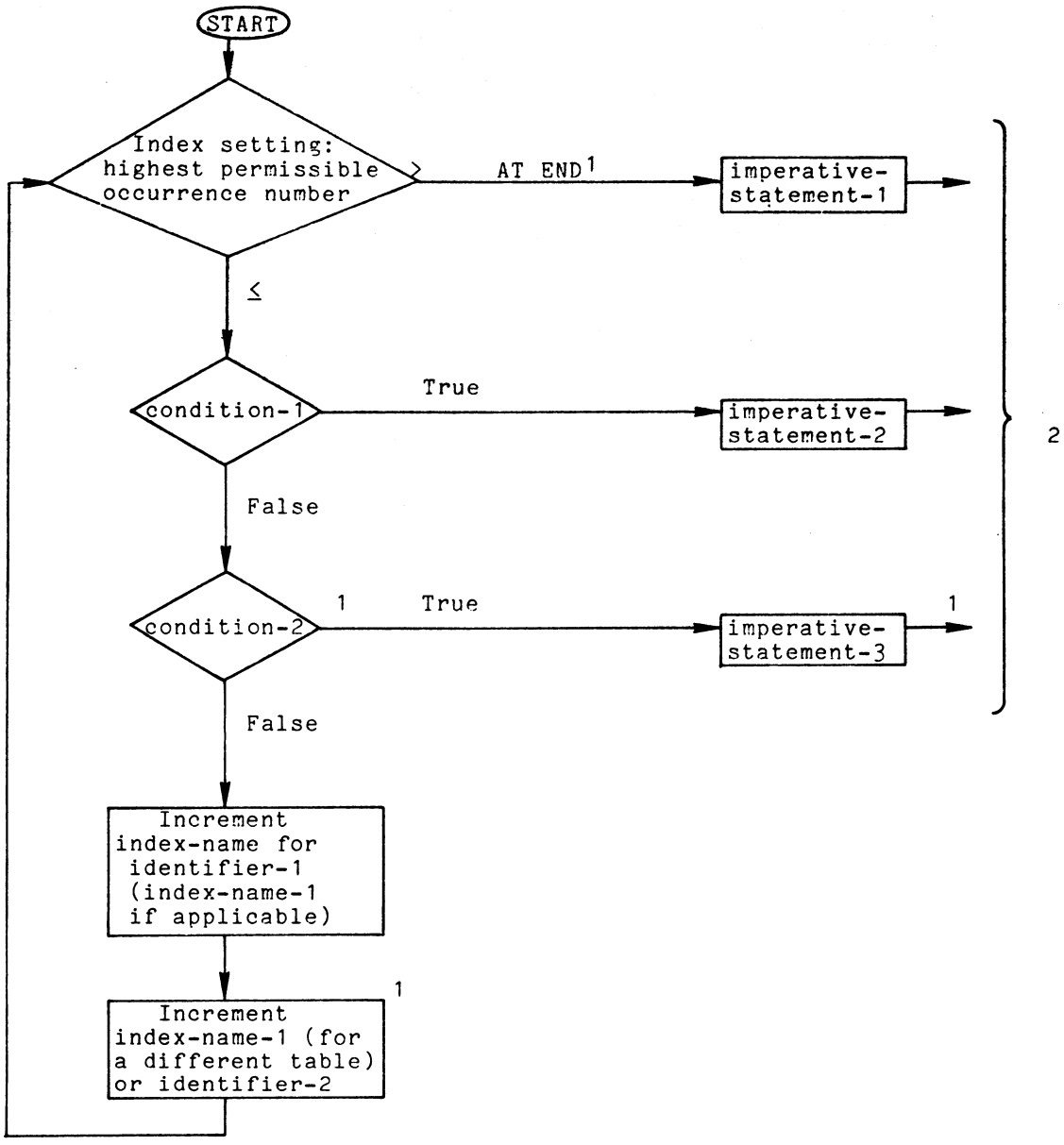


Figure 8-1. Flowchart of Format 1 SEARCH Operation

---

[1]These operations are options included only when specified in the SEARCH statement.

[2]Each of these control transfers is to the next executable sentence unless the imperative statement ends with a GO TO statement.

SET Statement

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

Format 1:

```
SET { index-name-1  ... } TO { index-name-2 }
    { identifier-1  ... }    { identifier-2 }
                             { integer-1    }
```

Format 2:

```
SET index-name-3 ... { UP BY   } { identifier-3 }
                     { DOWN BY } { integer-2    }
```

Format 3:

```
SET { mnemonic-name-1 } ... TO { ON  }
    { SWITCH-n        }        { OFF }
```

Syntax Rules:

1. All references in this section to index-name-1, identifier-1, and index-name-3 apply equally to all recursions thereof.
2. Identifier-1 and identifier-2 must name either index data items or elementary items described as an integer.
3. Identifier-3 must be described as an elementary numeric integer.
4. Integer-1 and integer-2 may be signed. Integer-1 must be positive.
5. When SWITCH-n is specified, n must be an unsigned integer ranging from 1 to 8.
6. Each mnemonic-name must be associated with an external switch (SWITCH-n), the status of which may be altered (see the SPECIAL NAMES paragraph).

General Rules:

Formats 1 and 2:

1. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY phrase.
2. If index-name-2 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.

SET

If index-name-3, etc. 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, etc. 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. The value of the index associated with an index-name after the execution of a SEARCH or PERFORM statement may be undefined.

3. In Format 1, the following action occurs:

   a. Index-name-1, etc. 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-2, identifier-2, or integer-1. If identifier-2 is an index data item, or if index-name-2 is related to the same table as index-name-1, no conversion takes place.

   b. If identifier-1, etc. is an index data item, it may be set equal to either the contents of index-name-2 or identifier-2, where identifier-2 is also an index data item; no conversion takes place in either case.

   c. If identifier-1, etc. is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-2. Neither identifier-2 nor integer-1 can be used in this case.

   d. The process is repeated for each recurrence of index-name-1 or identifier-1. Each time, the value of index-name-2 or identifier-2 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

4. In Format 2, the contents of index-name-3 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-3; thereafter, the process is repeated for each recurrence of index-name-3. For each repetition, the value of identifier-3 is used as it was at the beginning of the execution of the statement.

5. Data in Table 8-1 represents the validity of various operand combinations in the SET statement. Each numeric reference indicates the applicable foregoing general rule.

Format 3:

6. The status of each external switch explicitly referenced by its name SWITCH-n or associated with the specified mnemonic-name is modified so that the truth value resulting from the evaluation of a condition name associated with the switch will reflect an ON status if the ON phrase is specified, or an OFF status if the OFF phrase is specified.

SET

Table 8-1. Operand Combinations in SET Statement

Sending Item	Receiving Item		
	Integer Data Item	Index-Name	Index Data Item
Integer Literal	Invalid/3c	Valid/3a	Invalid/3b
Integer Data Item	Invalid/3c	Valid/3a	Invalid/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b (See Note.)
Index Data Item	Invalid/3c	Valid/3a (See Note.)	Valid/3b (See Note.)
NOTE: No conversion takes place.			

SECTION 9

FILE INPUT/OUTPUT

DESCRIPTION OF FILE INPUT/OUTPUT

File input/output provides the capability for transferring data from an external recording device into computer memory (input) or from memory to an external recording device (output) with a minimum of concern for the physical characteristics of the recording device or the processes required to complete the transfer.

Input/Output Organization

Sequential files are organized such that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except in the case where records are added to the end of the file.

> Stream organization is a form of sequential organization. A stream file is unstructured. Each record on the file is terminated by a new line character.

Relative file organization is permitted only on files in virtual memory. A relative file consists of records identified by relative record numbers. The file can be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is uniquely identified by a relative record number which is greater than zero. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through the ninth record areas.

A file whose organization is indexed is a file in the virtual memory file system in which data records can be accessed by the value of one or more keys associated with an index. The index provides a logical path to the data records according to the contents of a data item within each record, which is the record key.

The data item named in the RECORD KEY clause of the file control entry for a file is the prime record key for that file. For the purposes of inserting, updating, and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique and must not be changed when updating the record.

A data item named in the ALTERNATE RECORD KEY clause of the file control entry for a file is an alternate record key for that file. The value of an alternate record key may be nonunique if the DUPLICATES phrase is specified for it. These keys provide alternate access paths for retrieval of records from the file.

Access Modes

In the sequential access mode, records are accessed in the sequence in which the records were originally written (if organization is sequential or stream), the ascending order of the relative record numbers of all records within the file (if organization is relative), or the ascending order of the record key values (if organization is indexed). The order of retrieval of records, within a set of records having duplicate record key values, is the order in which the records were written into the set.

In the random access mode, records are accessed in the sequence controlled by the user. The desired record is accessed by placing its relative record number in a relative key data item (if organization is relative), or by placing the value of its record key in a record key data item (if organization is indexed). Random access is not allowed with sequential or stream organization.

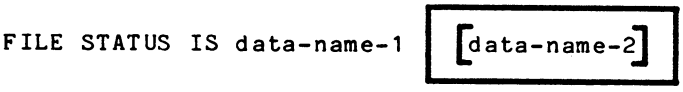
In the dynamic access mode, the user can alternate from sequential access to random access using appropriate forms of input/output statements. Dynamic access is not allowed with sequential or stream organization.

Current Record Pointer

The current record pointer is a conceptual entity used in this manual to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN, READ, and START statements.

Input/Output Status

The description of any file (in the Environment Division SELECT clause) in the source program can contain the optional clause:



If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of a CLOSE, DELETE, OPEN, READ, REWRITE, START, or WRITE statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input/output operation.

STATUS KEY 1

The leftmost character position of data-name-1 is known as status key 1 and is set to indicate one of the following conditions upon completion of the input/output operation:

'0' - Successful Completion. The input/output statement was successfully executed.

'1' - At End. The sequential READ statement was unsuccessfully executed either as a result of an attempt to read a record when no next logical record exists in the file or as a result of the first READ statement being executed for a file described with the OPTIONAL clause and that file was not available to the program at the time its associated OPEN statement was executed.

'2' - Invalid Key. The input/output statement was unsuccessfully executed because of one of the following conditions:

- Sequence error
- Duplicate key
- No record found
- Boundary violation

'3' - Permanent Error. The input/output statement was unsuccessfully executed because of conditions such as:

- File does not exist
- Read to an unopened file

STATUS KEY 2

The rightmost character position of data-name-1 is known as status key 2 and is used to further describe the results of the input/output operation. This character position contains a value as indicated below:

a. If no further information is available concerning the input/output operation, status key 2 contains a value of '0'.

b. When status key 1 contains a value of '2' (indicating an INVALID KEY condition), status key 2 designates the cause of that condition as follows:

- A value of '1' in status key 2 indicates a sequence error for a sequentially accessed indexed file. The ascending sequence requirements of successive record key values have been violated, or the prime record key value has been changed by the COBOL program between a READ statement and the next REWRITE statement.
- A value of '2' in status key 2 indicates a duplicate key value. An attempt has been made to write or rewrite a record that creates a duplicate key in a relative or indexed file.
- A value of '3' in status key 2 indicates that no record was found. An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

VALID COMBINATIONS OF STATUS KEYS 1 AND 2

The valid combinations of the values of status key 1 and status key 2 are presented in the following table. The file organizations for which that combination is permissible are located at each intersection.

Status Key 1	Status Key 2				
	No Further Information (0)	Sequence Error (1)	Duplicate Prime Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion(0)	Sequential Relative Indexed		Indexed		
At End (1)	Sequential Relative Indexed				
Invalid Key(2)		Indexed	Relative Indexed	Relative Indexed	Relative Indexed
Input-Output Error (3)	Sequential Relative Indexed				Sequential

STATUS KEY 3

Status key 3 is a 4-digit numeric data item. Its value is moved to data-name-2 following the rules of the MOVE statement. Table 9-1 indicates the values of status key 3.

Status key 3 has the format wxyz, where:

wx describes the operation

yz describes the cause of the error

ACTIONS FOLLOWING INPUT/OUTPUT STATEMENT EXECUTION

The following actions take place after any input/output statement is executed (successful or not successful).

1. Appropriate values are set in status key 1, status key 2 and status key 3. (These all have zero values for a successful execution.) See Table 9-2 for the values.

2. If status key 1 is zero, processing continues in line in the user's program.

Table 9-1. Values of Status Key 3

w = COBOL I/O verb in which the error occurred.	
1	OPEN
2	CLOSE
3	READ
4	WRITE
5	REWRITE
6	START
7	DELETE
8, 9, 0	undefined
x = Multics I/O system subroutine that discovered the error.	
0	none
1	iox_$find_iocb
2	iox_$attach or iox_$open
3	iox_$detach or iox_$close
4	iox_$read_record, iox_$write_record, iox_$rewrite_record, or iox_$delete_record
5	iox_$get_line, iox_$get_chars, or iox_$put_chars
6	iox_$seek_key or iox_$position
7	iox_$control or iox_$modes
8	iox_$read_key or iox_$read_length
9	undefined
y = General category of the cause of the error.	
0	successful completion
1	at end of file
2	invalid key
3	permanent error
9	unable to make file available
z = Specific cause of the error. This depends on the value of y. Possible combinations of yz are:	
00	no error
01	short record
10	at end of file
21	invalid key - sequence error
22	invalid key - duplicate
23	invalid key - record not found
24	invalid key - new key is not equal to the old key
30	unspecified error
31	file not open
32	invalid operation for current open mode
33	previous I/O operation was not READ
34	new record length is not equal to the old record length
35	long record
36	file already open or already closed
90	cannot make file available
91	file is busy
92	format error in file
93	cannot attach or detach the I/O switch
94	attach and open modes are incompatible
95	file does not exist
97	label error

Table 9-2. Possible Values of Status Keys 1, 2, and 3 Returned by Multics COBOL

Status Key 3	Status Keys 1 and 2	Explanation
0000	00	No error - I/O operation successfully completed.
1032	30	Attempt to open external file already open in an inconsistent mode.
1036	30	Attempt to open internal file which is already open.
1090	30	Attempt to open a file closed with the LOCK option.
1094	30	Attempt to open "PREATTACHED" file which has not been previously attached.
1193	30	Unable to locate an existing IOCB or to create a new IOCB.
1232	30	OPEN is an invalid operation for the current attachment.
1290	30	Unable to open file (cause unspecified).
1291	30	File is busy (i.e., being used by another program).
1292	30	File contains format error or is of a different organization than specified.
1293	30	Unable to attach I/O switch.
1294	30	Attachment and attempted opening mode are incompatible.
1295	30	File does not exist.
1297	30	UNINITIALIZED VOLUME. DO YOU WANT TO INITIALIZE? (If the response is YES, the volume is initialized, and processing continues. If the response is NO, further processing cannot continue; the I/O switch is closed.
1332	30	Unable to close file to force specified attachment since CLOSE is an invalid operation for the current attachment.
2036	30	Attempt to close internal file which is not currently open.
2332	30	Close is an invalid operation for the current attachment.
2390	30	Unable to close file (cause unspecified).
2392	30	Unable to close file due to format error.
2393	30	Unable to detach IOCB after closing file.
3001	00	A record has been successfully read which is shorter than the specified input area.
3031	30	Attempt to read an unopen file.
3032	30	Attempt to read a file opened in output mode.
3430	30	Unable to read record (cause unspecified).
3410	10	Unable to read record; at end of file.
3423	23	Unable to read record; record does not exist.
3432	30	READ is an invalid operation for current open mode.
3435	30	A record has been read which is longer than the specified input area.
3510	10	Attempt to read beyond the end of a stream file.
3530	30	Unable to read next line in a stream file.
3535	30	A record has been read in a stream file which is longer than the specified input area.
3621	21	Unable to seek key for input; key out of order.
3623	23	Unable to seek key for input; record not found.
3630	30	Unable to seek key for input (cause unspecified).
3632	30	Seek_key is an invalid operation for current open mode.
3810	10	Unable to read next key; at end of file.
3830	30	Unable to read next key (cause unspecified).
3832	30	Read_key is an invalid operation for current open mode.

Table 9-2 (cont) Possible Values of Status Keys 1, 2, and 3 Returned by Multics COBOL !!!

Status Key 3	Status Keys 1 and 2	Explanation
4031	30	Attempt to write an unopen file.
4032	30	Attempt to write a sequentially accessed file opened as I-O.
4430	30	Unable to write record.
4432	30	Write is an invalid operation for the current open mode.
4530	30	Unable to write stream record.
4532	30	Get_line is an invalid operation for the current open mode.
4621	21	Unable to seek key for output; attempt to write with an out-of-order key to an indexed sequential file.
4622	22	Unable to seek for output; duplicate key.
4630	30	Unable to seek key for output (cause unspecified).
4632	30	Seek_key is an invalid operation for current open mode.
5024	30	Invalid REWRITE attempted; new key is not identical to old key.
5031	30	Attempt to rewrite an unopen file.
5032	30	Invalid rewrite attempted; attempt to rewrite a file not open as I-O.
5033	30	Invalid rewrite attempted; the previous I/O operation to this sequential file was not READ.
5034	30	Invalid rewrite attempted; new record length is not equal to old record length for this nonsequential file.
5423	23	Unable to rewrite record; record not found.
5430	30	Unable to rewrite record (cause unspecified).
5432	30	Rewrite is an invalid operation for current open mode.
5621	21	Unable to seek key for rewrite; sequence error.
5623	23	Unable to seek key for rewrite; record not found.
5630	30	Unable to seek key for rewrite (cause unspecified).
5632	30	Seek_key is an invalid operation for current open mode.
6031	30	Attempt to start an unopen file.
6032	30	Attempt to start an output file.
6623	23	Record not found with a key matching the specified key.
6630	30	Unable to seek key which matches specified key.
6632	30	Seek_key is an invalid operation for current open mode.
6723	23	Record not found with key satisfying specified key relationship.
6730	30	Unable to seek_head (iox_$control operation).
6732	30	The control operation "seek_head" is not allowed for the current open mode.
7031	30	Attempt to delete in an unopen file.
7032	30	Attempt to delete in a file opened in output mode.
7033	30	Invalid delete attempted; the previous I/O operation on this sequential file was not READ.
7423	23	Unable to delete record; record not found.
7430	30	Unable to delete record (cause unspecified).
7432	30	Delete is an invalid operation for current open mode.
7621	21	Unable to seek key for delete; sequence error.
7623	23	Unable to seek key for delete; record not found.
7630	30	Unable to seek key for delete (cause unspecified).
7632	30	Seek_key is an invalid operation for current open mode.

Error Processing

When an error occurs on an input/output operation, the user can exercise a degree of control through specification of USE procedures that can inspect the FILE STATUS item associated with the file.

The following steps are taken when an input/output error is encountered:

1. A value is placed in the FILE STATUS item, if one is specified, to indicate the nature of the error. Refer to "Input/Output Status" earlier in this section for a description of the various error conditions and codes.

2. If the error is an AT END or INVALID KEY condition:

   a. If an AT END or INVALID KEY phrase , respectively, is specified in the input/output statement, control is transferred to the AT END/INVALID KEY imperative statement and a USE statement, if specified, is not executed.

   b. If the AT END or INVALID KEY phrase is not specified in the input/output statement, and a USE procedure is specified for the file, the USE procedure is executed.

   c. If the AT END or INVALID KEY phrase is not specified in the input/output statement, and a USE procedure is not specified for the file, the action taken is the same as for a non-I/O error. (Refer to "Non-Input/Output Errors" in Section 7.)

3. If the error is not an AT END or INVALID KEY condition:

   a. If a USE procedure is specified for the file, it is executed.

   b. If a USE procedure is not specified for the file, the action taken is the same as for a non-I/O error. (Refer to "Non-Input/Output Errors" in Section 7.)

4. In certain special cases the programmer cannot gain control of the error by a USE procedure. These cases are identified by a value of zero in status key 1 in Table 9-2. These cases are generally not errors, but normal conditions. If the programmer wishes to check these conditions, a test of status key 3 should be included after the input/output statement.

AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition, refer to "READ Statement" later in this section.

END-OF-PAGE Condition

The END-OF-PAGE condition can occur as the result of the execution of a WRITE statement. For details of the causes of the condition, refer to "WRITE statement" later in this section.

INVALID KEY Condition

The INVALID KEY condition can occur as the result of the execution of a START, READ, WRITE, REWRITE, or DELETE statement. For details of the causes of the condition, refer to the descriptions of the various statements later in this section.

LINAGE-COUNTER Register

The reserved word LINAGE-COUNTER is a name for a special register generated by the presence of a LINAGE clause in a file description entry. The implicit description is that of an unsigned integer whose size is equal to the size of integer-1 or the size of the data item referenced by data-name-1 in the LINAGE clause. For rules governing the linage counter, refer to "LINAGE Clause" later in this section.

Label Processing for Tape Files

Label processes are applied at appropriate times to magnetic tape files for which standard labels are specified. Files allocated to magnetic tape header and trailer labels are separate blocks recorded before the first physical record on each reel and following the last physical record, respectively.

Header label creation takes place whenever a labeled file is opened for output or a new volume of a multivolume sequential output file is accessed. A label record containing the FILE-ID and/or RETENTION period specified in the source program is recorded on the file.

Trailer label creation takes place whenever a labeled output file allocated to magnetic tape is closed or, for a multireel version of such a file, whenever an end-of-tape condition is encountered or a CLOSE REEL statement is executed.

Header label checking takes place whenever a labeled file is opened for input, extension, or input-output, or when a new volume of a multivolume sequential input file is accessed.

Trailer label checking occurs whenever an end-of-volume or end-of-file condition is encountered on a labeled input file allocated to magnetic tape.

Reel/Unit Swap Procedure

A reel/unit swap occurs upon the execution of a CLOSE REEL/UNIT statement and whenever execution of an input-output operation on a multivolume sequential file results in an end-of-volume condition on the current volume. It causes the next volume in sequence to become the current volume.

ENVIRONMENT DIVISION FOR INPUT/OUTPUT

Input-Output Section

The Input-Output Section provides the information needed to control the transmission and handling of data between external devices and the object program. It is subdivided into the following two paragraphs:

- FILE-CONTROL paragraph, which names all files used in the program and associates them with I/O switch names.
- I-O-CONTROL paragraph, which defines special input/output control techniques to be used in executing the object program.

General Format:

```
ENVIRONMENT DIVISION.
[ INPUT-OUTPUT SECTION.
  FILE-CONTROL.  {file-control-entry} ...
[ I-O-CONTROL.  [input-output-control-entry] ] ]
```

FILE-CONTROL PARAGRAPH

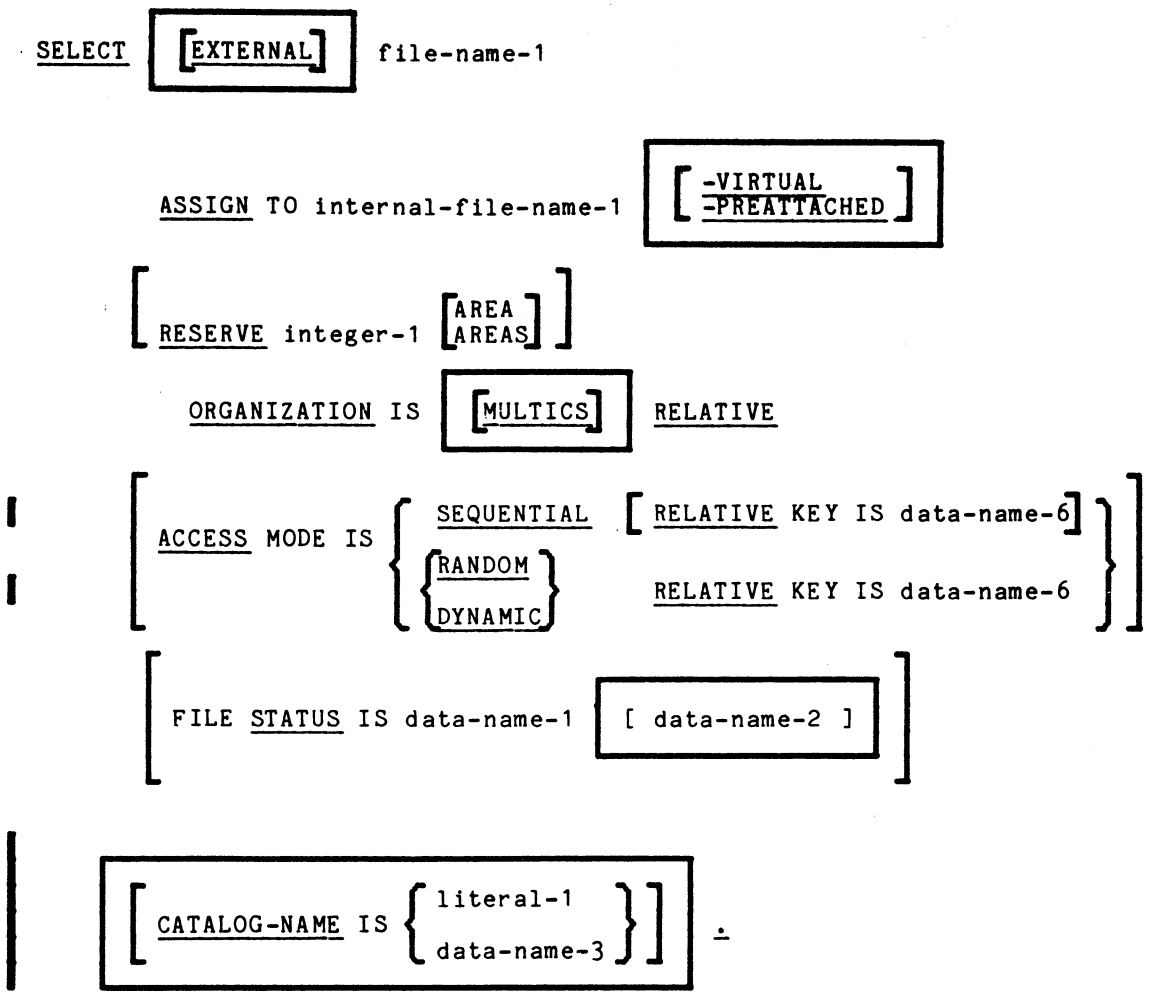
The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

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

Format 1 (Sequential Files)

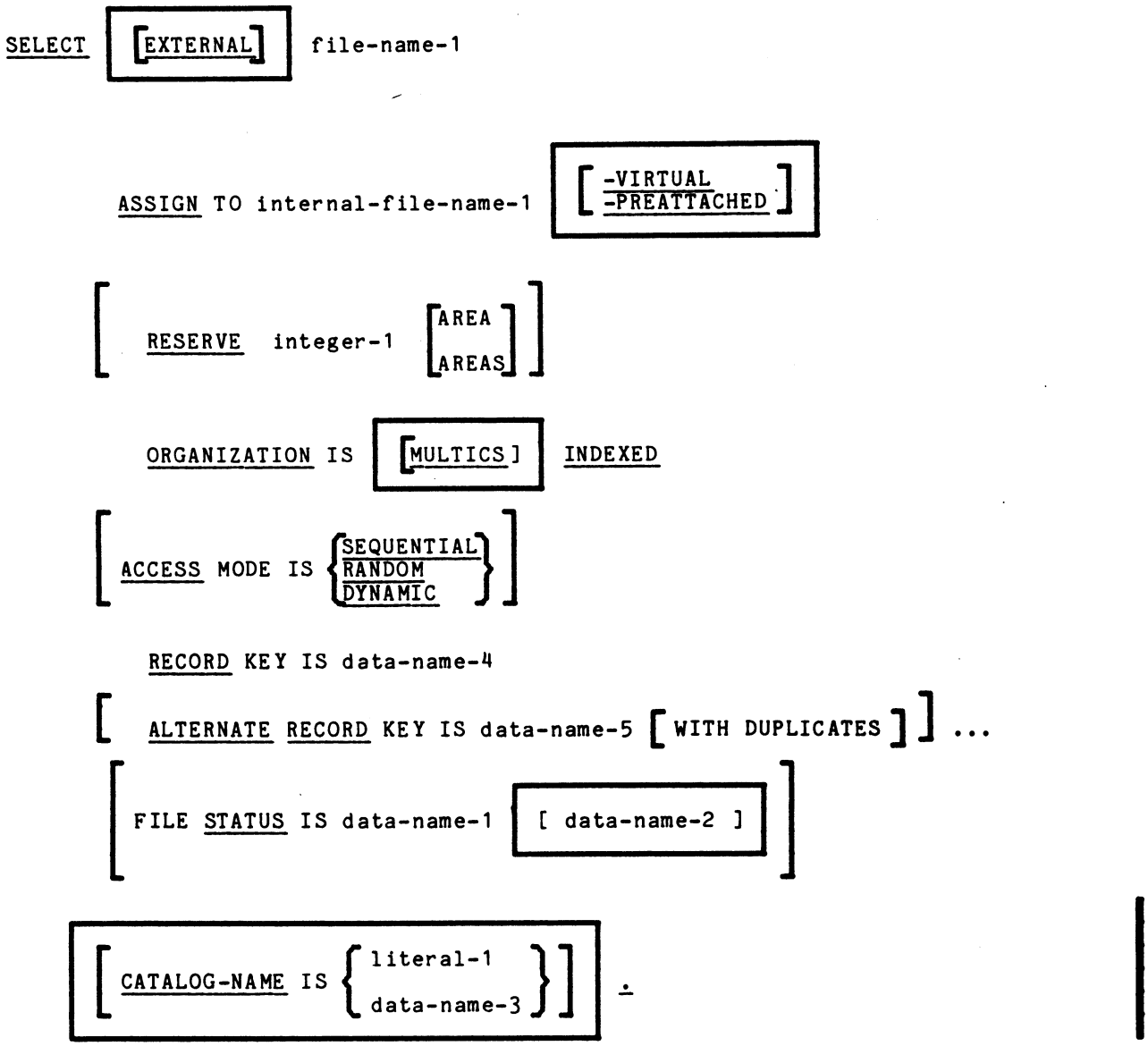
```
SELECT [OPTIONAL]  [EXTERNAL]  file-name-1

        ASSIGN  TO  internal-file-name-1  [ -VIRTUAL
                                          -PRINTER
                                          -CARD-PUNCH
                                          -CARD-READER
                                          -PREATTACHED
                                          -TAPE ]

[ RESERVE integer-1 [AREA
                     AREAS] ]

[ ORGANIZATION IS [ANSI
                   MULTICS
                   IBM-DOS
                   IBM-OS]  SEQUENTIAL ]

[ ACCESS MODE IS SEQUENTIAL ]

[ FILE STATUS IS data-name-1 [ data-name-2 ] ]

[ CATALOG-NAME IS { literal-1
                  { data-name-3 } ]

[ [WITH] { FLR
         { VLR
         { SPANNED } ] .
```

Format 2 (Relative Files):



FILE-CONTROL

Format 3 (Indexed Files):



Format 4 (Stream Files):

```
SELECT [EXTERNAL] file-name-1
       ASSIGN TO internal-file-name-1 [ -VIRTUAL
                                      | -PREATTACHED
                                      | -PRINTER ]
     [ RESERVE integer-1 [AREA
                         |AREAS] ]
       ORGANIZATION IS [MULTICS] STREAM
     [ ACCESS MODE IS SEQUENTIAL ]
     [ FILE STATUS IS data-name-1 [ data-name-2] ]
     [ CATALOG-NAME IS {literal-1   } ] .
                       {data-name-3}
```

FILE-CONTROL

Syntax Rules:

1. The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause can appear in any order.

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

3. Internal-file-name-1 can contain from one to 16 ASCII graphic characters. The blank, minus (or hyphen), and leading or trailing period characters cannot be part of internal-file-name-1.

4. In Format 1, when the ORGANIZATION IS SEQUENTIAL clause is not specified, the clause is implied.

5. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

6. Data-name-1, data-name-2, data-name-3, data-name-4, data-name-5 and data-name-6 may be qualified.

7. Data-name-1 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section, the Report Section, or the Communication Section. The left character is known as status key 1 and the right character is status key 2.

   Data-name-1 may not be defined in the Constant Section.

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

9. Data-name-6 must not be defined in a record description associated with a file name. The data item referenced by data-name-6 must be an unsigned integer.

   If data-name-6 is defined in a record description associated with the file-name, a warning diagnostic is issued. Data-name-6 must not be defined in the CONSTANT SECTION.

10. The data items referenced by data-name-4 and data-name-5 must be defined as a data item of category alphanumeric within a record description entry associated with the file-name. Neither data-name-4 nor data-name-5 can describe an item whose size is variable.

FILE-CONTROL

If data-name-4 is defined by a data item in the working storage section, a warning diagnostic is issued. The sizes of data-name-4 and data-name-5 must not exceed 255 characters.

11. Data-name-5 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name-4, or by any other data-name-5 associated with the file.

12. The device code is used to specify the physical medium on which the file resides and (in part) the Multics I/O module used for file processing.

If the device code is -VIRTUAL, -CARD-READER, -CARD-PUNCH, or -PRINTER, or if the device code is omitted, the file resides in virtual memory and the I/O module vfile_ is used.

If the device code is -TAPE, the file resides on magnetic tape reels. The I/O module to be used is determined by the ORGANIZATION clause.

a. ANSI-SEQUENTIAL specifies the use of the I/O module tape_ansi_.

b. IBM-OS-SEQUENTIAL specifies the use of the I/O module tape_ibm_.

c. IBM-DOS-SEQUENTIAL specifies the use of the I/O module tape_ibm_.

d. MULTICS SEQUENTIAL specifies the use of the I/O module tape_ibm_ (if the LABEL RECORDS OMITTED clause was used in the FD entry) or the I/O module tape_ansi_ (if the LABEL RECORDS STANDARD clause was used in the FD entry).

If the device code is -PREATTACHED, the I/O module and the file medium are selected by the user (by executing the Multics command io_call) prior to the execution of the program. In this case the file is presumed to be an external file.

13. If a device-code is specified (e.g., -VIRTUAL or -PREATTACHED), then it must be appended to the internal-file-name without any intervening blanks.

14. Data-name-2 specifies the variable item into which status key 3 is moved.

FILE-CONTROL

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

16. ORGANIZATION IS ANSI, IBM-OS, or IBM-DOS SEQUENTIAL may be specified only when the device code -TAPE has been specified. For tape files, ORGANIZATION IS MULTICS SEQUENTIAL is equivalent to ORGANIZATION IS ANSI SEQUENTIAL.

17. For non-tape files, the word MULTICS in the organization clause is for documentation only.

General Rules (all formats):

1. The ASSIGN clause associates the COBOL file referenced by file-name in the SELECT clause with a physical file. The following abbreviations are used in describing the process:

   P: the program name (from the PROGRAM-ID paragraph)
   I: the internal file name (from the ASSIGN clause)
   G: a 15-character unique name provided by the operating system

   a. For non-tape files, if the VALUE OF CATALOG-NAME clause is not specified in the file control entry or in the file description (FD) entry, the physical file is the virtual memory segment or multisegment file identified by the relative pathname "P.I".

      For tape files, either the VALUE OF CATALOG-NAME clause must be present or the file must be attached before the program is executed.

   b. The name of the I/O switch associated with the COBOL file is constructed as follows:

      for external files: I
      for internal files: I.G

      The I/O switch name can be used in references to Multics Commands (in particular io_call). If the file in question is an internal file, typing the I/O switch name may prove tedious.

   c. More detailed information on the interface of COBOL files with the Multics operating system may be found in the Multics COBOL Users' Guide, Order No. AS43.

2. The RESERVE clause allows the user to specify the number of input-output areas allocated.

The RESERVE clause is for documentation only.

FILE-CONTROL

3. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed by the program.

4. When ACCESS MODE IS SEQUENTIAL is specified, records in the file are accessed in the sequence dictated by the file organization.

   a. For sequential and stream files, this sequence is determined by the order of the WRITE statements when the file is created or extended.

   b. For relative files, this sequence is the order of ascending relative record numbers of existing records in the file.

   c. For indexed files this sequence is the order of ascending record key values within a given key of reference. ASCII collating sequence is used.

FILE-CONTROL

5. When the FILE STATUS clause is specified, a value is moved into the data item specified by data-name-1 (and data-name-2) after the execution of every statement that references that file, either explicitly or implicitly. This value indicates the status of the execution of the statement. Data-name-2, if specified, is set to a number associated with a corresponding error message.

6. The key word EXTERNAL in the SELECT clause specifies that the I/O switch associated with the COBOL file can be used by more than one program in the COBOL run unit. If the key word EXTERNAL is not used then the name associated with the I/O switch is unique within the operating system.

7. The default for the device code is -VIRTUAL.

8. If -PRINTER is specified, the final destination of the output file is a line printer. The file is created in virtual memory and may be sent to the printer by using the Multics dprint command. If -CARD-PUNCH is specified, the final destination of the output file is a card punch. The file is created in virtual memory and may be sent to a card punch by using the Multics dpunch command. If -CARD-READER is specified, the input file must have been previously placed in virtual memory by using the Multics punched card input facility.

9. If the device code -TAPE is specified, the COBOL file is assumed to be on magnetic tape. In this case the VALUE OF CATALOG-NAME clause is used to identify the tape reel (unless the I/O switch is attached prior to the execution of the program).

   The following rules apply to the names used to identify tape reels:

   a. If an IBM or ANSI tape is required, the reel identifier (specified as the value of literal-1 or the data item referenced by data-name-3) must consist of six or fewer characters. If the reel name is numeric, it is padded on the left with zeros by the I/O module involved; otherwise, it is padded on the right with blanks.

   b. If a non-standard tape is required, the reel name must consist of a number optionally followed by either ",7-track" or ",9-track", according to whether the tape is a 7-track or a 9-track reel. If neither string appears, then "9-track" is assumed.

10. If a non-tape file is specified by the device code, the VALUE OF CATALOG-NAME clause specifies the path name (full or relative) of the corresponding virtual memory segment or multisegment file.

FILE-CONTROL

11. The following rules apply to the I/O switch associated with the COBOL file:

    a. If the I/O switch associated with a COBOL file is not attached when the COBOL file is opened, the I/O switch is attached.

    b. When a COBOL file is closed, the I/O switch for the file is detached, but only if the switch was attached when the file was opened (in step a).

    c. The APPLY clause in the I-O-CONTROL paragraph can be used to override these rules.

12. If the device code -PREATTACHED is specified, then the I/O switch associated with the COBOL file must have been previously attached.

13. Virtual files are segments or multisegment files in the Multics virtual memory storage system.

14. Virtual files are the normal Multics files; that is, they do not have a special purpose such as printing or punching.

15. For ANSI or IBM tape files, VLR indicates that the internal structure of the file is in variable-length record format. Records in this case must not exceed 8188 characters in length. For other files, VLR indicates the file, when written, is to be handled as though it had variable-length records and thus this specification is of interest only in special cases, when special formatting of the file is desired.

16. For ANSI or IBM tape files, FLR indicates that the internal structure of the file is in fixed-length record format. Records in this case must not exceed 8192 characters in length. For other files, FLR indicates the file, when written, is to be handled as though it had fixed-length records and thus this specification is of interest only in special cases, when special formatting of the file is desired. If the records are described as having different sizes or are variable in length, the maximum size is always written. Records smaller than the maximum size are not padded. The extraneous characters are those from the previous contents of the record area.

17. For ANSI or IBM tape files, SPANNED indicates the internal structure of the file is spanned record format. Records in this case must not exceed 1,044,480 characters in length.

18. For additional information concerning COBOL object time input/output, refer to the Multics COBOL Users' Guide.

FILE-CONTROL

General Rules (Format 1):

1. ORGANIZATION IS ANSI SEQUENTIAL implies a standard ANSI tape file. Attachment is made to the tape_ansi_ I/O module.

2. ORGANIZATION IS IBM-DOS SEQUENTIAL implies IBM DOS tape file. Attachment is made to the tape_ibm_ I/O module -DOS control argument.

3. ORGANIZATION IS IBM-OS SEQUENTIAL implies an IBM OS tape file. Attachment is made to the tape_ibm_ I/O module.

General Rules (Format 2):

1. In a relative file, each record is identified by its logical ordinal position within the file.

2. Format 2 must be specified for relative files.

3. ORGANIZATION IS RELATIVE implies a Multics relative file. With this organization, the RELATIVE KEY specified (data-name-6) represents the record's logical ordinal position in the file.

General Rules (Formats 2 and 3):

1. If ACCESS MODE IS RANDOM is specified, the value of the relative key data item for relative files or the value of a record key data item for indexed files indicates the record to be accessed.

2. When ACCESS MODE IS DYNAMIC is specified, records in the file can be accessed either sequentially or randomly.

General Rules (Format 3):

1. In an indexed file each record is identified by the value of a key (data-name-4).

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

3. An ALTERNATE RECORD KEY clause specifies a record key that is an alternate record key for the file. The alternate record key provides an alternate access path to records in an indexed file.

4. The data descriptions of data-name-4 and data-name-5 as well as their relative positions within a record must be the same as that used when the file was created. The number of alternate keys for the file must be the same as that used when the file was created.

FILE-CONTROL

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

General Rules (Format 4):

1. Format 4 must be specified for unstructured files which are to be read and written as a stream of characters.

2. Each record on the file is terminated by a newline character. The newline character is not part of the record.

I-O-CONTROL

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph specifies the memory area that is to be shared by the different files, the location of the files on a multiple file reel, and input/output techniques to be applied to the files.

General Format:

```
I-O-CONTROL.

[ APPLY input-output-technique ON {file-name-1} ... ] ...

[ RERUN [ ON { file-name-2      } ] ]
       [       { CHECKPOINT-FILE } ]

            [ EVERY { [ [ END OF ] {REEL} ]          } OF file-name-3 ] ] ...
                    {   [        ] {UNIT} }          }                }
                    { integer-1 RECORDS               }                }
                    { integer-2 CLOCK-UNITS           }                }
                    { condition-name                  }                }

[ SAME { RECORD    } AREA FOR {file-name-4} ... ] ...
       { SORT      }
       { SORT-MERGE }

[ MULTIPLE FILE TAPE CONTAINS

    { file-name-5 [ POSITION IS integer-3 ] } ... ] ... .
```

Syntax Rules:

1. The I-O-CONTROL paragraph is optional.
2. The RERUN clause is for documentation purposes only.
3. The two forms of the SAME clause (SAME AREA, SAME RECORD AREA) are considered separately in the following:

   More than one SAME clause can 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.

I-O-CONTROL

c. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

d. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all 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 can also appear in that SAME RECORD AREA clause. 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.

e. If a file-name that does not represent a sort file appears in a SAME AREA clause and in one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all files named in that SAME AREA clause must be named in the SAME SORT AREA or SAME SORT-MERGE AREA clause(s).

4. The files referenced in the SAME AREA or SAME RECORD AREA clause are not required to have the same organization or access.

5. File-name-5, etc. must be specified with -TAPE in the associated associated ASSIGN clause. If file-name-5, etc. is specified with -PREATTACHED, the associated device must be a tape device.

6. Each file-name specified in a MULTIPLE FILE TAPE clause must have an associated file description entry.

7. The same file-name must not be specified more than once in a MULTIPLE FILE TAPE clause.

General Rules:

1. Since more than one input/output technique is available, the APPLY clause permits the user to select the appropriate technique for the object program. Specific names to designate the individual input/output techniques are specified under "Input/Output Techniques."

2. The SAME AREA clause specifies that it is invalid to have more than one file open at the same time. (Refer to Syntax Rule 3d above.)

3. 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 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.

I-O-CONTROL

4. The MULTIPLE FILE clause is required 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 of the same tape reel may be open at one time.

Input/Output Techniques

The input-output-technique variable specified in the APPLY clause of the I-O-CONTROL paragraph can consist following options.

```
[ FILE IS { TEMPORARY } ]
          { PERMANENT }

[ NO DETACH AT CLOSE ]

[ ATTACH-OPTIONS ARE { literal-1   } ]
                     { data-name-1 }

[                     { OUTPUT-MODE IS { GENERATION                          } }     ]
[                     {                { MODIFICATION                        } }     ]
[                     {                { REPLACEMENT { literal-2   }         } }     ]
[                     {                {             { data-name-2 }         } }     ]
[                     {                                                        }     ]
[                     { DEVICE IS { integer-1   }                            }     ]
[                     {           { data-name-3 }                            }     ]
[                     {                                                        }     ]
[ TAPE-OPTIONS ARE    { DENSITY IS { 800  }                                  } ... ]
[                     {            { 1600 }                                  }     ]
[                     {            { 6250 }                                  }     ]
[                     { RETAIN                                               }     ]
[                     { FORCE                                                }     ]
[                     { PROTECT                                              }     ]
[                     { ADDITIONAL { CATALOG-NAME IS   } { literal-3   } ... }     ]
[                     {            { CATALOG-NAMES ARE } { data-name-4 }     }     ]
```

I-O-CONTROL

Syntax Rules:

1. Contradictory input/output techniques must not be specified for the same file.

2. If -PREATTACHED is specified in the ASSIGN clause of the FILE-CONTROL entry, the ATTACH-OPTIONS and TAPE-OPTIONS clauses cannot be specified.

3. If the ATTACH-OPTIONS clause is specified, the TAPE-OPTIONS clause must be omitted. If the TAPE-OPTIONS clause is specified, the ATTACH-OPTIONS clause must be omitted. Each option within the TAPE-OPTIONS clause may be specified, but only once.

4. The value of literal-2 or the data item referenced by data-name-2 can contain from one to 16 ASCII graphic characters, excluding the blank, minus (or hyphen), and leading and trailing period characters.

5. The value of the data item referenced by data-name-3 must be an integer.

6. The ADDITIONAL CATALOG-NAME specified as the value of literal-3 or the data item referenced by data-name-4 must be six or less characters in length and must be nonnumeric. Data-name-4 must be a data description entry in the Working-Storage section.

General Rules:

1. The FILE IS TEMPORARY clause specifies that the associated file is to be deleted when the COBOL run-unit is terminated. The FILE IS PERMANENT clause indicates that the file is not to be deleted. If the FILE clause is not specified, PERMANENT is assumed. The FILE clause is ignored for external files.

2. The NO DETACH AT CLOSE clause indicates that the file is not to be detached from the associated device when the file is closed. If this clause is omitted then the file is detached (by a CLOSE statement) if it was attached (by the OPEN statement).

3. The file in the ATTACH-OPTIONS clause must be EXTERNAL. The value of literal-1 or the data item referenced by data-name-1 is in the format:

   "io-module-name option-1...option-n"

   where the options depend upon the particular I/O module specified by io-module-name. This specification overrides any attachment implied by the internal-file-name phrase and CATALOG-NAME clause. (Refer to Multics Programmers' Manual, Subroutines for descriptions of available I/O modules.)

4. The TAPE-OPTIONS phrase within the APPLY clause must be specified whenever the user wishes to use options other than defaults.

5. The OUTPUT-MODE clause allows one of three output operations to be performed on an existing ANSI or IBM tape file. Though each operation functions in a significantly different manner, they all share a common characteristic: when an output operation is specified on an existing file, that file is logically truncated at the point of operation, thereby destroying all files (if any) that follow consecutively from that point.

   a. If GENERATION is specified, the generation number recorded in the labels that bracket every file are incremented by one. Initially set to zero when the file was created, this number is increased by one for each successive generation of the file. When the number reaches 9999, the next increment resets it to zero.

   b. If MODIFICATION is specified, the entire contents of a file are replaced, but the structure of the file itself (as recorded in the labels) is retained. Every time a file is modified, the version number in its trailer labels is incremented by one. When the number reaches 99, the next increment resets it to zero.

   c. If REPLACEMENT is specified, a new file is added by replacing (overwriting) an existing file specified by the value of literal-2 or the data item referenced by data-name-2. If no file having a value of literal-2 or the data item referenced by data-name-2 exists, a new file is appended to the tape.

   d. If the file to be generated or modified does not exist, an error is indicated and control returns to Multics command level. If any or all TAPE-OPTIONS are specified, they must correspond to those recorded in the labels of previous GENERATION or MODIFICATION phrases.

I-O-CONTROL

e. If OUTPUT-MODE is not specified for a file opened in OUTPUT mode, a new file is created; an entirely new entity is added to the tape. This new file is appended immediately after the last (or only) file on the tape. The process of appending does not alter the previous contents of other files on the tape.

6. The DEVICE option specifies the maximum number of tape drives that can be used by a given ANSI or IBM tape file during an attachment. The value of integer-1 or the data item referenced by data-name-3 must range from 1 to n, where n equals the maximum number of drives on the system, but not greater than 63. When the DEVICE option is not specified, the default for an initial attachment to a file is one tape drive.

7. The DENSITY option specifies the density at which the tape file is recorded. Every file on a multiple file tape must be recorded at the same density. In the absence of the DENSITY option, density is determined as follows:

   open for input: DENSITY = density of the volume label

   open for output: (creating a new reel) DENSITY = 800 bpi

   open for output: (for existing reel) DENSITY = density of the volume label

8. The RETAIN option specifies the retention of resources across attachments, where resources equal devices (tape drives) and files (single-reel or multiple-reel ANSI or IBM tapes). When RETAIN is specified, all devices and files remain assigned to the process. Otherwise, all devices and files are unassigned.

9. The FORCE option specifies that the expiration date of the ANSI or IBM tape file being overwritten is to be ignored. EXTEND, MODIFICATION, GENERATION, and REPLACEMENT are overwrite operations. The expiration date is recorded in Julian form, and is set only when a file is created or generated.

10. The PROTECT option specifies that an ANSI or IBM tape set will be mounted without a write ring. Whenever a file set is needed, the I/O module message that specifies the tape to be mounted without a ring. If PROTECT is not specified, the tape will be mounted with a write ring. If a tape is mounted with a write ring and the I/O switch is opened for sequential input, the hardware file-protect feature is used to safeguard the tape.

11. The ADDITIONAL option specifies a list of volume identifiers that will be required within the process. If ADDITIONAL is not specified, no additional CATALOG-NAME(S) is needed, since either the file is contained on one reel or the user wishes to be interrogated when reel changes take place.

12. Refer to the Multics COBOL Users' Guide and the Multics Programmers' Manual, Reference Guide for additional information on object time I-O.

DATA DIVISION FOR INPUT/OUTPUT

File Section

In a COBOL program, the file description (FD) entry 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 records, the value of label items, and the names of the data records of which the file is composed. The entry itself is terminated by a period.

Record Description Structure

A record description consists of a set of data description entries which 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 can vary considerably, depending upon whether or not the entry is followed by subordinate entries. The structure of a record description is further defined in the Concept of Levels paragraph in Section 2, while the elements allowed in a record description are shown in the data description skeleton paragraph in Section 6.

FD SKELETON

File Description - Complete Entry Skeleton

The file description furnishes information concerning 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

      [ [ DEPENDING  ON  data-name-1 ] ] ]

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

[ VALUE OF { FILE-ID IS { data-name-2 }                    } ]
           {           { literal-1   }                     }
           { RETENTION IS { data-name-3 }                  }
           {              { literal-2   }                  }
           { CATALOG-NAME IS { data-name-4 }               }
           {                 { literal-3   }               } ]

[ DATA { RECORD IS  } { data-name-5 } ... ]
       { RECORDS ARE }

[ LINAGE IS { data-name-6 } LINES [ WITH FOOTING AT { data-name-7 } ]
            { integer-5   }                        { integer-6   }

[ LINES AT TOP { data-name-8 } ] [ LINES AT BOTTOM { data-name-9 } ] ]
               { integer-7   }                     { integer-8   }

[ CODE-SET IS alphabet-name ]
```

FD SKELETON

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 most cases and their order of entry is not significant.

3. One or more record description entries must follow the file description entry.

BLOCK CONTAINS

BLOCK CONTAINS CLAUSE

The BLOCK CONTAINS clause specifies the size of a physical record.

General Format:

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

General Rules:

1. The BLOCK CONTAINS clause is meaningful only for ANSI or IBM tape files. Otherwise, it serves only as documentation.

2. The default for this clause is: BLOCK CONTAINS 1 RECORD.

3. When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record. When the word RECORDS is specified, the physical record size is specified in terms of logical records.

4. If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum sizes of the physical record, respectively.

5. Specification of integer-1 is not meaningful but is accepted for documentation only.

6. If the file is fixed-length record format, block size must be greater than or equal to record size. If block size is not an exact multiple of record size, the actual block size used is integer-2 rounded downward to an exact multiple, and an observation is issued.

7. If the file is variable-length record format, block size must be greater than or equal to record size.

8. If the file is spanned record format, block size may be less than, equal to, or greater than record length.

9. In all cases, block size must be in the ranges:

   $18 \leq$ block size ≤ 8192 (tape_ibm_)

   $20 \leq$ block size ≤ 8192 (tape_ansi_)

BLOCK CONTAINS

Notes

1. Use of a block size greater than 2048 is a nonstandard Multics feature.

2. The term "record size" refers to the actual or maximum number of characters in the record as specified in the RECORD CONTAINS clause or as implied by the record description(s) in the FD entry; "block size" refers to the number of characters in a block. If the BLOCK CONTAINS clause is not specified, block size is identical to record size, and records are unblocked. If BLOCK CONTAINS ... RECORDS is specified, block size is equal to integer-2 multiplied by record size. If BLOCK CONTAINS ... CHARACTERS is specified or implied, block size is integer-2.

CODE-SET CLAUSE

The CODE-SET clause of the FD entry 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 alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase.
3. The CODE-SET clause must not be specified for relative or indexed files.

General Rules:

1. If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on external media. It also specifies the algorithm for converting character codes on external media from/to native character codes. This code conversion occurs during execution of an input or output operation. (Refer to "SPECIAL-NAMES Paragraph" in Section 6.)
2. If the CODE-SET clause is not specified, the native character set (ASCII) is assumed for data on the external media, and no code conversion takes place.

DATA RECORDS

DATA RECORDS CLAUSE

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

General Format:

```
DATA { RECORD  IS  } { data-name-1 } ...
     { RECORDS ARE }
```

Syntax Rules:

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

General Rules:

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records can be of differing sizes, different formats, etc. The order in which they are listed is not significant.

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

LABEL RECORDS CLAUSE

The LABEL RECORDS clause specifies whether labels are present.

General Format:

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

Syntax Rules:

1. The LABEL RECORDS clause is required in every file description entry.

General Rules:

1. If the device code -TAPE is specified along with MULTICS SEQUENTIAL organization, LABEL RECORDS OMITTED specifies use of the I/O module tape_ibm_ and LABEL RECORDS STANDARD specifies use of the I/O module tape_ansi_.
2. For all other files, this clause serves as documentation only.

LINAGE

LINAGE CLAUSE

The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. It also provides for specifying the top and bottom margins on a page, and the line number with a 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, and data-name-4 must reference elementary unsigned numeric integer data items.
2. The value of integer-1 must be greater than zero.
3. The value of integer-2 must not be greater than integer-1.
4. The value of integer-3 and integer-4 may be zero.
5. If the LINAGE clause is used with an external file, only integers are permitted in the clause and they must be the same in all programs in the run unit which declare the file.

LINAGE

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 of 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.

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 a logical page. The value must be greater than zero. That portion of a logical page in which lines can be written 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 that comprise the top margin of a 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 comprise the bottom margin of 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. This value must be greater than zero and not greater than the value of integer-1 or of 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 of integer-2 or the data item referenced by data-name-2 and the line represented by the value of integer-1 or the data item referenced by data-name-1, inclusive.

6. The values of integer-1, integer-3, and integer-4, if specified, will be used when the file is opened by the execution of an OPEN statement with the OUTPUT phrase to specify the number of lines that comprise each of the indicated sections of a logical page. The value of integer-2 or the data item referenced by data-name-2, if specified, is used at that time to define the footing area. These values are used for all logical pages written for the file during a given execution of the program.

7. The values of 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 comprise 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, will be used to specify the number of lines that are to comprise each of the indicated sections for the next logical page.

LINAGE

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. Rules governing the LINAGE-COUNTER are as follows:

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

   b. The LINAGE-COUNTER may be referenced, but not modified, by statements in the Procedure Division. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify the LINAGE-COUNTER by the file-name when necessary.

   c. The LINAGE-COUNTER is automatically modified, according to the following rules, during 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 the 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 of one.

      4) The value of LINAGE-COUNTER is automatically reset to one whenever the device is repositioned to the first line that can be written for each of the succeeding logical pages.

   d. The value of LINAGE-COUNTER is automatically set to one when 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 CLAUSE

The RECORD CONTAINS clause specifies the size of data records.

General Format:

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

  [ DEPENDING  ON  data-name-1 ]
```

Syntax Rules:

1. Data-name-1 may be qualified.

General Rules:

1. The size of each data record is completely defined within the record description entry; therefore, this clause is never required. When it is present, however, the following rules apply:

   a. Integer-2 cannot be used by itself unless all of 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 minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively. Integer-1 can be zero.

   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 fixed-length elementary items plus the sum of the maximum number of characters in any variable-length table item subordinate to the record. This sum can be different from the actual size of the record. (Refer to "USAGE Clause" in Section 6.)

RECORD CONTAINS

2. The DEPENDING ON clause cannot be specified when FLR has been specified.

3. Data-name-1 must be a numeric integer data item.

4. The WRITE and REWRITE statements use the contents of data-name-1 to determine the size of the record. The READ statement causes the contents of data-name-1 to contain the size of the record read. Note that any excess characters in the current record area are not altered.

5. For ANSI and IBM tape files, the following restrictions must be observed:

   a. For fixed-length record format files, record size must not exceed 8192 characters.

   b. For variable-length record format files, record size must not exceed 8188 characters.

   c. For spanned record format files, record size must not exceed 1,044,480 characters.

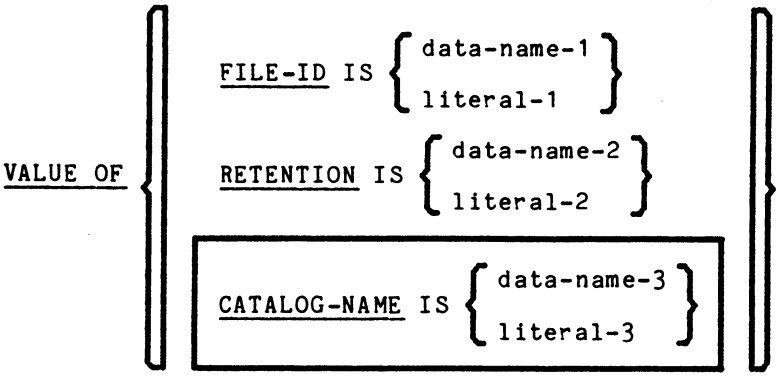
   d. For unblocked files of all formats, the minimum allowable record size is 18 characters.

VALUE OF

VALUE OF CLAUSE

The VALUE OF clause in the FD entry specifies the expected or desired contents of the file identification and/or retention fields of the file labels.

General Format:



Syntax Rules:

1. Data-name-1, data-name-2, and data-name-3 should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

2. Data-name-1 and data-name-2 must be in the Working-Storage Section.

3. FILE-ID and RETENTION reference data items within the appropriate label.

4. Literal-3 or the contents of the data item referenced by data-name-3 must be nonnumeric. Data-name-3 must be a data description entry in the Working-Storage Section.

General Rules:

1. The VALUE OF FILE-ID and the VALUE OF RETENTION clause is meaningful only for tape files and otherwise serves only for documentation.

2. FILE-ID specifies the name to be used in labels and is meaningful only with ORGANIZATION IS ANSI, IBM-OS, and IBM-DOS SEQUENTIAL files. (Refer to the SELECT clause.)

3. RETENTION specifies the expiration date of a tape file being created. The value specified by literal-2 or the data item referenced by data-name-2 indicates the minimum number of days the file is to be retained after its creation date. This value must be an unsigned numeric data item of no more than three digits.

4. For an input file, the appropriate label routine checks to see if the value of FILE-ID or RETENTION is equal to the value of either literal-1 or the data item referenced by data-name-2, whichever has been specified.

VALUE OF

For an output file, at the appropriate time the value of FILE-ID or RETENTION is made equal to the value of either literal-1 or the data item referenced by data-name-2, whichever has been specified.

5. If -TAPE is specified in the SELECT clause, the file is assumed to be on tape. For tape files, the VALUE OF CATALOG-NAME clause identifies the volume identification of the tape file.

   The VALUE OF CATALOG-NAME clause must be present for all tape files. For ANSI, IBM, and nonstandard tape, the following holds:

   The volume identification specified as the value of literal-1 of the data item referenced by data-name-3 must be six characters or less in length. If the VALUE OF CATALOG-NAME is less than six characters and entirely numeric, it is padded on the left with zeros by the I/O module. If the VALUE OF CATALOG-NAME is less than six characters and not entirely numeric, it is padded on the right with blanks.

6. The VALUUE OF CATALOG-NAME clause identifies the segment for virtual files. Literal-1 or the contents of data-name-3 is the pathname (full of relative) of the corresponding segment or multisegment file. If it is not specified, the character-string program-name.internal-file-name is assumed as the default.

PROCEDURE DIVISION FOR INPUT/OUTPUT

CLOSE Statement

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

General Format:

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

Syntax Rules:

1. The REWIND phrase must be used only for sequential files.
2. The files referenced in the CLOSE statement are not required to have the same organization or access.
3. The REEL or UNIT phrase must be used only for sequential files.
4. 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.

General Rules:

1. A file for which the EXTERNAL phrase in the SELECT clause in the Environment Division is specified is known as an external file; otherwise, that file is known as an internal file.
2. If the file is internal, the CLOSE statement can be executed only when the file is in open mode. If the file is external, the CLOSE statement can be issued to the file when it is in open mode or in closed mode. (In closed mode the statement is ignored.)
3. The FOR REMOVAL clause is used for documentation only.
4. The CLOSE statement writes out the present contents of buffers, writes system labels, and changes the state of the file to closed mode.

5. Generally, when a STOP RUN statement is executed, any files that are in open mode are closed. This default action can be modified by the appropriate parameters on the run_cobol command. Refer to the Multics COBOL Users' Guide for details.

6. A COBOL CLOSE for non-reels/units consists of two separate Multics functions:

   a. A Multics close function, which performs actions such as writing out contents of buffers, writing labels, and changing the state of the file to closed.

   b. A detach function, which breaks the relationship between the physical file and the logical file (as described in the source program).

7. The file is not detached if the NO DETACH AT CLOSE clause is specified in the I-O CONTROL paragraph. Otherwise the file is detached if it was attached by the OPEN statement.

8. Refer to "Non-Input/Output Errors" in Section 7, "Declaratives" in Section 4, and "Input/Output Status" in this section for discussions of error handling.

9. 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:

   - Non-reel/unit. A file whose input or output medium is such that the concepts of rewind and reels/units have no meaning. Non-reel/unit files are files contained in the Multics virtual memory storage system.

   - Sequential single-reel/unit. A sequential file that is entirely contained on one reel of tape.

   - Sequential multi-reel/unit. A sequential file that is contained on more than one reel of tape.

10. The results of executing each type of close CLOSE for each category of file are summarized in Table 9-3. Definitions of symbols used 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.

Table 9-3. Relationship of File Categories and CLOSE Statement Formats

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

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. 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.

B - No Rewind of Current Reel

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

CLOSE

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 standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. 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 are executed. If the file is positioned other than at its end, no ending label processing occurs.

Output Files: If label records are specified for the file, the labels are processed according to the standard label convention. The behavior of the CLOSE statement is undefined when label records are specified but not present, or when label records are not specified but are present. If label records are not specified for the file, label processing does not take place, but other closing operations are executed.

D - Reel/Unit Removal

Used for documentation only.

E - File Lock

The WITH LOCK clause ensures that this file cannot be opened again during subsequent execution. If the file is internal, it is locked for the remainder of the run-unit; if the file is external, it is locked for the remainder of the process.

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 the next data record on the new reel/unit available.

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 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.

H -

The present contents of buffers are written out, and the file state is changed to close mode and the corresponding I/O switch is detached, unless otherwise specified in the I-O-CONTROL paragraph.

I -

The current reel or analogous device is positioned at its physical beginning when the file is next closed.

X - Illegal

This is an illegal combination of a CLOSE option and a file category. The results at object time are unspecified.

11. Standard procedures involved in closing a file are described in detail in Multics Programmers' Manual, Subroutines under "tape_ansi_" and "tape_ibm_".

12. The file is automatically closed if it is in the open mode when a STOP RUN statement is executed, or if the program that initially opened it is cancelled.

13. 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.

14. If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement (except the SORT statements with the USING or GIVING phrases) can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

15. 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.

DELETE Statement

The DELETE statement logically removes a record from a relative or indexed file.

General Format:

```
DELETE file-name RECORD [ INVALID KEY imperative-statement-1 ]

[ NOT INVALID KEY imperative-statement-2 ] [ END-DELETE ]
```

Syntax Rules:

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

General Rules:

1. The associated file must be open in the I-O mode when this statement is executed. (Refer to the OPEN 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 input/output control system logically removes the record accessed by that READ statement from the file.
3. For a file in the random access or dynamic access mode, the input/output control system logically removes that record identified by the contents of the RELATIVE KEY or RECORD KEY data item associated with file-name from the file. If the file does not contain the record specified by the key, an INVALID KEY condition exists. Refer to "Error Processing" at the beginning of this section.
4. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.
5. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.
6. The current record pointer is not affected by the execution of a DELETE statement.
7. Execution of a DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated.

OPEN Statement

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 [ WITH NO REWIND ] } ...
     { OUTPUT { file-name-2 [ WITH NO REWIND ] } ...
     { I-O    { file-name-3 } ...
     { EXTEND { file-name-4 } ...                    } ...
```

Syntax Rules:

1. The NO REWIND phrase is for documentation purposes only.
2. The I-O phrase can be used only for files assigned to mass storage devices.
3. The EXTEND phrase can be used only for files with sequential organization.
4. The files referenced in the OPEN statement are not required to have the same organization or access.
5. The EXTEND phrase must not be specified for multiple file reels.

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 that references that file can be executed, either explicitly or implicitly (except for a SORT or MERGE statement with the USING or GIVING phrases).

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input/output statements. In Table 9-4, an 'X' at an intersection indicates that the specified statement can be used with the file organization and access mode given in the leftmost columns of the row and the open mode given at the top of the column.

Table 9-4. Input/Output Statements Permitted to Follow an OPEN Statement

File Organization	File Access Mode	Statement	Open Mode			
			Input	Output	Input-Output	Extend
Sequential	Sequential	READ	X		X	
		WRITE		X		X
		REWRITE			X	
Relative or Indexed	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	
		WRITE		X	X	
		REWRITE			X	
		START	X		X	
		DELETE			X	
Stream	Sequential	READ	X			
		WRITE		X		X

5. The execution of the OPEN statement does not obtain or release the first data record.

6. The file description entry for file-name-1, file-name-3, and file-name-4 must be equivalent to that entry used when this file was created.

7. 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.

8. 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 specified 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 specified conventions for output label writing.

   When label records are specified but not present, or when label records are not specified but are present, the OPEN statement behaves in accordance with standard Multics conventions.

9. If an input file is designated with the OPTIONAL phrase in its SELECT clause, the object program causes a check 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. (Refer to the READ statement.)

10. If the file is assigned to a tape device and the EXTEND phrase is not specified, execution of the OPEN statement causes the file to be positioned at its beginning.

11. 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 Format 1 READ statement for the file results in an AT END condition.

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

13. When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements that reference the file add records to the file as though the file is opened with the OUTPUT phrase.

14. 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. In the case of ANSI and IBM tapes, the user can modify the specific output mode concerning the creation by using the TAPE-OPTIONS clause in the I-O-CONTROL paragraph.

15. A file that has the EXTERNAL clause in the SELECT clause in the Environment Division is known as an external file, otherwise the file is known as an internal file.

16. Following the initial execution of an OPEN statement for an internal file, each subsequent OPEN statement for that same file must be preceded by the execution of a CLOSE statement for that file. (The OPEN and CLOSE statements must be in the same program.)

OPEN

17. If the file is an external file, the OPEN, CLOSE and other input/output statements can all be in different programs within the run-unit.

18. A COBOL OPEN consists of two separate Multics functions:

    a. A Multics ATTACH function, which establishes the relationship between the physical file and the logical file (as described in the source program).

    b. A Multics OPEN function, which performs actions such as setting pointers and changing the state of the file to open.

    A file can be attached by:

    a. Invoking the Multics io_call command, or by a CALL to the io_call subroutine. Refer to the Multics Programmers' Manual, Reference Guide, Order No. AG91 or the Multics COBOL Users' Guide, Order No. AS43.

    b. The successful execution of an OPEN statement in a COBOL program.

19. 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.

20. A PREATTACHED file must be attached before the OPEN statement is executed.

21. When a COBOL OPEN is executed, rules for external files are:

    a. Each program in the run-unit that references an external file has its own record area.

    b. Each program in the run-unit that references an external file has its own status keys.

    c. There is only one LINAGE-COUNTER for any external file.

22. Refer to "Non-Input/Output Errors" in Section 7, "Declaratives" in Section 4, and "Input/Output Status" in this section for discussion of error handling.

23. Standard procedures for handling labels upon opening the file are described in detail in Multics Programmers' Manual, Subroutines under "tape_ansi_" and "tape_ibm_".

READ

READ Statement

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 structured file.

Format 1 (Sequential Access Files):

```
READ file-name [ NEXT ] RECORD [ INTO identifier ]
  [ AT END imperative-statement-1 ]
    [ NOT AT END imperative-statement-2 ] [ END-READ ]
```

Format 2 (Random Access Files):

```
READ file-name RECORD [ INTO identifier ]
  [ KEY IS data-name ]
  [ INVALID KEY imperative-statement-1 ]
    [ NOT INVALID KEY imperative-statement-2 ] [ END-READ ]
```

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 the same storage area.
2. Format 1 must be used for all files in the sequential access mode.
3. The NEXT phrase must be specified for files in the dynamic access mode, when records are to be retrieved sequentially.
4. Format 2 is used for files in the random access mode or for files in the dynamic access mode when records are to be retrieved randomly.
5. The INVALID KEY phrase or the END phrase must be specified if no applicable USE procedure is specified for file-name.
6. Data-name must be the name of the data item specified as the record key associated with file-name.
7. Data-name can be qualified.
8. The KEY phrase can be specified only for indexed files.

9. For sequential files or stream files the NEXT phrase is optional, and has no effect on the execution of the READ statement.

General Rules:

1. The associated files must be open in the INPUT or I-O mode when this statement is executed. (Refer to the OPEN statement.)

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 was 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, possibly due to 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; and that record is then 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 with the established key of reference and that record is then 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. If the READ is successful, the logical 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 the implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are not disturbed 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 is unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record is 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 Format 1 READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

READ

9. 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 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.

10. 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.

11. 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 of reference is also undefined.

12. 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.

13. For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified retrieves the next logical record from that file as described in general rule 2.

14. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of the WRITE statements, or by execution of REWRITE statements which create such duplicate values.

15. For an indexed file if the KEY phrase is specified in a Format 2 READ statement, data-name is established as the key of reference for the 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 the file.

16. 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 the 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 the file.

17. If the RELATIVE KEY phrase is specified in the SELECT statement, 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.

READ

18. The execution of a Format 2 READ statement for a relative file 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 phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and the execution of the READ statement is unsuccessful. Refer to "Error Processing" at the beginning of this section.

19. For an indexed file, 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, until the first record having an equal value is found. The current record pointer is positioned to this record, which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

20. If the DEPENDING ON clause is specified (when this file is written) in the OCCURS clause, records can vary in length. It is the user's responsibility to determine the actual record length when the records are read.

21. Refer to the RECORD CONTAINS ... DEPENDING ON clause earlier in this section for additional information on records that can vary in size. After a READ, identifier-1 of the RECORD CONTAINS clause has its contents updated to reflect the size of the record read.

22. Records in files with stream organization are delimited by a newline character. This character is not transferred as part of the record.

23. Refer to "Non-Input/Output Errors" in Section 7, "Declaratives" in Section 4 and "Input/Output Status" in this section for discussions of error handling.

24. If the end of a reel or unit is recognized during execution of a READ statement and the logical end of 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.

25. If a file described with the OPTIONAL phrase 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 execution of the READ statement is unsuccessful. The standard end-of-file procedures are not performed. (Refer to "FILE-CONTROL Paragraph," "OPEN Statement," "USE Statement," and "Input/Output Status" in this section.) Execution of the program then proceeds as specified in general rule 12.

READ

26. Standard procedures for handling labels upon opening the file are described in Multics Subroutines and I/O Modules, Order No. AG93 under "tape_ansi_" and "tape_ibm_".

REWRITE Statement

The REWRITE statement logically replaces a record existing in a file.

General Format:

```
REWRITE record-name [ FROM identifier ] [ INVALID KEY imperative-statement-1 ]
        [ NOT INVALID KEY imperative-statement-2 ] [ END-REWRITE ]
```

Syntax Rules:

1. Record-name and identifier must not refer to the same storage area.
2. Record-name is the name of a logical record in the File Section of the Data Division and can be qualified.
3. The INVALID KEY phrase must not be specified for a REWRITE statement which references a relative file in the sequential access mode, or a sequential file.
4. The INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an appropriate USE procedure is not specified.

General Rules:

1. The file associated with record-name must be a structured file and must be open in the I-O mode when this statement is executed. (Refer to the OPEN 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 input/output control system logically replaces the record accessed by the READ statement.
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.

REWRITE

5. The execution of a REWRITE statement containing a FROM phrase is equivalent to the execution of the statement

   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. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

8. For a relative file accessed in either the random or dynamic access mode, the input/output control system logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists. The updating operation does not take place and the data in the record area is unaffected. Refer to "Error Processing" at the beginning of this section.

9. For an indexed 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.

10. For an indexed file in the random or dynamic access mode, the record to be replaced is specified by the prime record key data item.

11. The contents of alternate record key data items of the record being rewritten may differ from those in the record being replaced. The input/output control system utilizes the contents of the record key data items during the execution of the REWRITE statement in such a way that subsequent access of the record may be made based on any of those specified record keys.

12. For an indexed file, the INVALID KEY condition exists when:

    a. The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record key of the last record read from this file.

    b. The value contained in the prime record key data item does not equal that of any record stored in the file.

    c. The value contained in an alternate record key data item for which a DUPLICATES clause has not been specified, is equal to that of a record already stored in the file.

    The updating operation does not take place and the data in the record area is unaffected.

START

START Statement

The START statement provides a basis for logical positioning within a relative or 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-1 ]

[ NOT INVALID KEY imperative-statement-2 ] [ END-START ]
```

NOTE: The required relational character '=' is not underlined to avoid confusion with other symbols.

Syntax Rules:

1. File-name must be the name of a relative or indexed file with sequential or dynamic access.
2. Data-name can be qualified.
3. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.
4. If file-name is the name of a relative file, data-name, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.
5. If file-name is the name of an indexed file, and if the KEY phrase is specified, data-name can reference a data item specified as a record key associated with file-name, or it can reference any data item of category alphanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.

START

General Rules:

1. File-name must be open in the INPUT or I-O mode when the START statement is executed. (Refer to the OPEN statement.)

2. If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied.

3. For a relative file, 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.

4. For an indexed file, 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 5. If the operands are of unequal size, comparison proceeds as though the longer one is truncated on the right so that its length is equal to that of the shorter. All other nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE phrase will have no effect on the comparison (ASCII is always used).

   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.

5. For an indexed file, if the KEY phrase is specified, the comparison described in general rule 3 uses the data item referenced by data-name. 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 in subsequent Format 1 READ statements:

   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 data item specified by data-name, becomes the key of reference.

7. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

START

8. For an indexed file, if the execution of the START statement is not successful, the key of reference is undefined.

USE

USE Statement

The USE statement specifies procedures for input/output error handling that are in addition to the standard procedures provided by the input/output control system.

Format 1: (I-O exceptions)

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

Format 2: (I-O labels)

```
USE { BEFORE } STANDARD [ BEGINNING ] [ REEL ]
    { AFTER  }          [ ENDING    ] [ FILE ]
                                      [ UNIT ]

LABEL PROCEDURE ON { file-name-1 [ file-name-2 ] ... }
                   { INPUT                          }
                   { OUTPUT                         }
                   { I-O                            }
                   { EXTEND                         }
```

Syntax Rules:

1. When present, a USE statement must immediately follow a section header in the declarative 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. The same file-name may not appear in more than one USE procedure.
4. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.
5. The files explicitly or implicitly referenced in a USE statement are not all required to have the same organization or access.
6. Conditions may arise when a choice must be made between the execution of one of two USE procedures. The procedure specified by file-name shall prevail.

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 conditions, when the INVALID KEY phrase or AT END phrase, respectively, is not specified in the input/output statement.

USE

2. After the execution of a USE procedure, control is returned to the next executable statement after the I/O statement. A USE procedure is executed by the logical equivalent of a PERFORM.

3. A GO TO or a PERFORM from a declarative either to another declarative or to the nondeclarative portion of the program is allowed. An observation diagnostic is issued.

4. A GO TO from a nondeclarative to a declarative causes a fatal diagnostic to be issued. A PERFORM from a nondeclarative to a declarative is allowed; no diagnostic is issued.

WRITE Statement

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.

Format 1: (Sequential and Stream Files)

```
WRITE record-name [ FROM identifier-1 ]
      [ { BEFORE } ADVANCING { { identifier-2 } [ LINE  ] } ]
        { AFTER  }           { { integer      } [ LINES ] }
                             { { mnemonic-name } }
                             { { PAGE          } }
      [ AT { END-OF-PAGE } imperative-statement-1 ]
           { EOP         }
      [ NOT AT { END-OF-PAGE } imperative-statement-2 ] [ END WRITE ]
               { EOP         }
```

Format 2: (Relative and Indexed Files)

```
WRITE record-name [ FROM identifier-1 ]
      [ INVALID KEY imperative-statement-1 ]
      [ NOT INVALID KEY imperative-statement-2 ] [ END-WRITE ]
```

Syntax Rules:

1. Record-name and identifier-1 must not reference the same storage area.
2. When mnemonic-name is specified, the mnemonic-name must be associated with HOF or CHANNEL-n in the SPECIAL-NAMES paragraph of the Environment Division.
3. The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

4. The ADVANCING and END-OF-PAGE phrases may be used only if record-name is associated with a sequential file or stream file.

5. When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

6. If the END-OF-PAGE phrase is specified, a LINAGE clause must be specified by the FD entry of the associated file.

7. The words END-OF-PAGE and EOP are equivalent.

8. The ADVANCING mnemonic-name phrase cannot be specified if the FD entry for the associated file contains a LINAGE clause.

9. Integer or the value of the data item referenced by identifier-2 can be zero but not more than 120.

10. The INVALID KEY phrase must not be specified if record-name is associated with a sequential file.

11. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated relative or indexed file.

General Rules:

1. The associated file must be open in the OUTPUT, I-O, or EXTEND mode when the WRITE statement is executed. (Refer to the OPEN 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 was unsuccessful due to a boundary violation. 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 result of the execution of a WRITE statement containing a FROM phrase is equivalent to the execution of the statement

```
MOVE  identifier-1  TO  record-name
```

according to the rules specified for the MOVE statement without the corresponding phrase, followed by 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. (Refer to general rule 2.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. 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.

8. The execution of the WRITE statement releases a logical record to the input/output control system.

9. If the RECORD CONTAINS clause for the file contains a DEPENDING ON phrase, then the contents of data-name-1, prior to the execution of the WRITE statement, determines the number of characters written in the record.

Sequential and Stream Files:

1. Both the ADVANCING and END-OF-PAGE phrases 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 is provided 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 mnemonic-name is specified, it must be associated with the implementor-name HOF in the SPECIAL-NAMES paragraph. If specified, the representation of the printed page is advanced to the next head of form.

   d. 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.

   e. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to rules a, b, and c above.

   f. If mnemonic-name or 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, 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, repositioning to the next logical page is accomplished in accordance with an implementor-defined technique. If page has no meaning in conjunction with a specific device, advancing will be provided by the implementor to act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING&1&LINE.

2. When an attempt is made to write beyond the externally defined boundaries of a sequential or stream file, 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 indicating a boundary violation. Refer to "Input/Output Status" at the beginning of this section.

b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure is then executed.

c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the error is reported to the error_output I/O switch, and control returns to command level.

3. When a WRITE is executed, a newline character is appended to the record. This character is part of the record on the file, but is not returned via a READ. Thus, the user must not place newline characters in the record area.

4. If the logical end of the representation of the printed page is reached during execution of a WRITE statement with the END-OF-PAGE option, the imperative statement following END-OF-PAGE is executed. The logical end of a page is specified by the LINAGE clause associated with the record-name.

5. 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 simultaneously to 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, the operation proceeds as if integer-2 or data-name-2 had not been specified.

WRITE

6. After the recognition of an end-of-reel or an end-of-unit of an output file that is contained on one or more physical reels/units, 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.

Relative Files:

1. When a relative file is opened in the output mode, records can be placed into the file using one of the following procedures:

   a. If the access mode is sequential, the WRITE statement causes a record to be released to the input/output control system. The first record has a relative record number of one (1) and subsequent records released have relative record numbers of 2, 3, 4, ... If the RELATIVE KEY data item is specified in the file control entry for the associated file, the relative record number of the record just released is placed into the RELATIVE KEY data item by the input/output control system 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 input/output control system by the execution of the WRITE statement.

2. When a relative file is opened in the I-O mode and the access mode is random or dynamic, records are 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. The execution of a WRITE statement then causes the contents of the record area to be released to the input/output control system.

3. The INVALID KEY condition exists under the following circumstances:

   a. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record that already exists in the file.

   b. When an attempt is made to write beyond the externally defined boundaries of the file.

Indexed Files:

1. The value of the prime record key must be unique within the records in the file.

2. 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.

3. If the sequential access mode is specified for the file, records must be released to the input/output control system in ascending order of prime record key values.

4. If the random or dynamic access mode is specified, records can be released to the input/output control system in any program specified order.

5. The INVALID KEY condition exists under the following circumstances:

   a. When the 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.

   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.

   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.

   d. When an attempt is made to write beyond the externally defined boundaries of the file.

6. 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 input/output control system provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they were written.

Indexed and Relative Files:

1. When the INVALID KEY condition is recognized, the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the FILE STATUS data item, if any, of the associated file is set to a value indicating the cause of the condition. The execution of the program proceeds as stated under Input/Output Status at the beginning of this section.

SECTION 10

SORT-MERGE

DESCRIPTION OF SORT-MERGE

The sort-merge feature in Multics COBOL provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each record. Optionally, a user may apply some special processing to each of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the sort or after the records have been combined by the merge.

Record Ordering

The ability to arrange records into a particular order is a common requirement of a data processing user. The sort and merge features of COBOL provide facilities to assist in meeting this requirement.

While both are concerned with record ordering, the functions and capabilities of the SORT and MERGE statements are different in a number of respects. The SORT statement produces an ordered file from one or more files that may be completely unordered in the sort sequence, whereas the MERGE statement can only produce an ordered file from two or more files, each of which is already ordered in the specified sequence.

In many applications, it is necessary to apply some special processing to the contents of the sort or merge file(s) before or after sorting or merging. This special processing may consist of addition, deletion, creation, altering, editing, or other modification of the individual records in the file. The COBOL sort-merge feature allows the user to express these procedures in the COBOL language. A COBOL program may contain any number of sorts and merges, and each of them may have its own independent special procedures. The sort-merge feature automatically causes execution of these procedures in such a way that extra passes over the sort or merge files are not required.

Relationship with File Input/Output

The files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described explicitly or implicitly in the FILE-CONTROL paragraph as having sequential organization. No input/output statement may be executed for the file named in the sort-merge file description.

ENVIRONMENT DIVISION FOR SORT-MERGE - INPUT/OUTPUT SECTION

FILE-CONTROL Paragraph

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

General Format:

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

File Control Entry

The file control entry names a sort or merge file and specifies the association of the file to a storage device.

General Format:

SELECT file-name ASSIGN TO internal-file-name,

Syntax Rules:

1. Each sort or merge file described in the Data Division must be named once, and only once, as file-name in the FILE-CONTROL paragraph. Each sort or merge file specified in the file control entry must have a sort-merge file description entry in the Data Division.
2. Since file-name represents a sort or merge file, only the ASSIGN clause is permitted to follow file-name in the FILE-CONTROL paragraph.
3. Internal-file-name can contain from 1 to 16 ASCII graphic characters. The blank, minus (or hyphen), and leading or trailing period characters cannot be part of the internal-file-name.

General Rule:

1. The internal-file-name is for documentation purposes only.

I-O-CONTROL

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph for sort-merge specifies the memory area that is to be shared by different files.

General Format:

```
[ I-O-CONTROL.

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

Syntax Rules:

1. The I-O-CONTROL paragraph for sort-merge is optional.

2. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

3. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.

4. The three formats of the SAME clause (SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE 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 RECORD AREA clause.

   b. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

   c. If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and in one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE AREA clause(s). Refer to the I-O-CONTROL paragraph in Section 9, File Input/Output.

5. The files referenced in the SAME SORT AREA, SAME SORT-MERGE AREA, or SAME RECORD AREA clause are not all required to have the same organization or access.

I-O-CONTROL

General Rules:

1. 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 of 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 implicit redefinition of the area; that is, records are aligned on the leftmost character position.

2. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort file or a merge file. Files that do not represent sort or merge files may also be named in the clause.

I-O-CONTROL

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph for sort-merge specifies the memory area that is to be shared by different files.

General Format:

```
[ I-O-CONTROL.

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

Syntax Rules:

1. The I-O-CONTROL paragraph for sort-merge is optional.
2. In the SAME AREA clause, SORT and SORT-MERGE are equivalent.
3. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.
4. The three formats of the SAME clause (SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE 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 RECORD AREA clause.

   b. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

   c. If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and in one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE AREA clause(s). Refer to the I-O-CONTROL paragraph in Section 9, File Input/Output.

5. The files referenced in the SAME SORT AREA, SAME SORT-MERGE AREA, or SAME RECORD AREA clause are not all required to have the same organization or access.

General Rules:

1. 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 of 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 implicit redefinition of the area; that is, records are aligned on the leftmost character position.

2. If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort file or a merge file. Files that do not represent sort or merge files may also be named in the clause.

SD SKELETON

DATA DIVISION FOR SORT-MERGE

File Section for Sort-Merge

An SD file description is used to provide information about the size and the names of the data records associated with the file to be sorted or merged. There are no label procedures that the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements.

File Description -- Complete SD Entry Skeleton

The sort-merge file description furnishes information concerning the physical structure, identification, and record-names of the file to be sorted or merged.

General Format:

```
SD  file-name

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

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

                   [ DEPENDING ON data-name-2 ] ] .
```

Syntax Rules:

1. The level indicator SD identifies the beginning of the sort-merge file description and must precede the file-name.
2. The clauses that follow the name of the file are optional and their order of appearance is not significant.
3. One or more record description entries must follow the sort-merge file description entry. However, no input/output statements may be executed for this file.

PROCEDURE DIVISION FOR SORT-MERGE

MERGE Statement

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merged order, to an output procedure or to an output file.

General Format:

```
MERGE file name-1 ON { ASCENDING  } KEY { data-name-1 } ...
                     { DESCENDING }

                  [ ON { ASCENDING  } KEY { data-name-2 } ... ] ...
                       { DESCENDING }

        [ COLLATING SEQUENCE IS alphabet-name ]

          USING { file-name-2 } ...

        { OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
        {                                     { THRU    }                   }
        { GIVING  file-name-3                                               }
```

Syntax Rules:

1. File-name-1 must be described in a sort-merge file description entry in the Data Division.
2. Section-name-1 represents the name of an output procedure.

3. File-name-2, etc. and file-name-3 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, etc. and file-name-3 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the user's responsibility to describe the corresponding records in such a manner to cause an equal number of character positions to be allocated for the corresponding records.

4. The words THRU and THROUGH are equivalent.

5. Data-name-1, etc. and data-name-2, etc. are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. The data items identified by KEY data-names must not be variable-length data items.

   d. If file-name-1 has more than one record description, the data items identified by KEY data-names are required to be described in only one of the record descriptions.

   e. None of the data items identified by KEY data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

6. No more than one file-name from a multiple file reel can appear in the MERGE statement.

7. File-names must not be repeated within the MERGE statement.

8. MERGE statements may appear anywhere except in the declarative portion of the Procedure Division or in an input procedure or output procedure associated with a SORT or MERGE statement.

General Rules:

1. The MERGE statement merges all records contained on file-name-2, etc. The files referenced in the MERGE statement must not be open when the MERGE statement is executed. These files are automatically opened and closed by the merge operation with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file.

2. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. The leftmost data-name is the major key, the next data-name is the next most significant key, etc.

MERGE

a. When the ASCENDING phrase is specified, the merged sequence is from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the merged sequence is from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

3. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

   b. Second, the collating sequence established as the program collating sequence.

4. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any other procedure. In order to make merged records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned, one at a time in merged order, from file-name-1. The restrictions on the procedural statements within the output procedure are:

   a. The output procedure must contain no transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements that will cause an implied transfer of control to declaratives are allowed.

   b. The output procedures must contain no SORT or MERGE statements.

   c. The remainder of the Procedure Division must contain no transfers of control to points inside the output procedures; ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedures.

5. If OUTPUT PROCEDURE is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

6. Segmentation, as defined in Section 11, can be applied to programs containing the MERGE statement. However, the following restrictions apply:

   a. If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

      (1) Totally within nonindependent segments, or

      (2) Wholly contained in a single independent segment.

   b. If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

      (1) Totally within nonindependent segments, or

      (2) Wholly within the same independent segment as that MERGE statement.

7. If the GIVING phrase is specified, all the merged records in file-name-1 are automatically written on file-name-3 as the implied output procedure for this MERGE statement.

8. In the case of an equal compare (according to the rules for comparison of operands in a relation condition) on the contents of the data items identified by all the KEY data-names between records from two or more input files (file-name-2, etc.), the records are written on file-name-3 or returned to the output procedure, depending on the phrase specified, in the order that the associated input files are specified in the MERGE statement.

9. The results of the merge operation are predictable only when the records in the files referenced by file-name-2, etc., are ordered as described in the ASCENDING or DESCENDING KEY phrase associated with the MERGE statement.

RELEASE Statement

The RELEASE statement transfers records to the initial phase of a sort operation.

General Format:

RELEASE record-name [FROM identifier]

Syntax Rules:

1. A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement for a file whose sort-merge file description entry contains record-name. Refer to the SORT statement.
2. Record-name must be the name of a logical record in the associated sort-merge file description entry and may be qualified.
3. Record-name and identifier must not refer to the same storage area.

General Rules:

1. The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.
2. If the FROM phrase is used, the contents of the identifier data area are moved to record-name; then the contents of record-name are released to the sort file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The information in the record area is no longer available, but the information in the data area associated with identifier is available.
3. After the execution of the RELEASE statement, the logical record is no longer available in the record area unless the associated sort-merge 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 sort-merge file, as well as to the file associated with record-name. When control passes from the input procedure, the file consists of all those records that were placed in it by the execution of RELEASE statements.

RETURN

RETURN Statement

The RETURN statement is used to obtain either the sorted records from the final phase of a sort operation or merged records during a merge operation.

General Format:

```
RETURN file-name RECORD [ INTO identifier ] AT END imperative-statement-1
    [ NOT AT END imperative statement-2 ] [ END-RETURN ]
```

Syntax Rules:

1. File-name must be described by a sort-merge file description entry in the Data Division.
2. A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for file-name.
3. 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 the same storage area.

General Rules:

1. 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 RETURN statement.
2. The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT or MERGE statement, to be made available for processing in the record areas associated with the sort or merge file.
3. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied move does not occur if an AT END condition exists. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.
4. When the INTO phrase is used, the data is available in both the input record area and in the data area associated with identifier.

5. If no next logical record exists for the file when a RETURN statement is executed, the AT END condition occurs. The contents of the record areas associated with the file when the AT END condition occurs are undefined. After the execution of the imperative statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

SORT Statement

The SORT statement creates a sort file by executing input procedures or by transferring records from another file; sorts the records in the sort file on a set of specified keys; and, in the final phase of the sort operation, makes available each record from the sort file, in sorted order, to some output procedures or to an output file.

General Format:

```
SORT file-name-1 { ON { ASCENDING  } KEY { data-name-1 } ... } ...
                     { DESCENDING }

  [ COLLATING SEQUENCE IS alphabet-name ]

  { INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
  {                                     { THRU    }                 }
  { USING { file-name-2 } ...                                       }

  { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
  {                                      { THRU    }                  }
  { GIVING file-name-3                                                }
```

Syntax Rules:

1. File-name-1 must be described in a sort-merge file description entry in the Data Division.
2. Section-name-1 represents the name of an input procedure. Section-name-3 represents the name of an output procedure.

SORT

3. File-name-2, etc. and file-name-3 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, etc. and file-name-3 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the responsibility of the user to describe the corresponding records in such a manner to cause equal amounts of character positions to be allocated for the corresponding records.

4. Data-name-1, etc. are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

   d. None of the data items identified by KEY data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

5. The words THRU and THROUGH are equivalent.

6. SORT statements may appear anywhere except in the declarative portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.

7. No more than one file-name from a multiple file reel can appear in the SORT statement.

General Rules:

1. The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, the leftmost data-name is the major key, the next data-name is the next most significant key, etc.

   a. When the ASCENDING phrase is specified, the sorted sequence is from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the sorted sequence is from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

2. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

   a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

   b. Second, the collating sequence established as the program collating sequence.

3. The input procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. In order to transfer records to the file referenced by file-name-1, the input procedure must include the execution of at least one RELEASE statement. Control must not be passed to the input procedure except when a related SORT statement is being executed. The input procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

   a. The input procedure must contain no SORT or MERGE statements.

   b. The input procedure must contain no explicit transfers of control to points outside the input procedure; ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

   c. The remainder of the Procedure Division must contain no transfers of control to points inside the input procedure; ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

4. If INPUT PROCEDURE is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

5. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form part of any input procedure. In order to make sorted records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned, one at a time in sorted order, from the sort file. The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must contain no SORT or MERGE statements.

SORT

b. The output procedure must contain no explicit transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

c. The remainder of the Procedure Division must contain no transfers of control to points inside the output procedure; ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

6. If OUTPUT PROCEDURE is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

7. Segmentation, as defined in Section 11, can be applied to programs containing the SORT statement. However, the following restrictions apply:

   a. If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

      (1) Totally within nonindependent segments, or

      (2) Wholly contained in a single independent segment.

   b. If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

      (1) Totally within nonindependent segments, or

      (2) Wholly within the same independent segment as that SORT statement.

8. If the USING phrase is specified, all the records in file-name-2, etc. are transferred automatically to file-name-1. When the SORT statement is executed, file-name-2, etc. must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2, etc. These implicit functions are performed so that any associated USE procedures are executed. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2, etc. to the file area for file-name-1 and releasing records to the initial phase of the sort operation.

SORT

9. If the GIVING phrase is specified, all the sorted records in file-name-1 are automatically written on file-name-3 as the implied output procedure for this SORT statement. When the SORT statement is executed, file-name-3 must not be open. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of file-name-3. These implicit functions are performed so that any associated USE procedures are executed. The terminating function is performed as if a CLOSE statement, without optional phrases, had been executed for the file. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and moving the records from the file area for file name-1 to the file area for file-name-3. The records written onto file-name-3 are of a fixed size, equal to the size of the largest record described in file-name-1.

SECTION 11

SEGMENTATION

DESCRIPTION OF SEGMENTATION

ANSI COBOL-74 describes a segmentation facility that contains physical and logical aspects. The physical aspects are memory management by an overlay technique. Overlays are not meaningful in the Multics virtual memory system. Thus Multics COBOL does not support the physical aspects of segmentation. This has no effect on the logic of the object program. The logical aspects of segmentation affect statements such as ALTER, PERFORM, and SORT. The logical aspects of segmentation are supported by Multics COBOL and are described in the remainder of this section.

Unless otherwise indicated, the term segment is used in the COBOL sense. The entire Procedure Division is one Multics segment.

Program Segments

The Procedure Division for a source program may be written as a group of sections. Each section belongs to either the fixed portion of the object program or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

Fixed Portion

The fixed portion of the program is always in its last used state when called for by the program.

Independent Segments

An independent segment is in its initial state whenever control is transferred (either explicitly or implicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

1. Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

2. Control is transferred to that segment as the result of the implicit transfer of control between a SORT statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

3. Control is transferred explicitly to that segment from a segment with a different segment-number (with the EXIT PROGRAM exception noted below).

On subsequent transfers of control to the segment, an independent segment is in its last used state when:

1. Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in paragraphs 1 and 2 above).

2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

Refer to "Transfers of Control" in Section 2 for additional information.

Segmentation Control

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

STRUCTURE OF PROGRAM SEGMENTS

Segment-Numbers

Section classification is accomplished by means of a system of segment-numbers. The 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 0 through 99.
2. If the segment-number is omitted from the section header, the segment number is assumed to be zero.
3. Sections in the declarative portion must contain segment-numbers less than 50.

General Rules:

1. All sections having the same segment-number constitute a program segment. Sections having the same segment-numbers are not required to be physically contiguous in the source program.
2. Segments with segment-number 0 through 49 belong to the fixed portion of the object program.
3. Segments with segment-number 50 through 99 are independent segments.

SEGMENT-LIMIT

SEGMENT-LIMIT Clause

General Format:

[SEGMENT-LIMIT IS segment-number] .

Syntax Rules:

1. Segment number must be an integer ranging in value from 1 through 49.

General Rules:

1. The SEGMENT-LIMIT clause is for documentation purposes only.

RESTRICTIONS ON PROGRAM FLOW

When the segmentation feature is used, the following restrictions are placed on the ALTER, PERFORM, MERGE, and SORT statements.

ALTER Statement in Segmented Programs

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

All other uses of the ALTER statement are valid.

PERFORM Statement in Segmented Programs

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:

1. Sections and/or paragraphs wholly contained in one or more nonindependent segments.
2. Sections and/or paragraphs wholly contained in a single independent segment.

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:

1. Sections and/or paragraphs wholly contained in one or more nonindependent segments.
2. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

SORT Statement In Segmented Programs

If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

1. Totally within nonindependent segments, or
2. Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

1. Totally within nonindependent segments, or
2. Wholly within the same independent segment as that SORT statement.

RESTRICTIONS ON PROGRAM FLOW

When the segmentation feature is used, the following restrictions are placed on the ALTER, PERFORM, MERGE, and SORT statements.

ALTER Statement in Segmented Programs

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

All other uses of the ALTER statement are valid.

PERFORM Statement in Segmented Programs

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:

1. Sections and/or paragraphs wholly contained in one or more nonindependent segments.
2. Sections and/or paragraphs wholly contained in a single independent segment.

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:

1. Sections and/or paragraphs wholly contained in one or more nonindependent segments.
2. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

SORT Statement In Segmented Programs

If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

1. Totally within nonindependent segments, or
2. Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

1. Totally within nonindependent segments, or
2. Wholly within the same independent segment as that SORT statement.

SECTION 12

LIBRARY FACILITY

The library feature provides the capability for specifying text that is to be copied from a library.

The COBOL source text manipulation facilities are the COPY statement and the REPLACE statement. Each of these facilities can function either independently of the other or in conjunction with the other to provide an extensive capability to insert and replace source program text as part of the compilation of the source program.

COBOL libraries contain texts which are available to the compiler at compile time. The effect of the interpretation of the COPY statement is to generate, from a library text, text which is treated by the compiler as part of the source program. COBOL library text is placed in a COBOL library as a function independent of the COBOL source program. More than one library may be available when the program is compiled.

Similarly, COBOL source programs can be written in a programmer-defined notation which, at compile time, can be expanded into syntactically correct phrases, clauses, and statements. The effect of the interpretation of the REPLACE statement is to substitute new text for text appearing in the source program and have the substituted text treated by the compiler as part of the source program.

COPY STATEMENT

The COPY statement is used in the COBOL source program to incorporate text from a library into the source program.

General Format:

```
COPY text-name [ { OF } library-name ]
               [ { IN }              ]

     [ REPLACING { ==pseudo-text-1==  } BY { ==pseudo-text-2==  } } ... ]
                 { identifier-1       }    { identifier-2       }
                 { literal-1          }    { literal-2          }
                 { word-1             }    { word-2             }
```

Syntax Rules:

1. If two source texts have the same text-name then they must be placed in different libraries and text-name must be qualified by library-name. For information on usage, refer to the Multics COBOL Users' Guide, Order No. AS43.

2. The COPY statement must be preceded by a space and terminated by the separator period.

3. Pseudo-text-1 must contain one or more text words. It must not be null, nor may it consist solely of the character space(s), nor may it consist solely of comment lines.

4. Pseudo-text-2 may contain zero, one, or more text-words.

5. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued. However, both characters of a pseudo-text delimiter must be on the same line.

6. Word-1 or word-2 may be any single COBOL word except COPY.

7. A COPY statement can occur in the source program anywhere a character-string or a separator can occur except that a COPY statement must not occur within a COPY statement.

8. Pseudo-text-1 must not consist entirely of a separator comma or a separator semicolon.

9. If the word COPY appears in a comment-entry or in a place where a comment entry may appear, it is considered part of the comment-entry.

COPY

General Rules:

1. The compilation of a source program containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.

2. The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive.

   If additional lines are introduced into the source program as a result of the processing of COPY statements, the Indicator Area of the introduced line contains the same character as the line on which the text being replaced begins, unless that line contains a hyphen, in which case the introduced line contains a space.

3. 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 pseudo-text-1, identifier-1, word-1, and literal-1 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2, or literal-2.

4. For purposes of matching, identifier-1, word-1, and literal-1 are treated as pseudo-text containing only identifier-1, word-1, or literal-1, respectively.

5. The comparison operation to determine text replacement occurs in the following manner:

   a. 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 pseudo-text-1, identifier-1, 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 an equivalent number of contiguous library text-words.

   b. Pseudo-text-1, identifier-1, word-1, or literal-1 match the library text if, and only if, the ordered sequence of text-words that forms pseudo-text-1, identifier-1, word-1, or literal-1 is equal, character for character, to the ordered sequence of library text-words. For purposes of matching, each occurrence of a separator comma or semicolon in pseudo-text-1 or in the library text is considered to be a single space except when pseudo-text-1 consists solely of either a separator comma or semicolon, in which case it participates in the match as a text-word. Each sequence of one or more space separators is considered to be a single space.

   c. If no match occurs, the comparison is repeated with each next successive pseudo-text-1, identifier-1, 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.

   d. When all the REPLACING phrase 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 pseudo-text-1, identifier-1, word-1, or literal-1 specified in the REPLACING phrase.

COPY

e. Whenever a match occurs between pseudo-text-1, identifier-1, word-1, or literal-1 and the library text, the corresponding pseudo-text-2, identifier-2, 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 pseudo-text-1, identifier-1, word-1, or literal-1 specified in the REPLACING phrase.

f. 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.

6. Comment lines or blank lines occurring in the library text and in pseudo-text-1 are ignored for purposes of matching; and the sequence of text-words in the library text, if any, and in pseudo-text-1 is determined by the rules for Reference Format. Comment lines or blank lines appearing in pseudo-text-2 are copied into the resultant program unchanged whenever pseudo-text-2 is placed into the source program as a result of text replacement. Comment lines or blank lines appearing in library text are copied into the resultant source program unchanged with the following exception: a comment line or blank line in library text is not copied if that comment line or blank line appears within the sequence of text-words that match pseudo-text-1.

7. Debugging lines are permitted within library text and pseudo-text-2. Debugging lines are not permitted within pseudo-text-1; text-words within a debugging line participate in the matching rules as if the 'D' did not appear in the indicator area. If a COPY statement is specified on a debugging line, then the text that is the result of the processing of the COPY statement will appear as though it were specified on debugging lines with the following exception: comment lines in library text will appear as comment lines in the resultant source program.

8. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

9. The syntactic correctness of the library text cannot be independently determined. The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed.

10. Library text must conform to the rules for the COBOL reference format in Section 4.

11. Each word copied from the library, but not replaced, is copied into the same area of the resultant program as copied in the library. Each word in pseudo-text-2 that is to be placed in the resultant program is placed in the same area of the resultant program as in pseudo-text-2.

12. For purposes of compilation, text-words after replacement are placed in the source program according to the rules for reference format.

13. The text-name refers to a Multics segment that has the name of the form:

text-name.incl.cobol

If the text-name in the COPY statement is not qualified by a library-name then the library text is located by using the Multics search facility. If the text-name in the COPY statement is qualified by a library-name then library-name specifies a Multics directory which contains the segment:

text_name.incl.cobol

(For further details, refer to the Multics COBOL Users' Guide, Order No. AS43 and to the MPM Commands and Active Functions, Order No. AG92 for a description of the Multics search facility.)

REPLACE STATEMENT

The REPLACE statement is used to replace source program text.

General Format:

Format 1:

```
REPLACE { ==pseudo-text-1== BY ==pseudo-text-2==} ...
```

Format 2:

```
REPLACE OFF
```

Syntax Rules:

1. A REPLACE statement may occur anywhere in the source program where a character-string may occur. It must be preceded by a separator period except when it is the first statement in a separately compiled program.
2. A REPLACE statement must be terminated by a separator period.
3. Pseudo-text-1 must contain one or more text-words.
4. Pseudo-text-2 may contain zero, one, or more text-words.
5. Character-strings within pseudo-text-1 and pseudo-text-2 may be continued.
6. Pseudo-text-1 must not consist entirely of a separator comma or a separator semicolon.
7. If the word REPLACE appears in a comment-entry or in a place where a comment entry may appear, it is considered part of the comment-entry.

REPLACE

General Rules:

1. The Format 1 REPLACE statement specifies the text of the source program to be replaced by the corresponding text. Each matched occurrence of pseudo-text-1 in the source program is replaced by the corresponding pseudo-text-2.

2. The Format 2 REPLACE statement specifies that any text replacement currently in effect is discontinued.

3. A given occurrence of the REPLACE statement is in effect from the point at which it is specified until the next occurrence of the statement or the end of the separately compiled program respectively.

REPLACE

4. Any REPLACE statements contained in a source program are processed after the processing of any COPY statements contained in a source program.

5. The text produced as a result of the processing of a REPLACE statement must not contain a REPLACE statement or a COPY statement.

6. The comparison operation to determine text replacement occurs in the following manner:

   a. Starting with the leftmost source program text-word and the first pseudo-text-1, pseudo-text-1 is compared to an equivalent number of contiguous source program text-words.

   b. Pseudo-text-1 matches the source program text if, and only if, the ordered sequence of text-words that forms pseudo-text-1 is equal, character for character, to the ordered sequence of source program text-words. For purposes of matching each occurrence of a separator comma or semicolon in pseudo-text-1 or in the source program text is considered to be a single space except when pseudo-text-1 consists solely of either a separator comma or semicolon, in which case it participates in the match as a text-word. Each sequence of one or more space separators is considered to be a single space.

   c. If no match occurs, the comparison is repeated with each next successive occurrence of pseudo-text-1, until either a match is found or there is no next successive occurrence of pseudo-text-1.

   d. When all occurrences of pseudo-text-1 have been compared and no match has occurred, the next successive source program text-word is then considered as the leftmost source program text-word, and the comparison cycle starts again with the first occurrence of pseudo-text-1.

   e. Whenever a match occurs between pseudo-text-1 and the source program text, the corresponding pseudo-text-2 replaces the matched text in the source program. The source program text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost source program text-word. The comparison cycle starts again with the first occurrence of pseudo-text-1.

   f. The comparison operation continues until the rightmost text-word in the source program text which is within the scope of the REPLACE statement has either participated in a match or been considered as a leftmost source program text-word and participated in a complete comparison cycle.

7. Comment lines or blank lines occurring in the source program text and in pseudo-text-1 are ignored for purposes of matching, and the sequence of text-words in the source program text and in pseudo-text-1 is determined by the rules for Reference Format. Comment lines or blank lines in pseudo-text-2 are placed into the resultant program unchanged whenever pseudo-text-2 is placed into the source program as a result of text replacement. A comment line or blank line in source program text is not replaced if that comment line or blank line appears within the sequence of text-words that match pseudo-text-1.

8. Debugging lines are permitted in pseudo-text-2; they are not permitted in pseudo-text-1. Text-words within a debugging line participate in the matching rules as if the 'D' did not appear in the Indicator Area. If a REPLACE statement is specified on a debugging line, the text that is the result of the processing of the REPLACE statement appears as though it were specified on debugging lines.

9. Except for COPY and REPLACE statements, the syntactic correctness of the source program text cannot be determined until after all COPY and REPLACE statements have been completely processed.

10. Text-words inserted into the source program as a result of processing a REPLACE statement are placed in the source program according to the rules for Reference Format. Each word in pseudo-text-2 inserted into the source program is placed in the same area of the source program as it appears in pseudo-text-2.

11. If additional lines are introduced into the source program as a result of the processing of REPLACE statements, the Indicator Area of the introduced lines contains the same character as the line on which the text being replaced begins, unless that line contains a hyphen, in which case the introduced line contains a space.

SECTION 13

DEBUG FACILITY

DESCRIPTION OF THE DEBUG FACILITY

The debugging facility described in this section is defined by the ANS COBOL Standard. Changes must be made to a source program if the facility is to be used. Multics also offers powerful symbolic debugging facilities (probe and debug) which can be applied to a program without making source changes. These facilities are described in Section 6 of the Multics COBOL User's Guide, Order Number AS43 and in the Multics Programmers Manual - Commands and Active Functions, Order Number AG92.

The debug facility assists in error detection by:

1. Monitoring transfers of control to user-selected procedures during program execution.
2. Monitoring values of user selected data items during program execution.

The user-supplied statements required to accomplish such monitoring are included in the source program and can be compiled (or not) depending upon the presence or absence of the WITH DEBUGGING MODE clause in the source program. After the user statements have been compiled into the program, they may be executed (or ignored) at object program execution according to the setting of a run-time switch. The decisions concerning what to monitor and what information to display on the output device are explicitly at the discretion of the user. The main purpose of the COBOL debug feature is to provide convenient access to such information.

The elements of the COBOL language that support the debug facility are:

- The special register DEBUG-ITEM
- A compile time switch; WITH DEUGGING MODE
- An object time switch
- The USE FOR DEBUGGING statement
- Debugging lines

Special Register DEBUG-ITEM

The reserved word DEBUG-ITEM is the name of a special register generated automatically by compiler coding that supports the debug facility. Only one DEBUG-ITEM is allocated for each program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

Compile-Time Switch

The WITH DEBUGGING MODE clause, written as part of the SOURCE-COMPUTER paragraph, serves as a compile time switch governing the debugging statements written in the program. When this clause is specified in a program, all debugging sections and all debugging lines are compiled as described in this section. When this clause is omitted, all debugging lines and all debugging sections are compiled as if they were comment lines.

Object-Time Switch

An object-time switch dynamically activates the debug coding inserted by the compiler. This switch cannot be addressed in the program; it is set by using the -debug option in the run_cobol command which executes the program. If the switch is "ON", all the effects of the debugging language written in the source program are permitted. If the switch is "OFF", all the effects described in the USE FOR DEBUGGING statement are inhibited. The recompilation of the source program is not required to provide or remove this facility.

The object-time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause is not specified in the source program during program compilation.

USE FOR DEBUGGING Statement

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

Debugging Lines

A debugging line is any line with a "D" in the indicator area (character position 7) of the line. Refer to "Debugging Lines" later in this section.

WITH DEBUGGING MODE

ENVIRONMENT DIVISION FOR DEBUGGING

WITH DEBUGGING MODE Clause

The WITH DEBUGGING MODE clause serves as a compile-time switch by indicating that all debugging sections and all debugging lines are to be compiled If this clause is not specified, all debugging lines and sections are compiled as if they were comment lines.

General Format:

SOURCE-COMPUTER [HIS SERIES-60] MULTICS

WITH DEBUGGING MODE.

General Rules:

1. If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph in the Configuration Section of a program, all USE FOR DEBUGGING statements and all debugging lines are compiled.

2. If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph in the Configuration Section of a program, any USE FOR DEBUGGING statements and all associated debugging sections, and any debugging lines are compiled as if they were comment lines.

PROCEDURE DIVISION FOR DEBUGGING

USE FOR DEBUGGING Statement

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

USE FOR DEBUGGING

General Format:

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

USE FOR DEBUGGING ON { [ cd-name-1
                       [ ALL REFERENCES OF ] identifier-1
                         file-name-1
                         procedure-name-1
                         ALL PROCEDURES                    } ... .
```

Syntax Rules:

1. Debugging section(s), if specified, must appear together immediately after the declarative section header.
2. Except in the USE FOR DEBUGGING statement itself, there must be no reference to any nondeclarative procedure within the debugging section.
3. Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections.
4. Except for the USE FOR DEBUGGING statement itself, statements appearing within a given debugging section may reference procedure-names defined within a different USE procedure only with a PERFORM statement.
5. Procedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.
6. Any given identifier, cd-name, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.
7. The ALL PROCEDURES phrase can appear only once in a program.
8. When the ALL PROCEDURES phrase is specified, procedure-name-1, etc. must not be specified in any USE FOR DEBUGGING statement.
9. If the data description entry of the data item referenced by identifier-1, etc. contains an OCCURS clause or is subordinate to a data description entry that contains an OCCURS clause, identifier-1, etc. must be specified without the subscripting or indexing normally required.
10. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

General Rules:

1. In the following General Rules, all references to cd-name-1, identifier-1, file-name-1, and procedure-name-1 apply equally to all recurrances.
2. A reference to file-name-1, identifier-1, procedure-name-1, or cd-name-1 as a qualifier does not constitute reference to that item for the debugging described in the General Rules below. For purposes of debugging, a reference to identifier-1 with or without its associated subscripting or reference modifiers constitutes an explicit reference to identifier-1

USE FOR DEBUGGING

and causes the associated debugging section to be executed in accordance with General Rules 9 and 10.

3. Where more than one sequence of qualifiers may be used to reference a given data item or procedure-name, any form of explicit reference to such data item or procedure-name specified in the USE FOR DEBUGGING statement causes the execution of the debugging section.

4. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

5. When file name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. After the execution of any OPEN or CLOSE statement that references file-name-1.

   b. After the execution of any READ statement (after any other specified USE procedure) not resulting in the execution of an associated AT END or INVALID KEY imperative statement.

   c. After the execution of any DELETE or START statement that references file-name-1.

6. When procedure-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. Immediately before each execution of the named procedure.

   b. Immediately after the execution of an ALTER statement that references procedure-name-1.

7. The ALL PROCEDURES phrase causes the effects described in General Rule 6 to occur for every procedure-name in the program, except those appearing within a debugging section.

8. When the ALL REFERENCES OF identifier-1 phrase is specified, that debugging section is executed for every statement that explicitly references identifier-1 at each of the following times:

   a. In the case of a WRITE or REWRITE statement, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

   b. In the case of a GO TO statement with a DEPENDING ON phrase, immediately before control is transferred and prior to the execution of any debugging section associated with the procedure-name to which control is to be transferred.

   c. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification, or evaluation of the contents of the data item referenced by identifier-1.

   d. In the case of any other COBOL statement, immediately after execution of that statement.

   If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

9. When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed at each of the following times:

USE FOR DEBUGGING

a. In the case of a WRITE or REWRITE statement that explicitly references identifier-1, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

b. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification, or evaluation of the contents of the data item referenced by identifier-1.

c. Immediately after the execution of any other COBOL statement that explicitly references and causes the contents of the data item referenced by identifier-1 to be changed.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

10. The associated debugging section is not executed for a specific operand more than once as a result of the execution of a single statement, regardless of the number of times that operand is explicitly specified. In the case of a PERFORM statement that causes iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

    Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

11. When cd-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

    a. After the execution of any ENABLE, DISABLE, and SEND statement that references cd-name-1.

    b. After the execution of a RECEIVE statement referencing cd-name-1 that does not result in the execution of the NO DATA imperative statement.

    c. After the execution of an ACCEPT MESSAGE COUNT statement that references cd-name-1.

12. The special register DEBUG-ITEM is associated with each execution of a debugging section. DEBUG-ITEM provides information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```
01  DEBUG-ITEM.
    02  DEBUG-LINE      PICTURE IS X(6).
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-NAME      PICTURE IS X(30).
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-1     PICTURE IS S9999
                           SIGN IS LEADING SEPARATE CHARACTER.
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-2     PICTURE IS S9999
                           SIGN IS LEADING SEPARATE CHARACTER.
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-3     PICTURE IS S9999
                           SIGN IS LEADING SEPARATE CHARACTER.
    02  FILLER          PICTURE IS X  VALUE SPACE.
    02  DEBUG-CONTENTS  PICTURE IS X(n).
```

USE FOR DEBUGGING

13. Prior to each execution of a debugging section, the content of the data item referenced by DEBUG-ITEM is space-filled. The contents of data items subordinate to DEBUG-ITEM are then updated, according to the following General Rules, immediately before control is passed to that debugging section. The content of any data item not specified in the following General Rules remains spaces.

    Updating is accomplished in accordance with the rules for the MOVE statement, the sole exception being the move to DEBUG-CONTENTS when the move is treated exactly as if it was an alphanumeric-to- alphanumeric elementary move with no conversion of data from one form of internal representation to another.

14. The content of DEBUG-LINE is the internal line number assigned to the source statement (as indicated on the source listing).

15. DEBUG-NAME contains the first 30 characters of the name that caused the debugging section to be executed.

    All qualifiers of the name are separated in DEBUG-NAME by the word "IN" or "OF". Subscripts/indexes, if any, are not entered into DEBUG-NAME.

16. If the reference to a data item that causes the debugging section to be executed is subscripted or indexed, the occurrence number of each level is entered in DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3, respectively, as necessary.

17. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following General Rules.

18. If the first execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the first statement of that procedure.

    b. DEBUG-NAME contains the name of that procedure.

    c. DEBUG-CONTENTS contains "START PROGRAM".

19. If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.

20. If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

21. If a reference to procedure-name-1 in the INPUT or OUTPUT phrase of a SORT or MERGE statement causes the debugging section to be executed, the following conditions exist:

USE FOR DEBUGGING

a. DEBUG-LINE identifies the SORT or MERGE statement that references procedure-name-1.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains:

- If the reference to procedure-name-1 is in the INPUT phrase of a SORT statement, "SORT INPUT".
- If the reference to procedure name-1 is in the OUTPUT phrase of a SORT statement, "SORT OUTPUT".
- If the reference to procedure-name-1 is in the OUTPUT phrase of a MERGE statement, "MERGE OUTPUT".

22. If the transfer of control from the control mechanism associated with a PERFORM statement caused the debugging section associated with procedure name-1 to be executed, the following conditions exist:

a. DEBUG-LINE identifies the PERFORM statement that references procedure-name-1.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains "PERFORM LOOP".

23. If procedure-name-1 is a USE procedure that is to be executed, the following conditions exist:

a. DEBUG-LINE identifies the statement that causes execution of the USE procedure.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains "USE PROCEDURE".

24. If an implicit transfer of control from the previous sequential paragraph to procedure-name-1 causes the debugging section to be executed, the following conditions exist:

a. DEBUG-LINE identifies the previous statement.

b. DEBUG-NAME contains procedure name-1.

c. DEBUG-CONTENTS contains "FALL THROUGH".

25. If references to file name-1, cd-name-1 cause the debugging section to be executed, then:

a. DEBUG-LINE identifies the source statement that references file-name-1, cd-name-1.

b. DEBUG-NAME contains the name of file-name-1, cd-name-1.

c. For READ, DEBUG-CONTENTS contains the entire record read.

d. For all other references to file-name-1, DEBUG-CONTENTS contains spaces.

e. For any reference to cd-name-1, DEBUG-CONTENTS contains the contents of the area associated with the cd-name.

USE FOR DEBUGGING

26. If a reference to identifier-1 causes the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references identifier-1.

    b. DEBUG-NAME contains the name of identifier 1.

    c. DEBUG-CONTENTS contains the contents of the data item referenced by identifier-1 when control passes to the debugging section (see General Rules 6 and 7).

Debugging Lines

A debugging line is any line with a "D" (or a "d") in the indicator area of the line. Any debugging line that consists solely of spaces from margin A to margin R is considered to be the same as a blank line.

The contents of a debugging line must be such that a syntactically correct program is formed with (or without) the debugging lines being considered as comment lines.

A debugging line is considered to have all the characteristics of a comment line if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain a "D" in the indicator area, and character-strings may not be broken across two lines.

A debugging line is only permitted in the program following the OBJECT-COMPUTER paragraph.

SECTION 14

INTERPROGRAM COMMUNICATION

DESCRIPTION OF INTERPROGRAM COMMUNICATION

Interprogram communication is a facility through which a program can communicate with one or more programs. The communication is provided by the ability to transfer control from one program to another within a run-unit and the ability for both programs to have access to the same data items. Control can be transferred to one or more programs whose names are not known at program compilation time.

PROGRAM MODULARITY

Complex data processing problems are frequently solved by using separately compiled but logically coordinated programs which at program execution, form logical and physical subdivisions of a single run-unit. This method allows a large problem to be divided into smaller, more manageable segments that can be programmed and debugged independently. At program execution, control is transferred from program to program using the CALL and EXIT PROGRAM statements.

In COBOL terminology, a program is either a source program or an object program, depending on the context. A source program is a syntactically correct set of COBOL statements, and an object program is the set of instructions, constants, and other machine-oriented data resulting from the operation of a compiler on a source program. A run-unit is the total machine language necessary to solve a data processing problem; it includes one or more object programs as defined above, and it may include machine language from source languages other than COBOL.

When the statement of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other. This communication can take two forms:&&transfer of control and reference to common data.

Transfer of Control (CALL)

The CALL statement provides the means whereby control can be passed from one program to another within a run-unit. A program that is activated by a CALL statement can also contain CALL statements. However, the results are unpredictable where circularity of control is initiated; e.g., where program A calls program B, and then program B calls program A or another program that calls program A.

When control is passed to a called program, execution proceeds in the normal way from procedure statement to procedure statement beginning with the first nondeclarative statement. If control reaches a STOP RUN statement, this signals the logical end of the run-unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the point immediately following the CALL statement in the calling program. Stated briefly, the EXIT PROGRAM statement terminates only the program in which it occurs, and the STOP RUN statement terminates the entire run-unit.

If the called program is not a COBOL program, the termination of the run-unit or the return to the calling program must be programmed in accordance with the language rules of the called program.

Interprogram Data Storage

Program interaction requires that both programs have access to the same data items. In the calling program, the common data items are described along with all other data items in the File Section, Working-Storage Section, or Linkage Section. At object program execution, memory is allocated for the entire Data Division. In the called program, common data items are described in the Linkage Section; memory space is not allocated for this section at object program execution.

Communication between the called program and the common data items stored in the calling program is effected through USING phrases contained in both programs. The USING phrase in the calling program is contained in the CALL statement and the operands are a list of common data identifiers described in its Data Division. The USING phrase in the called program follows the Procedure Division header and the operands are a list of common data identifiers described in its Linkage Section. The identifiers specified by the USING phrase of the CALL statement indicate those data items available to a calling program that may be referred to in the called program. The sequence of appearance of the identifiers in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is significant. Corresponding identifiers refer to a single set of data which is available to the calling program. The correspondence is positional, and not by name. While the called program is being executed, every reference to an operand whose identifier appears in the called program's USING phrase is treated as if it is a reference to the corresponding operand in the USING phrase of the active CALL statement.

After control leaves a called program, its state is maintained. Therefore, initialization of the program in case of repetitive calls is not necessary.

INTERPROGRAM COMMUNICATION DATA DIVISION - LINKAGE SECTION

The Linkage Section in a program is meaningful only if the object program is to function under the control of a CALL statement and the CALL statement in the calling program contains a USING phrase.

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 programs. No space is allocated in the program for data items referenced by data-names 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 indexes.

Data items defined in the Linkage Section of the called program can be referenced within the Procedure Division of the called program only if they are specified as operands of the USING phrase of the Procedure Division header (see Section 3) or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

The structure of the Linkage Section is the same as that described for the Working-Storage Section in the Nucleus, beginning with a section header, followed by data description entries for noncontiguous data items and/or record description entries.

Each Linkage Section record-name and noncontiguous item name must be unique within the called program, since it cannot be qualified.

Of those items defined in the Linkage Section, only data-name-1, etc. in the USING phrase of the Procedure Division header, data items subordinate to these data-names, and condition-names or index-names associated with such data-names or subordinate data items can be referenced in the Procedure Division.

Noncontiguous Linkage Storage

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these data items is defined in a separate data description entry that begins with the special level-number 77.

The following data clauses are required in each data description entry:

1. Level-number 77
2. Data-name
3. The PICTURE clause or the USAGE IS INDEX clause

Other data description clauses are optional and can be used to complete the description of the item if necessary.

Linkage Records

Data elements in the Linkage Section that bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause used in an input or output record description can be used in a Linkage Section.

Linkage Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

USING

PROCEDURE DIVISION FOR INTERPROGRAM COMMUNICATION

Procedure Division Header

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

PROCEDURE DIVISION [USING { data-name-1 } ...] .

The USING phrase must be present only if the object program is to function under control of a CALL statement and the CALL statement in the calling program contains a USING phrase.

Each operand 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 the data-names mentioned the Procedure Division header in the called program, and the data-names mentioned 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 must define an equal number of character positions; however, they need not be the same name. A data-name must not appear more than once in the USING phrase in the Procedure Division header of the called program; however, a given data-name can appear more than once in the same USING phrase of a CALL statement.

If the USING phrase is specified, the INITIAL clause must not be present in any communication description (CD) entry.

CALL Statement

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

General Format:

```
CALL { identifier-1 } [ USING { data-name-1 } ... ]
     { literal-1    }         { literal-2 }

     [ ON OVERFLOW imperative-statement-1 ]

     [ NOT ON OVERFLOW imperative-statement 2 ] [ END-CALL ]
```

Syntax Rules:

1. Literal-1 must be a nonnumeric literal
2. Identifier-1 must be defined as an alphanumeric data item such that its value can be a program-name.
3. The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program; the number of operands in each USING phrase must be identical.
4. Each of the operands in the USING phrase must be defined as a data item in the File Section, Working-Storage Section, Communication Section, Constant Section, or Linkage Section, and each must have a level-number of 01 or 77. Data name-1, etc., can be qualified when they reference data items defined in the File Section or the Communication Section.
5. There may be no more than 511 operands.
6. All references to literal 2 apply equally to all recurrances.

General Rules:

1. The program whose name is specified by the value of literal 1 or identifier-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. The ON OVERFLOW phrase is used for program documentation and has no effect on the object program, since the condition can not occur on Multics.

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

6. The data names and literals specified by the USING phrase of the CALL statement indicate those data items available to a calling program that can be referred to in the called program. 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 significant. Corresponding data names refer to a single set of data available to both the called and calling programs. 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 indexes.

   If literal-2 is nonnumeric, the corresponding data name in the called program must be alphanumeric, equal in length to the number of characters in literal-2, and explicitly or implicitly have DISPLAY USAGE. If literal-2 is an unsigned numeric, the corresponding data-name in the called program,, must be an unsigned numeric, equal in length to the number of digits in literal-2, and have DISPLAY USAGE. If literal-2 is a signed numeric, the corresponding data-name must be numeric (with a separate leading sign), equal in length to the number of digits in literal-2, and have DISPLAY USAGE. (NOTE:&&This is the data-type produced by PL/I for fixed decimal data.)

   If literals are used, the user must ensure that the calling program does not attempt to modify the corresponding data item. All data is passed by reference.

7. The CALL statement can appear anywhere within a segmented program. The user must provide all controls necessary to ensure that the proper logic flow is maintained. Therefore, when a CALL statement appears in a section with a segment-number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement returns control to the calling program.

8. A called program can contain a FILE SECTION and its own local files. When a called program is exited via an EXIT PROGRAM statement, any files opened within that program remain open unless specifically closed within that program.

9. Files declared within a called program can be opened or closed by the calling program provided they are declared in the calling program with the EXTERNAL attribute and have the same internal-file-name (ifn) in both the calling and called programs.

10. If the called program cannot be found, an object time error results. (Refer to "Non-Input/Output Errors" in Section 7.)

11. For additional information on CALL, refer to "Interprogram Communication" in the Multics COBOL Users' Guide.

CANCEL Statement

The CANCEL statement releases the memory areas occupied by the referenced program.

General Format:

```
CANCEL { identifier-1 } ...
       { literal-1    }
```

Syntax Rules:

1. Literal-1, etc., must be nonnumeric literals.
2. Identifier 1, etc. must each be defined as an alphanumeric data item such that its value can be a program name.

General Rules:

1. After execution of a CANCEL statement, the program referred to therein ceases to have any logical relationship to the run-unit in which the CANCEL statement appears. A subsequently executed CALL statement naming the same program will result in that program being initiated in its initial state. Memory areas associated with the named programs are released and are made available for disposition by the operating system.
2. A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.
3. A logical relationship to a cancelled subprogram is established only by execution of a subsequent CALL statement.
4. A called program is cancelled either by being referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.
5. No action is taken when a CANCEL statement is executed naming a program that has not been called in this run-unit or has been called and is at present cancelled. Control passes to the next statement.
6. Any file that is open when a CANCEL statement is issued remains in its last used state. Any files remaining open at the end of the run-unit are automatically closed.

EXIT PROGRAM

EXIT PROGRAM Statement

The EXIT PROGRAM statement marks the logical end of a called program.

General Format:

EXIT PROGRAM .

Syntax Rules:

1. The EXIT PROGRAM statement must appear in a sentence by itself.
2. The EXIT PROGRAM sentence must be the only sentence in the paragraph.

General Rules:

1. The execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program that is not called causes the statement to act as if it were an EXIT statement. (Refer to the EXIT statement in Section 7.)

SECTION 15

COMMUNICATION FACILITY

DESCRIPTION OF THE COMMUNICATION FACILITY

The communication facility provides the capability for accessing processing, and creating messages or portions thereof It provides the ability to communicate with local and remote communication devices through the COBOL Message Control System.

The Multics runtime package, hereafter called the COBOL Message Control System or CMCS provides full support for the facility described by the ANSI COBOL 74 Standard. In addition, the PURGE verb from the CODASYL Journal of Development (JOD) is supported

COBOL Message Control System (CMCS)

The CMCS is the logical interface with the operating system under which the COBOL object program operates The primary functions of the CMCS are:

a. To act as an interface between the COBOL object program and the network of communication devices

b. To perform line discipline, including such tasks as dial up, polling, and synchronization.

c. To perform device-dependent tasks, such as character translation and insertion of control characters, so that the COBOL user can create device-independent programs

The first function, that of interfacing the COBOL object program with the communication devices. is the most obvious to the COBOL user. For the Multics implementation, the COBOL MCS does not usurp any communication device control functions from the operating system. Rather, the ability of the COBOL object program to enable or disable the logical connection between a terminal and the system is effective only while the terminal is used for communicating with the CMCS.

The user may actually be unaware that the other two functions exist. Messages from communication devices are placed in input queues by the CMCS while awaiting disposition by the COBOL object program. Output messages from the COBOL object program are placed in output queues by the CMCS while awaiting transmission to communication devices. The structures, formats, and symbolic names of the queues are defined by the CMCS Administrator to the CMCS by means of a special queue generation language, before the execution of the COBOL object program. Symbolic names for message sources and destinations are also defined at that time The user must specify symbolic names in the source program that are known to the CMCS.

During the execution of a COBOL object program, the CMCS performs all necessary activities to update the various queues as required.

Relationship with COBOL Object Program

The COBOL object program interfaces with the CMCS when it is necessary to send data, receive data, or to interrogate the status of the various queues that are created and maintained by the CMCS. In addition, the COBOL object program may direct the CMCS to establish or break the logical connection between the communication device and a specified portion of the CMCS queue structure. The method of handling the physical connection is a function of the Multics operating system.

Relationship of COBOL Program to the CMCS and Communication Devices

The interfaces that exist in a COBOL communication environment are established by the use of the level indicator CD and associated clauses in the Communication Section of the Data Division. There are two such interfaces.

1. The interface between the COBOL object program and the CMCS
2. The interface between the CMCS and the communication devices.

The COBOL source program uses three statements in the Procedure Division to control the interface with the CMCS:

1. The RECEIVE statement, which causes data in a queue to be passed to the COBOL object program.
2. The SEND statement, which causes data associated with the COBOL object program to be passed to one or more queues.
3. The ACCEPT MESSAGE COUNT statement causes the CMCS to indicate to the COBOL object program the number of complete messages in the specified queue structure.

The COBOL source program uses two statements in the Procedure Division to control the interface between the CMCS and the communication devices.

1. The ENABLE statement, which establishes a logical connection between the CMCS and one or more given communication devices.
2. The DISABLE statement, which breaks a logical connection between the CMCS and one or more given communication devices.

INVOKING THE COBOL OBJECT PROGRAM

The two methods of invoking the COBOL communication object program are scheduled initiation and COBOL message control system (CMCS) invocation. The only difference between the two methods is that CMCS invocation causes certain areas in the referenced communication description (CD entry) to be filled.

Scheduled Invocation of the COBOL Object Program

A COBOL object program using the communication facility may be scheduled for execution through normal means (e.g., invoked as a command). In this case, the COBOL program can use three methods to determine what messages, if any, are available in the input queue:

1. The ACCEPT MESSAGE COUNT statement
2. The RECEIVE statement with a NO DATA phrase
3. The RECEIVE statement without a NO DATA phrase (in which case a program wait is implied if no data is available)

Invocation of the COBOL Object Program by the CMCS

It is sometimes desirable to schedule a COBOL object communication program only when there is work available for it to do. If the CMCS determines that the program is required to process the available message, it schedules the program for execution. Each object program scheduled by the CMCS establishes a COBOL run unit. Prior to the execution of the object program, the CMCS places the symbolic queue and sub-queue names in the associated data items in the CD entry that specifies the FOR INITIAL INPUT clause.

If a subsequent RECEIVE statement is directed to the CD entry, the available message is passed to the object program.

Determining the Method of Scheduling

A COBOL source program can be written so that its object program can operate with either of two methods of scheduling. In order to determine the method used to load the COBOL object program, the following technique may be used:

1. One CD entry must contain a FOR INITIAL INPUT clause.
2. When the program contains a CD entry with the FOR INITIAL INPUT clause, the Procedure Division may contain statements to test the initial value of the symbolic queue name in the CD entry. If it is not space-filled, the CMCS invokes the object program and initializes the data item with the symbolic name of the queue containing the message to be processed.

The Concept of Messages and Message Segments

A message consists of some arbitrary amount of information, usually character data, whose beginning and end are defined or implied. As such the message constitutes the fundamental, but not necessarily the most elementary unit of data to be processed in a COBOL communication environment.

Messages are logically subdivided into smaller units called message segments which are delimited within a message by means of end of segment indicators (ESI). A message consisting of one or more segments is delimited from the next message by an end-of-message indicator (EMI). In a similar manner, a group of several messages may be logically separated from succeeding messages by an end of group indicator (EGI). When a message or a message segment is received by the COBOL program, a CD area is updated by the CMCS to indicate which, if any, delimiter was associated with the text transferred by the RECEIVE statement. On output the delimiter, if any, to be associated with the text released to the CMCS is specified in the SEND statement. Thus the presence of these indicators is recognized by the CMCS and the COBOL program; however, no indicators are included in the message text.

A precedence relationship exists between the indicators EGI, EMI, and ESI. EGI is the most inclusive and ESI is the least inclusive indicator. The existence of an indicator associated with message text implies the association of all "less inclusive indicators" with that text.

The Concept of Queues

The following discussion applies only when the COBOL communication environment is established using a CD entry without the FOR I-O clause.

Queues consist of one or more messages from one or more communication devices and as such, form the data buffers between the COBOL program and the CMCS. Input queues are logically separate from output queues.

The CMCS places in and removes only complete messages from queues. Portions of messages are not logically placed in queues until the entire message is available to the CMCS. That is, The CMCS does not pass a message segment to a COBOL program unless all segments of that message are in the input queue; even though the COBOL program uses the SEGMENT phrase of the RECEIVE statement. For output messages, the CMCS does not transmit any segment of a message until all of its segments are in the output queue. Interrogation of the number of messages that exist in a queue reflects only the number of complete messages that exist in the queue.

The process by which messages are placed in a queue is called enqueueing. The process by which messages are removed from a queue is called dequeueing.

INDEPENDENT ENQUEUEING AND DEQUEUEING

A message may be received by the CMCS from a terminal prior to the execution of the COBOL object program. As a result, the CMCS enqueues the message in the proper input queue (if the input queue is enabled) until the COBOL object program requests dequeueing with the RECEIVE statement. It is also possible that the COBOL object program causes the enqueueing of messages in an output queue which are not transmitted to a communications device until after the COBOL object program is terminated. Two common reasons for this occurrence are:

1. When the output queue is disabled
2. When the COBOL object program creates output messages at a speed faster than the destination can receive them

ENABLING AND DISABLING QUEUES

Usually, the MCS enables and disables queues based on time of day, message activity, or other factors unrelated to the COBOL object program. However, the COBOL program can enable and disable queues itself through use of the ENABLE and DISABLE statements.

QUEUE HIERARCHY

In order to control more explicitly the messages being enqueued and dequeued, a hierarchy of input queues (i.e., queues composed of queues) can be defined in the MCS. In COBOL, four levels of queues are available to the user. In order of decreasing significance, the four levels are named queue, sub-queue-1, sub-queue-2, and sub-queue-3. The full queue structure is depicted in Figure 15-1, where queues and sub-queues are named with the letters A through O. Messages are named with a letter according to their source (X, Y, or Z) and with a sequential number.

```
dcl 01 A,                 /* queue                    */
    02    B,              /* sub-queue-1              */
    03       D,           /* sub-queue-2              */
    04          H,        /* sub-queue-3              */
                          /* messages(Z1,Z2)          */
    04          I,        /* sub-queue-3              */
                          /* messages(X3,X4,X5)       */

    03       E,           /* sub-queue-2              */
    04          J,        /* sub-queue-3              */
                          /* messages(X1,Y3,Y5,Z5)    */
    04          K,        /* sub-queue-3              */
                          /* messages(Z6,Z7,Y6)       */

    02    C,              /* sub-queue-1              */
    03       F,           /* sub-queue-2              */
    04          L,        /* sub-queue-3              */
                          /* messages(Y7,Y8)          */
    04          M,        /* sub-queue-3              */
                          /* messages (Y1,Y2)         */

    03       G,           /* sub-queue-2              */
    04          N,        /* sub-queue-3              */
                          /* messages(X6)             */
    04          O,        /* sub-queue-3              */
                          /* messages(Z2,Z3,Z4,Y4)    */
```

Figure 15-1. Hierarchy of Queues

The CMCS establishes the following queueing algorithm:

1. Messages are placed in queues which are specified by the sender of the message.

2. With the RECEIVE statement, if the user does not specify a sub-queue level, the CMCS chooses the sub-queue in declaration order (e.g., if sub-queue-1 is not specified, the CMCS dequeues from sub-queue-1 B).

The following examples illustrate the above algorithm:

1. The program executes a RECEIVE statement, specifying via the CD entry:

   Queue A
   CMCS returns: Message Z1

2. The program executes a RECEIVE statement, specifying via the CD entry:

   Queue A
   Sub-queue-1 C
   CMCS returns: Message Y7

3. The program executes a RECEIVE statement, specifying via the CD entry:

   Queue A
   Sub-queue-1 B
   Sub-queue-2 E
   CMCS returns: Message X1

4. The program executes a RECEIVE statement, specifying via the CD entry:

   Queue A
   Sub-queue-1 C
   Sub-queue-2 G
   Sub-queue-3 N
   CMCS returns: Message X6

If the user wishes to access the next message in a queue, regardless of which sub-queue that message may be in, he or she may specify the queue name only. The CMCS, when supplying the message, returns to the COBOL object program any applicable sub-queue names via the data items in the associated CD entry. If, however, the user desires the next message in a given sub-queue, both the queue name and any applicable sub-queue names must be specified.

For output, the user specifies only the destination(s) of the message, and the CMCS places the message in the proper queue structures.

COMMUNICATION FACILITY DATA DIVISION - COMMUNICATION SECTION

In a COBOL program, the communication description (CD) entries represent the highest level of organization in the Communication Section. The Communication Section header is followed by a communication description entry consisting of a level indicator (CD), a data-name, and a series of independent clauses. These clauses indicate the queues and sub-queues, the message date and time, the source, the text length, the status and end keys, and message count of input. These clauses also specify the destination count, the text length, the status and error keys, and destinations for output. The entry itself is terminated by a period. These record areas may be implicitly redefined by user-specified record description entries following the various communication description clauses.

CD SKELETON

Communication Description Complete Entry Skeleton

The communication description specifies the interface area between the CMCS and a COBOL program.

Format 1:

```
CD cd-name FOR [ INITIAL ] INPUT

[
  [ [ SYMBOLIC QUEUE IS data-name-1 ]
      [ SYMBOLIC SUB-QUEUE-1 IS data-name-2 ]
      [ SYMBOLIC SUB-QUEUE-2 IS data-name-3 ]
      [ SYMBOLIC SUB-QUEUE-3 IS data-name-4 ]
      [ MESSAGE DATE IS data-name-5 ]
      [ MESSAGE TIME IS data-name-6 ]
      [ SYMBOLIC SOURCE IS data-name-7 ]
      [ TEXT LENGTH IS data-name-8 ]
      [ END KEY IS data-name-9 ]
      [ STATUS KEY IS data-name-10 ]
      [ MESSAGE COUNT IS data-name-11 ] ]
  [ data-name-, data-name-2 ... data-name-11 ]
]
```

CD SKELETON

Format 2:

```
CD cd-name FOR OUTPUT

    [ DESTINATION COUNT IS data-name-1 ]

    [ TEXT LENGTH IS data name-2 ]

    [ STATUS KEY IS data name-3 ]

    [ DESTINATION TABLE OCCURS integer-2 TIMES
      [ INDEXED BY { index-name-1 } ... ] ]

    [ ERROR KEY IS data name-4 ]

    [ SYMBOLIC DESTINATION IS data-name-5 ]
```

Syntax Rules:

Format 1:

1. The level indicator CD must appear only in the Communication Section.
2. Within a single program, the INITIAL clause may be specified in only one CD. The INITIAL clause must not be used in a program that specifies the USING phrase of the Procedure Division header.
3. Except for the INITIAL clause, the optional clauses may be specified in any order.
4. If none of the options in the format are specified, a level 01 data description entry must follow the CD entry. Any of the options may be followed by a level 01 data description entry.
5. For each input CD, a record area of 87 contiguous standard data format characters is allocated. This record area is defined to the CMCS as follows:
   a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1-12 in the record.
   b. The SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13-24 in the record.

CD SKELETON

c. The SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25-36 in the record.

d. The SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37-48 in the record.

e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of six digits without an operational sign occupying character positions 49-54 in the record.

f. The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of eight digits without an operational sign occupying character positions 55-62 in the record.

g. The SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63-74 in the record.

h. The TEXT LENGTH clause defines data-name-8 as the name of an elementary data item whose implicit description is that of an integer of four digits without an operational sign occupying character positions 75-78 in the record.

i. The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of one character occupying position 79 in the record.

j. The STATUS KEY clause clause defines data-name-10 as the name of an elementary alphanumeric data item of two characters occupying positions 80-81 in the record.

k. The MESSAGE COUNT clause defines data-name-11 as the name of an elementary data item whose implicit description is that of an integer of six digits without an operational sign occupying character positions 82-87 in the record.

The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clauses.

CD SKELETON

The use of either of the above options results in a record whose implicit description is equivalent to the following:

Implicit Description			Comment
01	data-name-0.		
02	data-name-1	PICTURE X(12).	SYMBOLIC QUEUE
02	data-name-2	PICTURE X(12).	SYMBOLIC SUB-QUEUE-1
02	data-name-3	PICTURE X(12).	SYMBOLIC SUB-QUEUE-2
02	data-name-4	PICTURE X(12).	SYMBOLIC SUB-QUEUE-3
02	data-name-5	PICTURE 9(06).	MESSAGE DATE
02	data-name-6	PICTURE 9(08).	MESSAGE TIME
02	data-name-7	PICTURE X(12).	SYMBOLIC SOURCE
02	data-name-8	PICTURE 9(04).	TEXT LENGTH
02	data-name-9	PICTURE X.	END KEY
02	data-name-10	PICTURE XX.	STATUS KEY
02	data-name-11	PICTURE 9(06).	MESSAGE COUNT

NOTE: In the above description, the information in the 'Comment' column is for clarification only and is not part of the description.

6. Record description entries following an input CD implicitly redefine this record and must describe a record of exactly 87 characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. The CMCS always references the record according to the data descriptions defined in syntax rule 5 above.

7. Data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

Format 2:

1. The level indicator CD must appear only in the Communication Section.

2. If none of the optional clauses under the CD entry are specified, a data description entry must follow the CD entry.

CD SKELETON

3. For each output CD, a record area of contiguous standard data format characters is allocated according to the following formula: (10 plus 13 multiplied by integer-2).

   a. The DESTINATION COUNT clause defines data-name-1 as the name of a data item whose implicit description is that of an integer without an operational sign occupying character positions 1 4 in the record.

   b. The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of four digits without an operational sign occupying character positions 5-8 in the record.

   c. The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of two characters occupying positions 9-10 in the record.

   d. Character positions 11-23 and every set of 13 characters thereafter will form table items of the following description:

      - The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of one character.
      - The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

   The use of the above clauses results in a record whose implicit description is equivalent to the following:

```
Implicit Description                          Comment

01  data-name-0.

    02  data-name-1  PICTURE 9(04).           DESTINATION COUNT
    02  data-name-2  PICTURE 9(04).           TEXT LENGTH
    02  data-name-3  PICTURE XX.              STATUS KEY
    02  data-name  OCCURS integer-2 TIMES.    DESTINATION TABLE

        03  data-name-4  PICTURE X.           ERROR KEY
        03  data-name-5  PICTURE X(12).       SYMBOLIC DESTINATION
```

   NOTE: In the above description, the information in the 'Comment' column is for clarification only and is not part of the description.

4. Record descriptions following an output CD implicitly redefine this record. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. The CMCS always references the record according to the data descriptions defined in syntax rule 3 for Format 2.

5. Data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.

CD SKELETON

6. If the DESTINATION TABLE OCCURS clause is not specified, one (1) ERROR KEY and one (1) SYMBOLIC DESTINATION area is assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.

7. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and data-name-5 may only be referred to by subscripting or indexing.

General Rules:

Format 1:

1. The input CD information constitutes the communication between the CMCS and the program as information about the message being handled. This information does not come from the terminal as part of the message.

2. The contents of the data items referenced by data-name-2, data-name-3, and data-name-4, when not being used, must contain spaces.

3. The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 contain symbolic names designating queues, sub-queues, ..., respectively. All symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the CMCS.

4. A RECEIVE statement causes the serial return of the 'next' message or a portion of a message from the queue as specified by the entries in the CD.

   If, during the execution of a RECEIVE statement, a message from a more specific source is needed, the contents of the data item referenced by data-name-1 can be made more specific by the use of the contents of the data items referenced by data-name-2, data-name-3, and in turn, data-name-4. When a given level of the queue structure is specified, all higher levels must also be specified.

   If less than all of the levels of the queue hierarchy are specified, the CMCS determines the 'next' message or portion of a message to be accessed.

   After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 contain the symbolic names of all the levels of the queue structure.

5. Whenever a program is scheduled by the CMCS to process a message, the symbolic names of the queue structure that demanded this activity are placed in the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause, as applicable. In all other cases, the contents of the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause are initialized to spaces.

   The symbolic names are inserted (or the initialization to spaces is completed) prior to the execution of the first Procedure Division statement.

CD SKELETON

The execution of a subsequent RECEIVE statement naming the same contents of the data items referenced by data-name-1 through data-name-4 returns the actual message that caused the program to be scheduled. Only at that time is the remainder of the CD updated.

6. If the CMCS attempts to schedule a program that does not contain an INITIAL clause, the results are undefined.

7. The contents of data-name-5 have the format 'YYMMDD' (year, month, day) and represent the date on which the CMCS recognizes that the message is complete.

   The contents of the data item referenced by data-name-5 are only updated by the CMCS as part of the execution of a RECEIVE statement.

8. The contents of data-name-6 have the format 'HHMMSSTT' (hours, minutes, seconds, hundredths of a second) and represent the time when the CMCS recognizes that the message is complete.

   The contents of the data item referenced by data-name-6 are only updated by the CMCS as part of the execution of a RECEIVE statement.

9. During the execution of a RECEIVE statement, the CMCS provides, in the data item referenced by data-name-7, the symbolic name of the communication terminal that is the source of the message being transferred. However, if the symbolic name of the communication terminal is not known to the CMCS, the contents of the data item referenced by data-name-7 contain spaces.

10. The CMCS indicates via the contents of the data item referenced by data-name-8 the number of character positions filled as a result of the execution of a RECEIVE statement.

11. The contents of the data item referenced by data-name-9 are set only by the CMCS as part of the execution of a RECEIVE statement according to the following rules:

    a. When the MESSAGE phrase of the RECEIVE statement is specified, then:

       - If an end of group is detected, the contents of the data item referenced by data-name-9 are set to 3.
       - If an end of message is detected, the contents of the data item referenced by data-name-9 are set to 2.
       - If less than a message is transferred, the contents of the data item referenced by data-name-9 are set to 0.

    b. When the SEGMENT phrase of the RECEIVE statement is specified, then:

       - If an end of group is detected, the contents of the data item referenced by data-name-9 are set to 3.
       - If an end of message is detected, the contents of the data item referenced by data name-9 are set to 2.

CD SKELETON

- If an end of segment is detected, the contents of the data item referenced by data name-9 are set to 1.
- If less than a message segment is transferred, the contents of the data item referenced by data-name-9 are set to 0.

c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by data-name-9.

12. The contents of the data item referenced by data name-10 indicate the status condition of the previously executed RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT, or DISABLE INPUT statements.

    The actual association between the contents of the data item referenced by data name-10 and the status condition itself is defined in Figure 15-1.

13. The contents of the data item referenced by data-name-11 indicate the number of messages that exist in a queue, sub-queue-1, ..., The CMCS updates the contents of the data item referenced by data-name-11 only as part of the execution of an ACCEPT statement that contains a COUNT phrase.

Format 2:

1. The nature of the output CD information is such that it is not sent to the terminal, but constitutes the communication between the program and the CMCS as information about the message being handled.

2. During the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the contents of the data item referenced by data name-1 will indicate to the CMCS the number of symbolic destinations that are to be used from the area referenced by data name-5.

   The CMCS finds the first symbolic destination in the first occurrence of the area referenced by data name-5, the second symbolic destination in the second occurrence of the area referenced by data name-5, ..., up to and including the occurrence of the area referenced by data-name-5 indicated by the contents of data name-1.

   If, during the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement the value of the data item referenced by data-name-1 is outside the range of 1 through integer-2, an error condition is indicated and the execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is terminated.

3. It is the responsibility of the user to ensure that the value of the data item referenced by data-name-1 is valid when the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is executed.

4. As part of the execution of a SEND statement, the CMCS interprets the contents of the data item referenced by data-name-2 to be the user's indication of the number of leftmost character positions of the data item referenced by the associated SEND identifier from which data is to be transferred.

CD SKELETON

5. Each occurrence of the data item referenced by data-name-5 contains a symbolic destination previously known to the CMCS. These symbolic destination names must follow the rules for the formation of system-names.

6. The contents of the data item referenced by data-name-3 indicate the status condition of the previously executed SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

   The actual association between the contents of the data item referenced by data name-3 and the status condition itself is defined in Figure 15-1.

7. If, during the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the CMCS determines that any specified destination is unknown, the contents of the data item referenced by data-name-3 and all occurrences of the data items referenced by data-name-4 are updated.

   The contents of the data item referenced by data-name-4 when equal to one (1) indicate that the associated value in the area referenced by data name-5 has not been previously defined to the CMCS. Otherwise, the contents of the data item referenced by data-name-4 are set to zero (0).

CD SKELETON

1-RECEIVE
2-SEND
3-ACCEPT MESSAGE COUNT
4-ENABLE INPUT
5-ENABLE INPUT TERMINAL
6-ENABLE OUTPUT
7-DISABLE INPUT
8-DISABLE INPUT TERMINAL
9-DISABLE OUTPUT
10-STATUS KEY CODE

(see legend above)

1	2	3	4	5	6	7	8	9	10	Description
X	X	X	X	X	X	X	X	X	00	No error detected. Action completed.
	X								10	One or more destinations are disabled. Action completed.
	X				X			X	20	One or more destinations are unknown. Action completed for known destinations. No action taken for unknown destinations.
X		X	X			X			20	One or more queues or sub-queues are unknown. No action taken.
				X			X		20	The source is unknown. No action taken.
*										
	X				X			X	30	Content of DESTINATION COUNT invalid. No action taken.
			X	X	X	X	X	X	40	Password invalid. No enabling/disabling action taken.
	X								50	Character count greater than length of sending field. No action taken.
	X								60	Partial segment with either zero character count or no sending area specified. No action taken.
*										

Note: This figure indicates the possible contents of the data items referenced by data-name-10 for Format 1 and by data-name-3 and data-name-4 for Format 2 at the completion of each statement shown. An 'X' on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

Figure 15-2. Communication Status Key/Error Key Conditions

ACCEPT MESSAGE COUNT

PROCEDURE DIVISION FOR THE COMMUNICATION FACILITY

ACCEPT MESSAGE COUNT Statement

The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available

General Format:

ACCEPT cd-name MESSAGE COUNT

Syntax Rules:

1. cd-name must reference an input CD.

General Rules:

1. The ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in the queue structure designated by the contents of the data items specified by data name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name.

2. Upon execution of the ACCEPT MESSAGE COUNT statement, the contents of the area specified by a communication description (CD) entry must contain at least the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-10 (STATUS KEY) and data-name-11 (MESSAGE COUNT) of the area associated with the communication entry to be appropriately updated.

DISABLE Statement

The DISABLE statement notifies the CMCS to inhibit data transfer between specified destinations and output queues for output or between specified sources and input queues for input.

General Format:

```
DISABLE { INPUT [ TERMINAL ] } cd-name WITH KEY { identifier-1 }
        { OUTPUT             }                  { literal-1    }
```

Syntax Rules:

1. cd-name must reference an input CD when the INPUT phrase is specified.
2. cd-name must reference an output CD when the OUTPUT phrase is specified.
3. Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric.

General Rules:

1. The DISABLE statement provides a logical disconnection between the CMCS and the specified sources or destinations. When this logical disconnection is already in existence, or is to be handled by some other means external to this program, the DISABLE statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the CMCS is not affected by the DISABLE statement.
2. When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all associated queues and sub-queues is deactivated. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful to the CMCS.
3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queues and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are deactivated.

DISABLE

4. When the OUTPUT phrase is specified, the logical path for a destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are deactivated.

5. Literal-1 or the contents of the data-name referenced by identifier-1 are matched with a password built into the system. The DISABLE statement is honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

   The CMCS is capable of handling a password of from one to ten characters inclusive.

6. The CMCS ensures that the execution of a DISABLE statement causes the logical disconnection to occur at the earliest time the source or destination is inactive. Execution of the DISABLE statement never causes the remaining portion of the message to be terminated during transmission to or from a terminal.

ENABLE Statement

The ENABLE statement notifies the CMCS to allow data transfer between specified destinations and output queues for output or between specified sources and input queues for input.

General Format:

```
ENABLE { INPUT [ TERMINAL ] } cd-name WITH KEY { identifier-1 }
       { OUTPUT             }                  { literal-1    }
```

Syntax Rules:

1. cd-name must reference an input CD when the INPUT phrase is specified.
2. cd-name must reference an output CD when the OUTPUT phrase is specified.
3. Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric.

General Rules:

1. The ENABLE statement provides a logical connection between the CMCS and the specified sources or destinations. When this logical connection is already in existence, or is to be handled by some other means external to this program, the ENABLE statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the CMCS is not affected by the ENABLE statement.
2. When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all associated queues and sub-queues that are already enabled is activated. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful to the CMCS.
3. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queues and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are activated.

4. When the OUTPUT phrase is specified, the logical path for a destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are activated.

5. Literal-1 or the contents of the data item referenced by identifier-1 are matched with a password built into the system. The ENABLE statement is honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

   The CMCS is capable of handling a password of from one to ten characters inclusive.

PURGE Statement

The PURGE statement eliminates from the CMCS a partial message that has been released by one or more SEND statements.

General Format:

PURGE cd-name

Syntax Rule:

1. cd-name must reference an output CD.

General Rules:

1. Execution of a PURGE statement causes the CMCS to eliminate any partial message awaiting transmission to the destinations specified in the CD referenced by cd-name.
2. Any message that has associated with it an EMI or EGI is not affected by the execution of a PURGE statement.
3. The contents of the data item referenced by data-name-3 (STATUS KEY) and the contents of the data item referenced by data-name-4 (ERROR KEY) of the area referenced by cd-name are updated by the CMCS. (See "General Rules" under "The Communication Description -- Complete Entry Skeleton.")

RECEIVE

RECEIVE Statement

The RECEIVE statement allows a message, message segment, or a portion of a message or a segment, and related information concerning that data to be made available to the COBOL program from a queue maintained by the CMCS. The RECEIVE statement also allows a specific imperative statement to be executed when no data is available.

General Format:

```
RECEIVE cd-name { MESSAGE } INTO identifier-1
                { SEGMENT }

    [ NO DATA imperative-statement-1 ]

    [ WITH DATA imperative-statement-2 ] [ END-RECEIVE ]
```

Syntax Rules:

1. cd-name must reference an input CD.

General Rules:

1. The contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name designate the queue structure containing the message.

2. The message, message segment, or portion of a message or a segment is transferred to the receiving character positions of the area referenced by identifier-1, aligned to the left without spacefill.

3. When, during the execution of a RECEIVE statement, the CMCS causes data to be made available in the data item referenced by identifier-1, control is transferred to the next executable statement, whether or not the NO DATA phrase is specified.

4. When, during the execution of a RECEIVE statement, the CMCS does not make data available in the data item referenced by identifier-1, the following conditions occur:

   a. If the NO DATA phrase is specified, the RECEIVE operation is terminated with the indication that activities are complete (see general rule 5), and the imperative statement in the NO DATA phrase is executed.

   b. If the NO DATA phrase is not specified, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.

RECEIVE

c. If one or more queues or sub-queues is unknown to the CMCS, control passes to the next executable statement, whether or not the NO DATA phrase is specified. See Figure 15-1.

5. The data items identified by the input CD are appropriately updated by the CMCS during each execution of a RECEIVE statement.

6. A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or more than a single segment (when the SEGMENT phrase is used). However, the CMCS does not pass any portion of a message to the object program until the entire message is available in the input queue, even if the SEGMENT phrase of the RECEIVE statement is specified.

7. When the MESSAGE phrase is specified, end-of-segment indicators are ignored, and the following rules apply to the data transfer:

   a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.

   b. If the message size is less than the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 with no spacefill.

   c. If the message size is greater than the area referenced by identifier-1, the message fills the area referenced by identifier-1 from left to right starting with the leftmost character of the message. The remainder of the message can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue, sub-queue, ..., . The remainder of the message, for the purposes of applying rules 7a, 7b, and 7c (above), is treated as a new message.

8. When the SEGMENT phrase is specified, the following rules apply:

   a. If a segment is the same size as the area referenced by identifier-1, the segment is stored in the area referenced by identifier-1.

   b. If the segment size is less than the area referenced by identifier-1, the segment is aligned to the leftmost character position of the area referenced by identifier-1 with no spacefill.

   c. If the segment size is greater than the area referenced by identifier-1, the segment fills the area referenced by identifier-1 from left to right starting with the leftmost character of the segment. The remainder of the segment can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue, sub-queue, ..., . The remainder of the segment, for the purposes of applying these rules (8a, 8b, and 8c), is treated as a new segment.

RECEIVE

9. If the text to be accessed by the RECEIVE statement has an end-of-message indicator or an end-of-group indicator associated with it, the existence of an end of-segment indicator associated with the text is implied and the text is treated as a message segment according to general rule 8.

10. After the execution of a RECEIVE statement has returned a portion of a message, only the subsequent execution of RECEIVE statements in that run unit can cause the remaining portion of the message to be returned.

11. After the execution of a STOP RUN statement, any input message only partially obtained in that run unit is returned to the input queue.

SEND Statement

The SEND statement causes a message, a message segment, or a portion of a message or a segment to be released to one or more output queues maintained by the CMCS.

Format 1:

```
SEND  cd-name  FROM  identifier-1
```

Format 2:

```
SEND  cd-name [ FROM  identifier-1 ] { WITH  identifier-2 }
                                     { WITH  ESI          }
                                     { WITH  EMI          }
                                     { WITH  EGI          }

[ { BEFORE } ADVANCING { { { identifier-3 } [ LINE  ] } }  ]
  { AFTER  }           { { { integer      } [ LINES ] } }
                       {   { mnemonic-name }           }
                       {   { PAGE          }           }
```

Syntax Rules:

Formats 1 and 2:

1. cd-name must reference an output CD.
2. Identifier-2 must reference a one-character integer that does not have an operational sign.
3. When identifier-3 is used in the ADVANCING phrase, it must be the name of an elementary integer item.
4. Integer, or the value of the data item referenced by identifier-3, may be zero.
5. Mnemonic-name must be defined by the SPECIAL-NAMES paragraph of the Environment Division, and can have an integer value only in the range of 1-16.

General Rules

Formats 1 and 2·

1. When a receiving communication device (printer display screen card punch) is oriented to a fixed line size:

   a. Each message or message segment begins at the leftmost character position of the physical line.

   b. A message or message segment that is smaller than the physical line size is released so as to appear space-filled to the right.

   c. Excess characters of a message or message segment are not truncated. The characters are packed to a size equal to that of the physical line and then transmitted to the communication device. The process continues on the next line with the excess characters

2. When a receiving communication device (paper tape punch, another computer etc.) is oriented to handle variable-length messages, each message or message segment begins in the next available character position of the communication device.

3. As part of the execution of a SEND statement, the CMCS interprets the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name to be the user's indication of the number of leftmost character positions of the data item referenced by identifier 1 from which data is to be transferred.

   If the contents of the data item referenced by data name-2 (TEXT LENGTH) of the area referenced by cd-name are zero, no characters of the data item referenced by identifier-1 are transferred.

   If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred. See Figure 15-1.

4. As part of the execution of a SEND statement, the contents of the data item referenced by data name-3 (STATUS KEY) of the area referenced by cd-name are updated by the CMCS.

5. The effect of placing special control characters within the contents of the data item referenced by identifier 1 is undefined.

6. A single execution of a Format 1 SEND statement releases only a single portion of a message or of a message segment to the CMCS.

A single execution of a Format 2 SEND statement never releases to the CMCS more than a single message or a single message segment as indicated by the contents of the data item referenced by identifier-2 or by the specified indicator ESI EMI, or EGI.

However, the CMCS does not transmit any portion of a message to a communication device until the entire message is placed in the output queue.

7. During the execution of the run unit, the disposition of a portion of a message not terminated by an EMI or EGI is undefined. However, the message does not logically exist for the CMCS and therefore cannot be sent to a destination.

   After the execution of a STOP RUN statement, any portion of a message transferred from the run unit via a SEND statement, but not terminated by an EMI or EGI, is purged from the system. Thus, no portion of the message is sent.

8. After the execution of a SEND statement has released a portion of a message to the CMCS, only the subsequent execution of SEND statements in the same run unit can cause the remaining portion of the message to be released.

General Rules:

Format 2 Only:

1. The content of the data item referenced by identifier-2 indicates that the content of the data item referenced by identifier-1 is to have an end-of-segment indicator, an end-of-message indicator, or an end-of-transmission indicator associated with it as shown below:

If the content of the data item referenced by identifier-2 is	then the content of the data item referenced by identifier-1 has associated with it	which means
'0'	no indicator	no indicator
'1'	ESI	end-of-segment indicator
'2'	EMI	end-of-message indicator
'3'	EGI	end-of-group indicator

Any character other than '1', '2', or '3' will be interpreted as '0'.

SEND

If the content of the data item referenced by identifier-2 is other than '1', '2', or '3', and identifier 1 is not specified, then an error is indicated by the value in the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred.

2. The ESI indicates to the CMCS that the message segment is complete. The EMI indicates to the CMCS that the message, which may consist of more than one segment, is complete.

   The EGI indicates to the CMCS that the group of messages is complete.

   The CMCS recognizes these indicators and establishes the internal mechanisms necessary to maintain group, message, and segment control.

3. The hierarchy of ending indicators is EGI EMI, and ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI.

4. The ADVANCING phrase allows control of the vertical positioning of each message or message segment on a communication device where vertical positioning is applicable. If vertical positioning is not applicable for the device, the CMCS ignores the vertical positioning specified or implied.

5. If identifier-2 is specified, and the content of the data item referenced by identifier-2 is zero, the ADVANCING phrase is ignored by the CMCS.

6. On a communication device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing is provided as if the user had specified AFTER ADVANCING 1 LINE.

7. If the ADVANCING phrase is specified, explicitly or implicitly, and vertical positioning is applicable, the following rules apply:

   a. If identifier-3 or integer is specified, the characters transmitted to the communication device are repositioned vertically downward the number of lines equal to the value associated with the data item referenced by identifier-3 or integer.

   b. If mnemonic-name is specified, the characters transmitted to the communication device are positioned according to the rules specified for that communication device.

   c. If the BEFORE phrase is used, the message or message segment is represented on the communication device before vertical repositioning according to general rules 7a and 7b above.

   d. If the AFTER phrase is used, the message or message segment is represented on the communication device after vertical repositioning according to general rules 7a and 7b above.

   e. If the PAGE phrase is specified, the characters transmitted to the communication device are represented on the device before or after (depending on the phrase used) the device is repositioned to the next page. If PAGE is specified, but page has no meaning in conjunction with a specific device, then advancing is provided as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

SECTION 16

REPORT WRITER

DESCRIPTION OF THE REPORT WRITER

The report writer feature provides the facility for producing reports by specifying the physical appearance of a report instead of specifying the detailed procedures necessary to produce the report.

A hierarchy of levels is used to define the logical organization of a report. Each report is divided into report groups, which in turn are divided into sequences of items. This hierarchical structure permits explicit reference to a report group with implicit reference to other levels in the hierarchy. A report group contains one or more items to be presented on one or more lines.

The report writer feature emphasizes the organization, format and contents of an output report Although a report can be produced using the standard COBOL language, the report writer language characteristics provide a more concise method for report structuring and report production. Much of the Procedure Division coding which would normally be supplied by the user is instead provided automatically by the Report Writer Control System (RWCS). Thus, the user is relieved of writing procedures for moving data, constructing print lines, counting lines on a page, numbering pages, producing heading and footing lines, recognizing the end of logical data subdivisions, updating sum counters, etc. All of these operations are accomplished by the RWCS from source language statements that appear primarily in the Report Section of the Data Division of the source program.

Data movement to a report is directed by the Report Section clauses SOURCE, SUM, and VALUE. Fields of data are positioned on a print line by means of the COLUMN NUMBER clause. The PAGE clause specifies the length of the page, the size of the heading and footing areas, and the size of the area in which the detail lines will appear Data items may be specified to form a control hierarchy. During the execution of a GENERATE statement, the RWCS uses the control hierarchy to check automatically for control breaks. When a control break occurs, summary information (subtotals) can be presented.

LINE-COUNTER

The reserved word LINE-COUNTER is a name for a special register that is generated for each report description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from zero (0) through 999999. The usage is COMP-6. The value in LINE-COUNTER is maintained by the RWCS and is used to determine the vertical positioning of a report. The value in LINE-COUNTER may be accessed by Procedure Division statements; however, only the RWCS can change the value of LINE-COUNTER.

PAGE-COUNTER

The reserved word PAGE-COUNTER is a name for a special register that is generated for each report description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from one (1) through 999999. The usage is COMP-6. The value in PAGE-COUNTER is maintained by the RWCS and is used by the program to number the pages of a report. The value in PAGE-COUNTER may be altered by Procedure Division statements.

In the Report Section, neither a sum counter nor the special registers LINE-COUNTER and PAGE-COUNTER can be used as a subscript.

Relationship with File Input/Output

The report file is described in Section 9, File Input/Output, and is subject to the following restrictions.

An OPEN statement, specifying either the OUTPUT phrase or the EXTEND phrase, must have been executed prior to the execution of the INITIATE statement, and a CLOSE statement, without the REEL phrase or the UNIT phrase, must be executed for this file subsequent to the execution of the TERMINATE statement. No other input/output statement may be executed for this file.

DATA DIVISION FOR THE REPORT WRITER

File Section for the Report Writer

A REPORT(S) clause is required in the FD entry to list the names of the reports to be produced.

Report Section for the Report Writer

In the Report Section, the description of each report must begin with a report description entry (RD entry) and be followed by the entries that describe the report groups within the report.

REPORT DESCRIPTION ENTRY

In addition to naming the report, the RD entry defines the format of each page of the report by specifying the vertical boundaries of the region within which each type of report group may be printed. The RD entry also specifies the control data items. When the report is produced, changes in the values of the control data items cause the detail information of the report to be processed in groups called control groups.

Each report named in the REPORT(S) clause of an FD entry in the File Section must be the subject of an RD entry in the Report Section. Furthermore, each report in the Report Section must be named in one, and only one, FD entry.

REPORT GROUP DESCRIPTION ENTRY

The report groups that will make up the report are described following the RD entry. The description of each report group begins with a report group description entry; that is, an entry that contains a level-number 01 and a TYPE clause. Subordinate to the report group description entry there may appear group and elementary entries that further describe the characteristics of the report group.

FD SKELETON

File Description Entry For The Report Writer

The file description furnishes information concerning the physical structure, identification, and report-names pertaining to a given report file.

General Format:

```
FD  file-name

[ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                            { CHARACTERS }
[ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]
[ CODE-SET IS { alphabet-name } ]
              { STANDARD-1    }
              { NATIVE        }
              { ASCII         }
              { EBCDIC        }
  LABEL { RECORD  IS } { STANDARD }
        { RECORDS ARE } { OMITTED  }
  { REPORT  IS } report-name-1 [ report-name-2 ] ...
  { REPORTS ARE }
[ VALUE OF { FILE-ID      IS { data-name-1 } } ]
           {                 { literal-1   } }
           { RETENTION    IS { data-name-2 } }
           {                 { literal-2   } }
           { CATALOG-NAME IS { data-name-3 } }
           {                 { literal-3   } }
```

FD SKELETON

Syntax Rules:

1. The level indicator FD identifies the beginning of a file description and must precede the file name.

2. The clauses which follow the name of the file are optional in most cases and their order of entry is not significant.

3. The file referenced by file-name must be defined, explicitly or implicitly, in the FILE-CONTROL paragraph of the Environment Division, as a sequential file. In addition, each report named in the REPORT(S) clause must be the subject of a report description entry in the Report Section.

4. No record description entries are permitted for file-name and no input/output statements, except the OPEN statement with either the OUTPUT or EXTEND phrase and the CLOSE statement without either the REEL or UNIT phrase, may be executed for this file.

General Rule:

Descriptions, applications, and rules governing the usage of the LABEL RECORDS and VALUE OF clauses are presented in Section 9, File Input/Output.

RD SKELETON

Report Description - Complete Entry Skeleton

The report description entry names a report, specifies any identifying characters to be appended to each print line, and describes the physical structure and organization of that report.

General Format:

```
RD report-name

  [ CODE literal-1 ]

  [ { CONTROL  IS  } { { data-name-1 } ...                          } ]
  [ { CONTROLS ARE } { FINAL [ { data-name-1 } ... ]                } ]
  [ PAGE [ LIMIT  IS ] integer-1 [ LINE  ]                          ]
         [ LIMITS ARE]           [ LINES ]
    [ HEADING integer-2 ]
      [ FIRST DETAIL integer-3 ]
      [ LAST DETAIL integer-4 ]
      [ FOOTING integer-5 ]                                         ] .
```

Syntax Rules:

1. The report-name must appear in one, and only one, REPORT clause.
2. The order of appearance of the clauses following the report-name is immaterial.
3. Report-name is the highest permissible qualifier that may be specified for LINE-COUNTER, PAGE-COUNTER, and all data-names defined within the Report Section.
4. One or more report group description entries must follow the report description entry.

PAGE-COUNTER Rules:

1. PAGE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the Report Section.
2. In the Report Section, a reference to PAGE-COUNTER can only appear in a SOURCE clause. Outside of the Report Section, PAGE-COUNTER may be used in any context in which a data-name with an integral value can appear.

RD SKELETON

3. If more than one PAGE-COUNTER exists in a program, PAGE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

   In the Report Section, an unqualified reference to PAGE-COUNTER is implicitly qualified by the name of the report in which the reference is made. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be explicitly qualified by that report-name.

4. Execution of the INITIATE statement causes the Report Writer Control System to set the PAGE-COUNTER of the referenced report to one (1).

5. PAGE-COUNTER is automatically incremented by one (1) each time the RWCS executes a page advance.

6. PAGE-COUNTER may be altered by Procedure Division statements.

LINE-COUNTER Rules:

1. LINE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the Report Section.

2. In the Report Section, a reference to LINE-COUNTER can only appear in a SOURCE clause. Outside of the Report Section, LINE-COUNTER may be used in any context in which a data-name with an integral value may appear. However, only the RWCS can change the contents of LINE-COUNTER.

3. If more than one LINE-COUNTER exists in a program, LINE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

   In the Report Section, an unqualified reference to LINE-COUNTER is implicitly qualified by the name of the report in which the reference is made. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER must be explicitly qualified by that report-name.

4. Execution of an INITIATE statement causes the RWCS to set the LINE-COUNTER of the referenced report to zero (0). The RWCS also automatically resets LINE-COUNTER to zero each time it executes a page advance.

5. The value of LINE-COUNTER is not affected by the processing of nonprintable report groups nor by the processing of a printable report group whose printing is suppressed by the SUPPRESS statement.

6. At the time each print line is presented, the value of LINE-COUNTER represents the line number on which the print line is presented. The value of LINE-COUNTER after the presentation of a report group is governed by the presentation rules for the report group.

REPORT GROUP SKELETON

Report Group Description - Complete Entry Skeleton

The report group description entry specifies the characteristics of a report group and of the individual items within a report group.

Format 1:

```
01 [ data-name-1 ]
   [ LINE NUMBER IS { integer-1 [ ON NEXT PAGE ] } ]
                    { PLUS integer-2             }
   [ NEXT GROUP IS { integer-3       } ]
                   { PLUS integer-4  }
                   { NEXT PAGE       }
                 { { REPORT HEADING } }
                 { { RH             } }
                 { { PAGE HEADING   } }
                 { { PH             } }
                 { { CONTROL HEADING } { data-name-2 } }
                 { { CH              } { FINAL       } }
       TYPE IS   { { DETAIL  } }
                 { { DE      } }
                 { { CONTROL FOOTING } { data-name-3 } }
                 { { CF              } { FINAL       } }
                 { { PAGE FOOTING   } }
                 { { PF             } }
                 { { REPORT FOOTING } }
                 { { RF             } }
   [ [ USAGE IS ] DISPLAY ] .
```

Format 2:

```
level-number [ data-name-1 ]
   [ LINE NUMBER IS { integer-1 [ ON NEXT PAGE ] } ]
                    { PLUS integer-2             }
   [ [ USAGE IS ] DISPLAY ] .
```

REPORT GROUP SKELETON

Format 3:

```
level-number [ data-name-1 ]
   [ BLANK WHEN ZERO ]
   [ GROUP INDICATE ]
   [ { JUSTIFIED } RIGHT ]
   [ { JUST      }       ]
 [ LINE NUMBER IS { integer-1 [ ON NEXT PAGE ] } ]
                  { PLUS integer-2             }
 [ COLUMN NUMBER IS integer-3 ]
   { PICTURE } IS character-string
   { PIC     }
```

```
[ [ SIGN is ] { LEADING  } SEPARATE CHARACTER ]
              { TRAILING }
```

REPORT GROUP SKELETON

```
{  SOURCE IS identifier-1                                          }
{  VALUE IS literal                                                }
{  {{ SUM { identifier-2 } ...                                     }
{     [ UPON { data-name-2 } ... ] } ...                           }
{     [ RESET ON { data-name-4 } ] ] }                             }
{                { FINAL     }                                     }
[  [ USAGE IS ] DISPLAY ] .
```

Syntax Rules:

1. The report group description entry can appear only in the Report Section.
2. Except for the data-name clause, which when present must immediately follow the level-number, the clauses may be written in any sequence.
3. In Format 2, the level-number may be any integer from 02 to 48, inclusive. In Format 3, the level-number may be any integer from 02 to 49, inclusive.
4. The description of a report group may consist of one, two, or three hierarchic levels:
   a. The first entry that describes a report group must be a Format 1 entry.
   b. Both Format 2 and Format 3 entries may be immediately subordinate to a Format 1 entry.
   c. A Format 3 entry may be immediately subordinate to a Format 2 entry.
   d. Format 3 entries must be elementary.
5. In a Format 1 entry, data-name-1 is required only when:
   a. A DETAIL report group is referenced by a GENERATE statement.
   b. A DETAIL report group is referenced by the UPON phrase of a SUM clause.
   c. A report group is referenced in a USE BEFORE REPORTING statement.
   d. The name of a CONTROL FOOTING report group is used to qualify a reference to a sum counter. If specified, data-name-1 may be referenced only by a GENERATE statement, the UPON phrase of a SUM clause, a USE BEFORE REPORTING sentence, or as a sum counter qualifier.
6. A Format 2 entry must contain at least one optional clause.
7. In a Format 2 entry, data-name-1 is optional. If present, it may be used only to qualify a sum counter reference.
8. In the Report Section, the USAGE clause is used only to declare the usage of printable items.

REPORT GROUP SKELETON

a. If the USAGE clause appears in a Format 3 entry, that entry must define a printable item.

b. If the USAGE clause appears in a Format 1 or Format 2 entry, at least one subordinate entry must define a printable item.

9. An entry that contains a LINE NUMBER clause must not have a subordinate entry that also contains a LINE NUMBER clause.

10. In Format 3:

a. A GROUP INDICATE clause may appear only in a TYPE DETAIL report group.

b. A SUM clause may appear only in a TYPE CONTROL FOOTING report group.

c. An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.

d. Data-name-1 is optional but may be specified in any entry. Data-name-1, however, may be referenced only if the entry defines a sum counter.

e. An entry that contains a SOURCE clause or a VALUE clause must also have a COLUMN NUMBER clause.

11. Table 16-1 shows all permissible clause combinations for a Format 3 entry. The table is read from left to right along the selected row.

REPORT GROUP SKELETON

Table 16-1. Permissible Clause Combinations in Format 3 Entries

PIC	COLUMN	SOURCE	SUM	VALUE	JUST	BLANK WHEN ZERO	GROUP INDICATE	USAGE	LINE OR SIGN
M			M						P
M	M		M			P		P	P
M	P	M			P		P	P	P
M	P	M				P	P	P	P
M	M			M	P		P	P	P

NOTES:

1. An 'M' indicates that the presence of the clause is mandatory.
2. A 'P' indicates that the presence of the clause is permitted, but not required.
3. A blank space indicates that the clause is not permitted.

General Rule:

Format 1 is the report group entry. The report group is defined by the contents of this entry and all of its subordinate entries.

Report Group Presentation Rules Tables:

A separate presentation rules table is given for each of the following types of report groups: REPORT HEADING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING. In addition, DETAIL report groups, CONTROL HEADING report groups, and CONTROL FOOTING report groups are treated jointly in the Body Group Presentation Rules Table. These tables and rules specify:

1. The permissible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report group
2. The requirements that are placed on the use of these clauses
3. The interpretation that the RWCS gives to these clauses

Columns 1 and 2 of a presentation rules table list all of the permissible combinations of LINE NUMBER and NEXT GROUP clauses for the designated report group TYPE. Consequently, to identify the set of presentation rules that apply

REPORT GROUP SKELETON

to a particular combination of LINE NUMBER and NEXT GROUP clauses, the presentation rules table is read from left to right along the selected row.

The applicable rules columns of a presentation rules table are partitioned into two parts. The first part specifies the rules that apply if the report description contains a PAGE clause, and the second part specifies the rules that apply if the PAGE clause is omitted. The purpose of the rules named in the applicable rules columns is detailed below.

1. Upper Limit Rules and Lower Limit Rules. These rules specify the vertical subdivisions of the page within which the specified report group may be presented.

   In the absence of a PAGE clause, the printed report is not considered to be partitioned into vertical subdivisions. Consequently, within the tables no upper limit rule and lower limit rule is specified for a report description in which the PAGE clause is omitted.

2. Fit Test Rules. The fit test rules are applicable only to body groups, and hence fit test rules are specified only within the Body Group Presentation Rules Table. At object time, the RWCS applies the fit test rules to determine whether the designated body group can be presented on the page to which the report is currently positioned.

   However, even for body groups there are no fit test rules when the PAGE clause is omitted from the report description entry.

3. First Print Line Position Rules. The first print line position rules specify where on the report medium the RWCS will present the first print line of the given report group.

   The presentation rules tables do not specify where on the report medium the RWCS will present the second and subsequent print lines (if any) of a report group. Certain general rules determine where the second and subsequent print lines of a report group will be presented. Refer to the LINE NUMBER clause general rules for this information.

4. Next Group Rules. The next group rules relate to the proper use of the NEXT GROUP clause.

5. Final LINE-COUNTER Setting Rules. The terminal values that the RWCS places in LINE-COUNTER after presenting report groups are specified by the final LINE-COUNTER setting rules.

LINE NUMBER Clause Notation:

Column 1 of the presentation rules tables uses a shorthand notation to describe the sequence of LINE NUMBER clauses that may appear in the description of a report group. The meaning of the abbreviations used in column 1 is as follows:

1. The letter 'A' represents one or more absolute LINE NUMBER clauses, none of which have the NEXT PAGE phrase, that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry.

2. The letter 'R' represents one or more relative LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry.

REPORT GROUP SKELETON

3. The letters 'NP' represent one or more absolute LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry, with the NEXT PAGE phrase appearing in the first, and only in the first, LINE NUMBER clause.

When two abbreviations appear together, they refer to a sequence of LINE NUMBER clauses that consists of the two specified consecutive sequences. For example, AR' refers to a report group description entry within which the 'A' sequence (defined in rule 1 above) is immediately followed by the 'R' sequence (defined in rule 2 above).

LINE NUMBER Clause Sequence Substitutions:

When 'AR' is shown to be a permissible sequence in the presentation rules tables, 'A' is also permissible and the same presentation rules are applicable.

When 'NP R' is shown to be a permissible sequence in the presentation rules tables, 'NP' is also permissible and the same presentation rules are applicable.

Saved Next Group Integer Description:

Saved next group integer is a data item that can be addressed only by the RWCS. When an absolute NEXT GROUP clause specifies a vertical positioning value that cannot be accommodated on the current page, the RWCS stores that value in saved next group integer. After page advance processing, the RWCS positions the next body group using the value stored in saved next group integer.

REPORT HEADING Group Presentation Rules:

Table 16-2 contains the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT HEADING report group.

1. Upper Limit Rule. The first line number on which the REPORT HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower Limit Rules.

   a. The last line number on which the REPORT HEADING report group can be presented is the line number that is obtained by subtracting one (1) from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

   b. The last line number on which the REPORT HEADING report group can be presented is the line number specified by integer-1 of the PAGE clause.

3. First Print Line Position Rules.

REPORT GROUP SKELETON

a. The first print line of the REPORT HEADING report group is presented on the line number specified by the integer of the first LINE NUMBER clause that is presented.

b. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the integer of the first LINE NUMBER clause that is presented and the value obtained by subtracting one (1) from the value of integer-2 of the HEADING phrase of the PAGE clause.

c. The REPORT HEADING report group is not presented.

REPORT GROUP SKELETON

Table 16-2. REPORT HEADING Group Presentation Rules

**		Applicable Rules***						
		If the PAGE clause is specified					If the PAGE clause is omitted	
Sequence of LINE NUMBER Clauses*	NEXT GROUP Clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R	Absolute	1	2a	3a	4a	5a	Illegal Combination+	
A R	Relative	1	2a	3a	4b	5b	Illegal Combination+	
A R	NEXT PAGE	1	2b	3a	4c	5c	Illegal Combination+	
A R		1	2a	3a		5d	Illegal Combination+	
R	Absolute	1	2a	3b	4a	5a	Illegal Combination++	
R	Relative	1	2a	3b	4b	5b	3d	5b
R	NEXT PAGE	1	2b	3b	4c	5c	Illegal Combination++	
R		1	2a	3b		5d	3d	5d
				3c		5e	3c	5e

* Refer to the LINE NUMBER Clause Notation for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

\+ Refer to the LINE NUMBER clause in this section.

++ Refer to the NEXT GROUP clause in this section.

REPORT GROUP SKELETON

d. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the contents of its LINE-COUNTER (in this case, zero) to the integer of the first LINE NUMBER clause.

4. Next Group Rules.

a. The NEXT GROUP integer must be greater than the line number on which the final print line of the REPORT HEADING report group is presented. In addition, the NEXT GROUP integer must be less than the line number specified by the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

b. The sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group is presented must be less than the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

c. NEXT GROUP NEXT PAGE signifies that the REPORT HEADING report group is to be presented entirely by itself on the first page of the report. The RWCS processes no other report group while positioned to the first page of the report.

5. Final LINE-COUNTER Setting Rules.

a. After the REPORT HEADING report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

b. After the REPORT HEADING report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

c. After the REPORT HEADING report group is presented, the RWCS places zero into LINE-COUNTER as the final LINE-COUNTER setting.

d. After the REPORT HEADING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the REPORT HEADING report group was presented.

e. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

REPORT GROUP SKELETON

PAGE HEADING Group Presentation Rules:

Table 16-3 contains the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE HEADING report group.

Table 16-3. PAGE HEADING Group Presentation Rules

**		Applicable Rules***				
		If the PAGE clause is specified****				
Sequence of LINE NUMBER Clauses*	NEXT GROUP Clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting
A R		1	2	3a		4a
R		1	2	3b		4a
				3c		4b

* Refer to the LINE NUMBER Clause Notation for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

**** If the PAGE clause is omitted from the report description entry, then a PAGE HEADING report group may not be defined. Refer to the TYPE clause.

1. Upper Limit Rules. If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the first line number on which the PAGE HEADING report group can be presented is one greater than the final LINE-COUNTER setting established by the REPORT HEADING.

   Otherwise, the first line number on which the PAGE HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

REPORT GROUP SKELETON

2. Lower Limit Rule. The last line number on which the PAGE HEADING report group can be presented is the line number that is obtained by subtracting one (1) from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

3. First Print Line Position Rules.

   a. The first print line of the PAGE HEADING report group is presented on the line number specified by the integer of the first LINE NUMBER clause that is presented.

   b. If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the sum of the final LINE-COUNTER setting established by the REPORT HEADING report group and the integer of the first LINE NUMBER clause that is presented in the PAGE HEADING report group defines the line number on which the first print line of the PAGE HEADING report group is presented.

      Otherwise, the sum of the integer of the first LINE NUMBER clause that is presented in the PAGE HEADING report group and the value obtained by subtracting one (1) from the value of integer-2 of the HEADING phrase of the PAGE clause defines the line number on which the first print line of the PAGE HEADING report group is presented.

   c. The PAGE HEADING report group is not presented.

4. Final LINE-COUNTER Setting Rules.

   a. The final LINE-COUNTER setting is the line number on which the final print line of the PAGE HEADING report group was presented.

   b. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

Body Group Presentation Rules:

Table 16-4 contains the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in CONTROL HEADING, DETAIL, and CONTROL FOOTING report groups.

1. Upper Limit Rule. The first line number on which a body group can be presented is the line number specified by the FIRST DETAIL phrase of the PAGE clause.

REPORT GROUP SKELETON

Table 16-4. Body Group Presentation Rules

**		Applicable Rules***							
		If the PAGE clause is specified						If the PAGE clause is omitted	
Sequence of LINE NUMBER Clauses*	NEXT GROUP Clause	Upper Limit	Lower Limit	Fit Test	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R	Absolute	1	2	3a	4a	5	6a	Illegal Combination+	
A R	Relative	1	2	3a	4a		6b	Illegal Combination+	
A R	NEXT PAGE	1	2	3a	4a		6c	Illegal Combination+	
A R		1	2	3a	4a		6d	Illegal Combination+	
R	Absolute	1	2	3b	4b	5	6a	Illegal Combination++	
R	Relative	1	2	3b	4b		6b	4d	6f
R	NEXT PAGE	1	2	3b	4b		6c	Illegal Combination++	
R		1	2	3b	4b		6d	4d	6d
NP R	Absolute	1	2	3c	4a	5	6a	Illegal Combination+	
NP R	Relative	1	2	3c	4a		6b	Illegal Combination+	
NP R	NEXT PAGE	1	2	3c	4a		6c	Illegal Combination+	
NP R		1	2	3c	4a		6d	Illegal Combination+	
					4c		6e	4c	6e

* Refer to the LINE NUMBER Clause Notation for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

\+ Refer to the LINE NUMBER clause in this section.

++ Refer to the NEXT GROUP clause in this section.

REPORT GROUP SKELETON

2. Lower Limit Rules. The last line number on which a CONTROL HEADING report group or DETAIL report group can be presented is the line number specified by the LAST DETAIL phrase of the PAGE clause.

   The last line number on which a CONTROL FOOTING report group can be presented is the line number specified by the FOOTING phrase of the PAGE clause.

3. Fit Test Rules.

   a. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, then the body group is presented on the page to which the report is currently positioned.

      Otherwise, the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. (See Final LINE-COUNTER Setting rule 6a.) If the saved next group integer was not so set, the body group is presented on the page to which the report is currently positioned. If the saved next group integer was so set, the RWCS moves the saved next group integer into LINE-COUNTER, resets the saved next group integer to zero, and reapplies fit test rule 3a.

   b. If a body group has been presented on the page to which the report is currently positioned, the RWCS computes a trial sum in a work location. The trial sum is computed by adding the contents of LINE-COUNTER to the integers of all LINE NUMBER clauses of the report group. If the trial sum is not greater than the body group's lower limit integer, then the report group is presented on the current page. If the trial sum exceeds the body group's lower limit integer, then the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS reapplies fit test rule 3b.

      If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. If the saved next group integer was not so set, the body group is presented on the page to which the report is currently positioned. If the saved next group integer was so set, the RWCS moves the saved next group integer into LINE-COUNTER, resets the saved next group integer to zero, and computes a trial sum in a work location.

      The trial sum is computed by adding the contents of LINE-COUNTER to the integer one (1) and the integers of all but the first LINE NUMBER clause of the body group. If the trial sum is not greater than the body group's lower limit integer, then the body group is presented on the current page. If the trial sum exceeds the body group's lower limit integer, then the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

REPORT GROUP SKELETON

c. If a body group has been presented on the page to which the report is currently positioned, the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS reapplies fit test rule 3c.

If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. If the saved next group integer was not so set, the body group is presented on the page to which the report is currently positioned. If the saved next group integer was so set, the RWCS moves the saved next group integer into LINE-COUNTER and resets the saved next group integer to zero. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, the body group is presented on the page to which the report is currently positioned. Otherwise, the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

4. First Print Line Position Rules.

a. The first print line of the body group is presented on the line number specified by the integer of the first LINE NUMBER clause that is presented.

b. If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if no body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line immediately following the line indicated by the value contained in LINE-COUNTER.

If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if a body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line that is obtained by adding the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause that is presented in the current body group.

If the value in LINE-COUNTER is less than the line number specified by the FIRST DETAIL phrase of the PAGE clause, then the first print line of the body group is presented on the line specified by the FIRST DETAIL phrase.

REPORT GROUP SKELETON

c. The body group is not presented.

d. The sum of the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause that is presented defines the line number on which the first print line is presented.

5. Next Group Rule. The integer of the absolute NEXT GROUP clause must specify a line number that is not less than that specified in the FIRST DETAIL phrase of the PAGE clause, and that is not greater than that specified in the FOOTING phrase of the PAGE clause.

6. Final LINE-COUNTER Setting Rules.

a. If the body group that has just been presented is a CONTROL FOOTING report group and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases, the RWCS makes a comparison of the line number on which the final print line of the body group was presented and the integer of the NEXT GROUP clause. If the former is less than the latter, then the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting. If the former is equal to or greater than the latter, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting; in addition, the RWCS places the NEXT GROUP integer into the saved next group integer location.

b. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases, the RWCS computes a trial sum in a work location. The trial sum is computed by adding the integer of the NEXT GROUP clause to the line number on which the final print line of the body group was presented. If the sum is less than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places that sum into LINE-COUNTER as the final LINE-COUNTER setting. If the sum is equal to or greater than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

REPORT GROUP SKELETON

c. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

   For all other cases, the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

d. The final LINE-COUNTER setting is the line number on which the final print line of the body group was presented.

e. LINE-COUNTER is unaffected by the processing of a nonprintable body group.

f. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

   For all other cases, the RWCS places the sum of the line number on which the final print line was presented and the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

PAGE FOOTING Presentation Rules:

Table 16-5 contains the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE FOOTING report group.

1. Upper Limit Rule. The first line number on which the PAGE FOOTING report group can be presented is the line number obtained by adding one (1) to the value of integer-5 of the FOOTING phrase of the PAGE clause.

REPORT GROUP SKELETON

Table 16-5. PAGE FOOTING Presentation Rules

**		Applicable Rules***				
		If the PAGE clause is specified****				
Sequence of LINE NUMBER Clauses*	NEXT GROUP Clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting
A R	Absolute	1	2	3a	4a	5a
A R	Relative	1	2	3a	4b	5b
A R		1	2	3a		5c
				3b		5d

* Refer to the LINE NUMBER Clause Notation for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

**** If the PAGE clause is omitted from the report description entry, then a PAGE FOOTING report group may not be defined. Refer to the TYPE clause.

2. Lower Limit Rule. The last line number on which the PAGE FOOTING report group can be presented is the line number specified by integer-1 of the PAGE clause.

3. First Print Line Position Rules.

   a. The first print line of the PAGE FOOTING report group is presented on the line specified by the integer of the first LINE NUMBER clause that is presented.

   b. The PAGE FOOTING report group is not presented.

REPORT GROUP SKELETON

4. Next Group Rules.

   a. The NEXT GROUP integer must be greater than the line number on which the final print line of the PAGE FOOTING report group is presented. In addition, the NEXT GROUP integer must not be greater than the line number specified by integer-1 of the PAGE clause.

   b. The sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group is presented must not be greater than the line number specified by integer-1 of the PAGE clause.

5. Final LINE-COUNTER Setting Rules.

   a. After the PAGE FOOTING report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

   b. After the PAGE FOOTING report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

   c. After the PAGE FOOTING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the PAGE FOOTING report group was presented.

   d. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

REPORT FOOTING Presentation Rules:

Table 16-6 contains the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT FOOTING report group.

1. Upper Limit Rules.

   a. If a PAGE FOOTING report group has been presented on the page to which the report is currently positioned, then the first line number on which the REPORT FOOTING report group can be presented is one greater than the final LINE-COUNTER setting established by the PAGE FOOTING report group.

      Otherwise, the first line number on which the REPORT FOOTING report group can be presented is the line number obtained by adding one (1) and the value of integer-5 of the PAGE clause.

REPORT GROUP SKELETON

Table 16-6. REPORT FOOTING Presentation Rules

**		Applicable Rules***						
		If the PAGE clause is specified					If the PAGE clause is omitted	
Sequence of LINE NUMBER Clauses*	NEXT GROUP Clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R		1a	2	3a		4a	Illegal Combination+	
R		1a	2	3b		4a	3d	4a
NP R		1b	2	3c		4a	Illegal Combination+	
				3e		4b	3e	4b

* Refer to the LINE NUMBER Clause Notation for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

\+ Refer to the LINE NUMBER clause in this section.

REPORT GROUP SKELETON

b. The first line number on which the REPORT FOOTING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

2. Lower Limit Rule. The last line number on which the REPORT FOOTING report group can be presented is the line number specified by integer-1 of the PAGE clause.

3. First Print Line Position Rules.

   a. The first print line of the REPORT FOOTING report group is presented on the line specified by the integer of the first LINE NUMBER clause that is presented.

   b. If a PAGE FOOTING report group has been presented on the page to which the report is currently positioned, then the sum of the final LINE-COUNTER setting established by the PAGE FOOTING report group and the integer of the first LINE NUMBER clause that is presented in the REPORT FOOTING report group defines the line number on which the first print line of the REPORT FOOTING report group is presented. Otherwise, the sum of the integer of the first LINE NUMBER clause that is presented in the REPORT FOOTING report group, and the line number specified by the value of integer-5 of the FOOTING phrase of the PAGE clause defines the line number on which the first print line of the REPORT FOOTING report group is presented.

   c. The NEXT PAGE phrase in the first absolute LINE NUMBER clause that is presented indicates that the REPORT FOOTING report group is to be presented on a page on which no other report group has been presented. The first print line of the REPORT FOOTING report group is presented on the line number specified by the integer of the first LINE NUMBER clause that is presented.

   d. The sum of the contents of LINE COUNTER and the integer of the first LINE NUMBER clause that is presented defines the line number on which the first print line is presented.

   e. The REPORT FOOTING report group is not presented.

4. Final LINE-COUNTER Setting Rules.

   a. The final LINE-COUNTER setting is the line number on which the final print line of the REPORT FOOTING report group is presented.

   b. LINE-COUNTER is unaffected by the processing of a nonprintable report group.

CODE

The CODE clause specifies a two-character literal that identifies each print line as belonging to a specific report.

General Format:

CODE literal-1

Syntax Rules:

1. Literal-1 is a two-character nonnumeric literal.
2. If the CODE clause is specified for any report in a file, it must be specified for all reports in the same file.

General Rules:

1. When the CODE clause is specified, literal-1 is automatically placed in the first two character positions of each Report Writer logical record
2. The positions occupied by literal-1 are not included in the description of the print line, but are included in the logical record size.

COLUMN NUMBER

The COLUMN NUMBER clause identifies a printable item and specifies the column number position of the item on a report line.

General Format:

```
[ COLUMN NUMBER IS integer-1 ]
```

Syntax Rules:

1. The COLUMN NUMBER clause can only be specified at the elementary level within a report group. The COLUMN NUMBER clause, if present, must appear in or be subordinate to an entry that contains a LINE NUMBER clause.

2. Within a given print line, the printable items must be defined in ascending column number order such that each character defined occupies a unique position.

General Rules:

1. The COLUMN NUMBER clause indicates that the object of a SOURCE clause or the object of a VALUE clause or the sum counter defined by a SUM clause is to be presented on the report line. The absence of a COLUMN NUMBER clause indicates that the entry is not to be presented on a report line.

2. Integer-1 specifies the column number of the leftmost character position of the printable item.

3. The Report Writer Control System (RWCS) supplies space characters for all positions of a print line that are not occupied by printable items.

4. The first position of the print line is considered to be column number 1.

The CONTROL clause establishes the levels of the control hierarchy for the report.

General Format:

```
[ CONTROL  IS ]  [ { data-name-1 } ...                          ]
[ CONTROLS ARE ] [ FINAL [ { data-name-1 } ... ]                ]
```

Syntax Rules:

1. Data-name-1, etc. must not be defined in the Report Section. Data-name-1, etc. may be qualified but must not be subscripted or indexed.

2. Each data-name must identify a different data item.

3. Data-name-1, etc. must not have a data item subordinate to it whose size is variable as defined in the OCCURS clause.

General Rules:

1. The data-names and the word FINAL specify the levels of the control hierarchy. FINAL, if specified, is the highest control, data-name-1 is the major control, the next recurrence of data-name-1 is an intermediate control, etc. The last recurrence of data-name-1 is the minor control.

2. The execution of the chronologically first GENERATE statement for a given report causes the RWCS to save the values of all control data items associated with that report. On subsequent executions of all GENERATE statements for that report, control data items are tested by the RWCS for a change of value. A change of value in any control data item causes a control break to occur. The control break is associated with the highest level for which a change of value is noted.

3. The Report Writer Control System tests for a control break by comparing the contents of each control data item with the prior contents saved from the execution of the previous GENERATE statement for the same report. The RWCS applies the inequality relation test described under "Relation Condition" in Section 6, as follows:

   a. If the control data item is a numeric data item, the relation test is for the comparison of two numeric operands.

   b. If the control data item is an index data item, the relation test is for the comparison of two index data items.

   c. If the control data item is a data item other than that described in general rules 3a and 3b above, the relation test is for the comparison of two nonnumeric operands.

   Refer to the PROGRAM COLLATING SEQUENCE phrase under "OBJECT-COMPUTER" in Section 6 for additional information.

4. The FINAL phrase is used when the most inclusive control group in the report is not associated with a control data-name.

The data-name clause specifies the name of the data being described.

General Format:

data-name

Syntax Rule:

In the Report Section, a data-name is not required in a data description entry; the word FILLER must not be used.

General Rules:

1. In the Report Section, data-name must be specified in the following cases:

   a. When the data-name represents a report group to be referred to by a GENERATE statement or a USE statement in the Procedure Division.

   b. When reference is to be made to the sum counter in the Procedure Division or the Report Section.

   c. When a DETAIL report group is referenced in the UPON phrase of the SUM clause.

   d. When the data-name is required to provide sum counter qualification.

GROUP INDICATE

The GROUP INDICATE clause specifies that the associated printable item is presented only on the first occurrence of its report group after a control break or page advance.

General Format:

GROUP INDICATE

Syntax Rules:

1. The GROUP INDICATE clause may only appear in a DETAIL report group entry that defines a printable item.

General Rules:

1. If a GROUP INDICATE clause is specified, it causes the SOURCE or VALUE clause to be ignored and spaces to be supplied, except in the following cases:
   a. On the first presentation of the DETAIL report group in the report.
   b. On the first presentation of the DETAIL report group after every page advance.
   c. On the first presentation of the DETAIL report group after every control break.
2. If neither a PAGE clause nor a CONTROL clause is specified in the report description entry, then a GROUP INDICATE printable item is presented the first time its DETAIL is presented after the INITIATE statement is executed. Thereafter, spaces are supplied for indicated items with SOURCE or VALUE clauses.

LINE NUMBER

The LINE NUMBER clause specifies vertical positioning information for its report group.

General Format:

```
LINE NUMBER IS { integer-1 [ ON NEXT PAGE ] }
               { PLUS integer-2             }
```

Syntax Rules:

1. Integer-1 and integer-2 must not exceed three significant digits in length.

   Neither integer-1 nor integer-2 may be specified in such a way as to cause any line of a report group to be presented outside of the vertical subdivision of the page designated for the report group type, as defined by the PAGE clause. Integer-2 may be zero.

2. Within a given report group description entry, an entry that contains a LINE NUMBER clause must not contain a subordinate entry that also contains a LINE NUMBER clause.

3. Within a given report group description entry, all absolute LINE NUMBER clauses must precede all relative LINE NUMBER clauses.

4. Within a given report group description entry, successive absolute LINE NUMBER clauses must specify integers that are in ascending order. The integers need not be consecutive.

5. If the PAGE clause is omitted from a given report description entry, only relative LINE NUMBER clauses can be specified in any report group description entry within that report.

6. Within a given report group description entry, a NEXT PAGE phrase can appear only once and, if present, must be in the first LINE NUMBER clause in that report group description entry.

7. A LINE NUMBER clause with the NEXT PAGE phrase can appear only in the description of body groups and in a REPORT FOOTING report group.

8. Every entry that defines a printable item (see the COLUMN NUMBER clause) must either contain a LINE NUMBER clause or be subordinate to an entry that contains a LINE NUMBER clause.

9. The first LINE NUMBER clause specified within a PAGE FOOTING report group must be an absolute LINE NUMBER clause.

LINE NUMBER

General Rules:

1. A LINE NUMBER clause must be specified to establish each print line of a report group.

2. The RWCS effects the vertical positioning specified by a LINE NUMBER clause before presenting the print line established by that LINE NUMBER clause.

3. Integer-1 specifies an absolute line number. An absolute line number specifies the line number on which the print line is presented.

4. Integer-2 specifies a relative line number. If a relative LINE NUMBER clause is not the first LINE NUMBER clause in the report group description entry, then the line number on which its print line is presented is determined by calculating the sum of the line number on which the previous print line of the report group was presented and integer-2 of the relative LINE NUMBER clause.

   If a relative LINE NUMBER clause is the first LINE NUMBER clause in the report group description entry, then the line number on which its print line is presented is determined by the rules stated in the Report Group Presentation Rules Tables in this section.

5. The NEXT PAGE phrase specifies that the report group is to be presented beginning on the indicated line number on a new page.

NEXT GROUP

The NEXT GROUP clause specifies information for vertical positioning of a page following the presentation of the last line of a report group.

General Format:

```
NEXT GROUP IS { integer-1          }
              { PLUS integer-2     }
              { NEXT PAGE          }
```

Syntax Rules:

1. A report group entry must not contain a NEXT GROUP clause unless the description of that report group contains at least one LINE NUMBER clause.
2. Integer-1 and integer-2 must not exceed three significant digits in length.
3. If the PAGE clause is omitted from the report description entry, only a relative NEXT GROUP clause may be specified in any report group description entry within that report.
4. The NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a PAGE FOOTING report group.
5. The NEXT GROUP clause must not be specified in a REPORT FOOTING report group or in a PAGE HEADING report group.

General Rules:

1. Any positioning of the page specified by the NEXT GROUP clause takes place after the presentation of the report group in which the clause appears. Refer to the Report Group Presentation Rules Tables in this section.
2. The vertical positioning information supplied by the NEXT GROUP clause is interpreted by the RWCS along with information from the TYPE and PAGE clauses, and the value in LINE-COUNTER, to determine a new value for LINE-COUNTER.
3. The NEXT GROUP clause is ignored by the RWCS when it is specified on a CONTROL FOOTING report group that is at a level other than the highest level at which a control break is detected.
4. The NEXT GROUP clause of a body group refers to the next body group to be presented, and therefore can affect the location at which the next body group is presented. The NEXT GROUP clause of a REPORT HEADING report group can affect the location at which the PAGE HEADING report group is presented. The NEXT GROUP clause of a PAGE FOOTING report group can affect the location at which the REPORT FOOTING report group is presented.

The PAGE clause defines the length of a page and the vertical subdivisions within which report groups are presented.

General Format:

```
PAGE [ LIMIT  IS  ]
     [ LIMITS ARE ] integer-1 [ LINE  ]
                              [ LINES ]
    [ HEADING integer-2 ]
    [ FIRST DETAIL integer-3 ]
    [ LAST DETAIL integer-4 ]
    [ FOOTING integer-5 ]
```

Syntax Rules:

1. The HEADING, FIRST DETAIL, LAST DETAIL, and FOOTING phrases may be written in any order.
2. Integer-1 must not exceed three significant digits in length.
3. Integer-2 must be greater than or equal to one (1).
4. Integer-3 must be greater than or equal to integer-2.
5. Integer-4 must be greater than or equal to integer-3.
6. Integer-5 must be greater than or equal to integer-4.
7. Integer-1 must be greater than or equal to integer-5.
8. The following rules indicate the vertical subdivision of the page in which each TYPE of report group may appear when the PAGE clause is specified. Also, see the Page Regions Table which follows.

   a. A REPORT HEADING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

      A REPORT HEADING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

   b. A PAGE HEADING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

   c. A CONTROL HEADING or DETAIL report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-4, inclusive.

d. A CONTROL FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-5, inclusive.

e. A PAGE FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

f. A REPORT FOOTING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

A REPORT FOOTING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

9. All report groups must be described so that they can be presented on one page. The RWCS never splits a multiline report group across page boundaries.

General Rules:

1. The vertical format of a report page is established using the integer values specified in the PAGE clause.

a. Integer-1 defines the size of a report page by specifying the number of lines available on each page.

b. HEADING integer-2 defines the first line number on which a REPORT HEADING or PAGE HEADING report group may be presented.

c. FIRST DETAIL integer-3 defines the first line number on which a body group may be presented. REPORT HEADING and PAGE HEADING report groups may not be presented on or beyond the line number specified by integer-3.

d. LAST DETAIL integer-4 defines the last line number on which a CONTROL HEADING or DETAIL report group may be presented.

e FOOTING integer-5 defines the last line number on which a CONTROL FOOTING report group may be presented. PAGE FOOTING and REPORT FOOTING report groups must follow the line number specified by integer-5.

2. If the PAGE clause is specified, the following implicit values are assumed for any omitted phrases:

a. If the HEADING phrase is omitted, a value of one (1) is assumed for integer-2.

b. If the FIRST DETAIL phrase is omitted, a value equal to integer-2 is given to integer-3.

c. If the LAST DETAIL and the FOOTING phrases are both omitted, the value of integer-1 is given to both integer-4 and integer-5.

d. If the FOOTING phrase is specified and the LAST DETAIL phrase is omitted, the value of integer-5 is given to integer-4.

e. If the LAST DETAIL phrase is specified and the FOOTING phrase is omitted, the value of integer-4 is given to integer-5.

3. If the PAGE clause is omitted, the report consists of a single page of indefinite length.

4. The presentation rules for each type of report group are contained in the Report Group Presentation Rules Tables in this section.

5. The page regions established by the PAGE clause are described in Table 16-7.

Table 16-7. Page Regions

Report Groups That May Be Presented in the Region	First Line Number of the Region	Last Line Number of the Region
REPORT HEADING described with NEXT GROUP NEXT PAGE REPORT FOOTING described with LINE integer-1 NEXT PAGE	integer-2	integer-1
REPORT HEADING not described with NEXT GROUP NEXT PAGE PAGE HEADING	integer-2	integer-3 minus 1
CONTROL HEADING DETAIL	integer-3	integer-4
CONTROL FOOTING	integer-3	integer-5
PAGE FOOTING REPORT FOOTING not described with LINE integer-1 NEXT PAGE	integer-5 plus 1	integer-1

REPORT

The REPORT clause specifies the names of reports that make up a report file.

General Format:

```
{ REPORT  IS  }
{ REPORTS ARE } { report-name-1 } ...
```

Syntax Rules:

1. Each report-name specified in a REPORT clause must be the subject of a report description entry in the Report Section. The order of appearance of the report-names is not significant.
2. A report-name must appear in only one REPORT clause.
3. The subject of a file description entry that specifies a REPORT clause may only be referred to by the OPEN OUTPUT, OPEN EXTEND, and CLOSE statements.

General Rule:

The presence of more than one report-name in a REPORT clause indicates that the file contains more than one report.

SOURCE

The SOURCE clause identifies the sending data item that is moved to an associated printable item defined within a report group description entry.

General Format:

SOURCE IS identifier-1

Syntax Rules:

1. Identifier-1 may be defined in any section of the Data Division. If identifier-1 is a Report Section data item, it can only be:

   a. PAGE-COUNTER, or

   b. LINE-COUNTER, or

   c. A sum counter of the report within which the SOURCE clause appears.

2. Identifier-1 specifies the sending data item of the implicit MOVE statement that the RWCS will execute to move identifier-1 to the printable item. Identifier-1 must be defined so that it conforms to the rules for sending items in the MOVE statement.

General Rule:

The RWCS formats the print lines of a report group just prior to presenting the report group. (Refer to the TYPE clause.) At this time, the implicit MOVE statements specified by SOURCE clauses are executed by the RWCS.

SUM

The SUM clause establishes a sum counter and names the data items to be summed.

General Format:

```
{ SUM { identifier-1 } ...
    [ UPON { data-name-1 } ... ] } ...
    [ RESET ON { data-name-2 } ]
               { FINAL       }
```

Syntax Rules:

1. Identifier-1, etc. must be defined as numeric data items. When defined in the Report Section, identifier-1, etc. must be the names of sum counters.

   If the UPON phrase is omitted, any identifiers in the associated SUM clause which are themselves sum counters must be defined either in the same report group that contains this SUM clause or in a report group which is at a lower level in the control hierarchy of this report.

   If the UPON phrase is specified, any identifiers in the associated SUM clause must not be sum counters.

2. Data-name-1, etc. must be the names of DETAIL report groups described in the same report as the CONTROL FOOTING report group in which the SUM clause appears. Data-name-1, etc. may be qualified by a report-name.

3. A SUM clause can appear only in the description of a CONTROL FOOTING report group.

4. Data-name-2 must be one of the data-names specified in the CONTROL clause for this report. Data-name-2 must not be a lower level control than the associated control for the report group in which the RESET phrase appears.

   FINAL, if specified in the RESET phrase, must also appear in the CONTROL clause for this report.

5. The highest permissible qualifier of a sum counter is the report-name.

General Rules:

1. The data item that is the subject of the Data Description entry in which the SUM clause appears is a sum counter. At object program execution, the RWCS adds directly into the sum counter each of the values contained in identifier-1, etc. This addition is performed under the rules of the ADD statement.

2. Only one sum counter exists for an elementary report entry regardless of the number of SUM clauses specified in the elementary report entry.

3. If the elementary report entry for a printable item contains a SUM clause, the sum counter serves as a source data item. The RWCS moves the data contained in the sum counter, according to the rules of the MOVE statement, to the printable item for presentation.

4. If a data-name appears as the subject of an elementary report entry that contains a SUM clause, the data-name is the name of the sum counter; the data-name is not the name of the printable item that the entry may also define.

5. It is permissible for Procedure Division statements to alter the contents of sum counters.

6. Addition of the identifiers into sum counters is performed by the RWCS during the execution of GENERATE and TERMINATE statements. The three categories of sum counter incrementing are called subtotalling, crossfooting, and rolling forward. Subtotalling is accomplished during execution of GENERATE statements only, after any control break processing but before processing of the DETAIL report group. Crossfooting and rolling forward are accomplished during the processing of CONTROL FOOTING report groups.

7. The UPON phrase provides the capability to accomplish selective subtotalling for the DETAIL report groups named in the phrase.

8. The RWCS adds each individual addend into the sum counter at a time that depends upon the characteristics of the addend.

   a. When the addend is a sum counter defined in the same CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed crossfooting.

      Crossfooting occurs when a control break takes place and at the time the CONTROL FOOTING report group is processed.

      Crossfooting is performed according to the sequence in which sum counters are defined within the CONTROL FOOTING report group. That is, all crossfooting into the first sum counter defined in the CONTROL FOOTING report group is completed, and then all crossfooting into the second sum counter defined in the CONTROL FOOTING report group is completed. This procedure is repeated until all crossfooting operations are completed.

      When one of the addends is the sum counter defined by the Data Description entry in which the SUM clause appears, the initial value of that sum counter at the time of summation is used in the summing operation.

   b. When the addend is a sum counter defined in a lower level CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed rolling forward. A sum counter in a lower level CONTROL FOOTING report group is rolled forward when a

control break occurs and at the time that the lower level CONTROL FOOTING report group is processed.

c. When the addend is not a sum counter, the accumulation into a sum counter of such an addend is called subtotalling. If the SUM clause contains the UPON phrase, the addends are subtotalled when a GENERATE statement for the designated DETAIL report group is executed. If the SUM clause does not contain the UPON phrase, the addends which are not sum counters are subtotalled when any GENERATE data-name statement is executed for the report in which the SUM clause appears.

9. If two or more of the identifiers specify the same addend, then the addend is added into the sum counter as many times as the addend is referenced in the SUM clause. It is permissible for two or more of the data-names to specify the same DETAIL report group. When a GENERATE data-name statement for such a DETAIL report group is given, the incrementing occurs repeatedly, as many times as data-name appears in the UPON phrase.

10. For the subtotalling that occurs when a GENERATE report-name statement is executed, refer to the GENERATE statement in this section.

11. In the absence of an explicit RESET phrase, the RWCS will set a sum counter to zero when the RWCS is processing the CONTROL FOOTING report group within which the sum counter is defined. If an explicit RESET phrase is specified, then the RWCS will set the sum counter to zero when the RWCS is processing the designated level of the control hierarchy. Refer to the TYPE clause.

Sum counters are initially set to zero by the RWCS during the execution of the INITIATE statement for the report containing the sum counter

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be processed by the Report Writer Control System (RWCS).

General Format:

```
                  { { REPORT HEADING }                                  }
                  { { RH             }                                  }
                  {                                                     }
                  { { PAGE HEADING   }                                  }
                  { { PH             }                                  }
                  {                                                     }
                  { { CONTROL HEADING } { data-name-1 }                 }
                  { { CH             } { FINAL        }                 }
TYPE IS           {                                                     }
                  { { DETAIL         }                                  }
                  { { DE             }                                  }
                  {                                                     }
                  { { CONTROL FOOTING } { data-name-2 }                 }
                  { { CF             } { FINAL        }                 }
                  {                                                     }
                  { { PAGE FOOTING   }                                  }
                  { { PF             }                                  }
                  {                                                     }
                  { { REPORT FOOTING }                                  }
                  { { RF             }                                  }
```

Syntax Rules:

1. RH is an abbreviation for REPORT HEADING.
   PH is an abbreviation for PAGE HEADING.
   CH is an abbreviation for CONTROL HEADING.
   DE is an abbreviation for DETAIL.
   CF is an abbreviation for CONTROL FOOTING.
   PF is an abbreviation for PAGE FOOTING.
   RF is an abbreviation for REPORT FOOTING.

2. REPORT HEADING, PAGE HEADING, CONTROL HEADING FINAL, CONTROL FOOTING FINAL, PAGE FOOTING, and REPORT FOOTING report groups may each appear no more than once in the description of a report.

3. PAGE HEADING and PAGE FOOTING report groups may be specified only if a PAGE clause is specified in the corresponding report description entry.

4. Data-name-1, data-name-2, and FINAL, if present, must be specified in the CONTROL clause of the corresponding report description entry. At most, one CONTROL HEADING report group and one CONTROL FOOTING report group can be specified for each data-name or FINAL in the CONTROL clause of the report description entry. However, neither a CONTROL HEADING report group nor a CONTROL FOOTING report group is required for a data-name or FINAL specified in the CONTROL clause of the report description entry.

5. In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups SOURCE clauses, SUM clauses and USE statements must not reference any of the following:

   a. Group data items containing a control data item.

   b. Data items subordinate to a control data item.

   c. A redefinition or renaming of any part of a control data item.

   In PAGE HEADING and PAGE FOOTING report groups: SOURCE clauses and USE statements must not reference control data names.

6. When a GENERATE report name statement is specified in the Procedure Division, the corresponding report description entry must include no more than one DETAIL report group. If no GENERATE data-name statements are specified for such a report, a DETAIL report group is not required.

7. The description of a report must include at least one body group.

General Rules:

1. DETAIL report groups are processed by the RWCS as a direct result of GENERATE statements. If a report group is other than TYPE DETAIL, its processing is an automatic RWCS function.

2. The REPORT HEADING phrase specifies a report group that is processed by the RWCS only once per report, as the first report group of that report. The REPORT HEADING report group is processed during the execution of the chronologically first GENERATE statement for that report.

3. The PAGE HEADING phrase specifies a report group that is processed by the RWCS as the first report group on each page of that report except under the following conditions:

   a. A PAGE HEADING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.

   b. A PAGE HEADING report group is processed as the second report group on a page when it is preceded by a REPORT HEADING report group that is not to be presented on a page by itself.

   Refer to the Report Group Presentation Rules Tables in this section for further information.

4. The CONTROL HEADING phrase specifies a report group that is processed by the RWCS at the beginning of a control group for a designated control data-name or, in the case of FINAL, is processed during the execution of the chronologically first GENERATE statement for that report. During the execution of any GENERATE statement at which the RWCS detects a control break, any CONTROL HEADING report groups associated with the highest control level of the break and lower levels are processed.

5. The DETAIL phrase specifies a report group that is processed by the RWCS when a corresponding GENERATE statement is executed

6. The CONTROL FOOTING phrase specifies a report group that is processed by the RWCS at the end of a control group for a designated control data-name.

   In the case of FINAL, the CONTROL FOOTING report group is processed only once per report as the last body group of that report. During the execution of any GENERATE statement in which the RWCS detects a control break, any CONTROL FOOTING report group associated with the highest level of the control break or more minor levels is presented. All CONTROL FOOTING report groups are presented during the execution of the TERMINATE statement if at least one GENERATE statement has been executed for the report.

7. The PAGE FOOTING phrase specifies a report group that is processed by the RWCS as the last report group on each page except under the following conditions:

   a. A PAGE FOOTING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.

   b. A PAGE FOOTING report group is processed as the second to last report group on a page when it is followed by a REPORT FOOTING report group that is not to be processed on a page by itself.

8. The REPORT FOOTING phrase specifies a report group that is processed by the RWCS only once per report and as the last report group of that report. The REPORT FOOTING report group is processed during the execution of a corresponding TERMINATE statement, if at least one GENERATE statement has been executed for the report.

9. The sequence of steps executed by the RWCS when it processes a REPORT HEADING, PAGE HEADING, CONTROL HEADING, PAGE FOOTING, or REPORT FOOTING report group is described below.

   a. If a USE BEFORE REPORTING procedure that references the data name of the report group is present, the USE procedure is executed.

   b. If a SUPPRESS statement has been executed or if the report group is not printable, no further processing is done for the report group.

   c. Otherwise, the RWCS formats the print lines and presents the report group according to the presentation rules for that type of report group.

10. The sequence of steps executed by the RWCS when it processes a CONTROL FOOTING report group is described below.

    The GENERATE rules specify that when a control break occurs, the RWCS produces the CONTROL FOOTING report groups beginning at the minor level, and proceeding upwards through the level at which the highest control break was sensed. In this regard, it should be noted that even though no CONTROL FOOTING report group has been defined for a given control data-name, the RWCS will still have to execute the step described in general rule 10f below if a RESET phrase within the report description specifies that control data-name.

a. Sum counters are crossfooted; that is, all sum counters defined in this report group that are operands of SUM clauses in the same report group are added to their sum counters.

b. Sum counters are rolled forward; that is, all sum counters defined in the report group that are operands of SUM clauses in higher level CONTROL FOOTING report groups are added to the higher level sum counters.

c. If a USE BEFORE REPORTING procedure that references the data-name of the report group is present, the USE procedure is executed.

d. If a SUPPRESS statement has been executed or if the report group is not printable, the RWCS next executes the step described in general rule 10f below.

e. Otherwise, the RWCS formats the print lines and presents the report group according to the presentation rules for CONTROL FOOTING report groups.

f. The RWCS then resets those sum counters that are to be reset when the RWCS processes this level in the control hierarchy.

11. The DETAIL report group processing that the RWCS executes in response to a GENERATE data-name statement is described in general rules 11a through 11e.

When the description of a report includes exactly one DETAIL report group, the detail related processing that the RWCS executes in response to a GENERATE report-name statement is described in general rules 11a through 11d. These steps are performed as though a GENERATE data-name statement were being executed.

When the description of a report includes no DETAIL report groups, the detail related processing that the RWCS executes in response to a GENERATE report-name statement is described in general rule 11a. This step is performed as though the description of the report included exactly one DETAIL report group, and a GENERATE data-name statement were being executed.

a. The RWCS performs any subtotalling that has been designated for the DETAIL report group.

b. If a USE BEFORE REPORTING procedure that refers to the data-name of the report group is present, the USE procedure is executed.

c. If a SUPPRESS statement has been executed or if the report group is not printable, no further processing is done for the report group.

d. If the DETAIL report group is being processed as a result of a GENERATE report-name statement, no further processing is done for the report group.

e. Otherwise, the RWCS formats the print lines and presents the report group according to the presentation rules for DETAIL report groups.

12. When the RWCS is processing a CONTROL HEADING, CONTROL FOOTING, or DETAIL report group, as described in general rules 9, 10, and 11, the RWCS may have to interrupt the processing of that body group after determining that the body group is to be presented, and execute a page advance (and process PAGE FOOTING and PAGE HEADING report groups) before actually presenting the body group.

13. During control break processing, the values of control data items that the RWCS used to detect a given control break are referred to as prior values.

    a. During control break processing of a CONTROL FOOTING report group, any references to control data items in a USE procedure or SOURCE clause associated with that CONTROL FOOTING report group are supplied with prior values.

    b. When a TERMINATE statement is executed, the RWCS makes the prior control data item values available to SOURCE clause or USE procedure references in CONTROL FOOTING and REPORT FOOTING report groups as though a control break had been detected in the highest control data-name.

    c. All other data item references within report groups and their USE procedures access the current values that are contained within the data items at the time the report group is processed.

PROCEDURE DIVISION FOR THE REPORT WRITER

The GENERATE statement directs the RWCS to produce a report in accordance with the report description that was specified in the Report Section of the Data Division.

General Format:

GENERATE { data-name / report-name }

Syntax Rules:

1. The data-name must name a TYPE DETAIL report group and may be qualified by a report-name.

2. Report-name may be used only if the referenced report description contains all of the following:

   a. A CONTROL clause

   b. Not more than one DETAIL report group

   c. At least one body group

General Rules:

1. In response to a GENERATE report-name statement, the RWCS performs summary processing. If all of the GENERATE statements that are executed for a report are of the form GENERATE report-name, the report that is produced is called a summary report. A summary report is one in which no DETAIL report group is presented.

2. In response to a GENERATE data-name statement, the RWCS performs detail processing that includes specific processing for the DETAIL report group designated by the GENERATE statement. Normally, the execution of a GENERATE data-name statement causes the RWCS to present the designated DETAIL report group.

3. During the execution of the chronologically first GENERATE statement for a given report, the RWCS saves the values within the control data items. During the execution of the second and subsequent GENERATE statements for the same report, and until a control break is detected, the RWCS utilizes this set of control values to determine whether a control break has occurred. When a control break occurs, the RWCS saves the new set of control values, which it thereafter uses to sense for a control break until another control break occurs.

4. During report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS must advance the report to a new page in order to present a body group. Refer to the Report Group Presentation Rules Tables in this section.

5. When the chronologically first GENERATE statement for a given report is executed, the RWCS processes (in order) the report groups that are named below, provided that such report groups are defined within the report description. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in general rule 4. Refer to the TYPE clause for a description of the actions taken by the RWCS when it processes each type of report group.

   a. The REPORT HEADING report group is processed.

   b. The PAGE HEADING report group is processed.

   c. All CONTROL HEADING report groups are processed from major to minor.

   d. If a GENERATE data-name statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name statement is being executed, certain steps that are involved in the processing of a DETAIL report group are performed.

6. When a GENERATE statement other than the chronologically first is executed for a given report, the RWCS performs the steps enumerated below, as applicable. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in general rule 4. Refer to the TYPE clause for a description of the actions taken by the RWCS when it processes each type of report group.

   a. Sense for control break. The rules for determining the equality of control data items are the same as those specified for relation conditions. If a control break has occurred, then:

      - Enable the CONTROL FOOTING USE procedures and CONTROL FOOTING SOURCE clauses to access the control data item values that are given in the description of the TYPE clause.

      - Process the CONTROL FOOTING report groups in the order minor to major. Only CONTROL FOOTING report groups that are not more major than the highest level at which a control break occurred are processed.

      - Process the CONTROL HEADING report groups in the order major to minor. Only the CONTROL HEADING report groups that are not more major than the highest level at which a control break occurred are processed.

   b. If a GENERATE data-name statement is being executed, process the designated DETAIL report group. If a GENERATE report-name statement is being executed, certain of the steps that are involved in the processing of a DETAIL report group are performed.

7. GENERATE statements for a report can be executed only after an INITIATE statement for the report has been executed and before a TERMINATE statement for the report has been executed.

INITIATE

The INITIATE statement causes the Report Writer Control System (RWCS) to begin processing a report.

General Format:

```
INITIATE { report-name } ...
```

Syntax Rule:

Each report-name must be defined by a report description entry in the Report Section of the Data Division.

General Rules:

1. The INITIATE statement performs the following initialization functions for each named report:

   a. All sum counters are set to zero.

   b. LINE-COUNTER is set to zero.

   c. PAGE-COUNTER is set to one (1).

2. The INITIATE statement does not open the file with which the report is associated. Therefore, an OPEN statement with either the OUTPUT phrase or the EXTEND phrase for the file must be executed prior to the execution of the INITIATE statement.

3. A subsequent INITIATE statement for a particular report-name must not be executed unless an intervening TERMINATE statement has been executed for that report-name.

4. If more than one report-name is specified in an INITIATE statement, the result of executing this INITIATE statement is the same as if a separate INITIATE statement had been written for each report-name in the same order as specified in the INITIATE statement.

SUPPRESS

The SUPPRESS statement causes the Report Writer Control System (RWCS) to inhibit the presentation of a report group.

General Format:

SUPPRESS PRINTING

Syntax Rule:

The SUPPRESS statement may only appear in a USE BEFORE REPORTING procedure.

General Rules:

1. The SUPPRESS statement inhibits presentation only for the report group named in the USE procedure within which the SUPPRESS statement appears.
2. The SUPPRESS statement must be executed each time the presentation of the report group is to be inhibited.
3. When the SUPPRESS statement is executed, the RWCS is instructed to inhibit processing the following report group functions:
   a. The presentation of the print lines of the report group.
   b. The processing of all LINE clauses in the report group.
   c. The processing of the NEXT GROUP clause in the report group.
   d. The adjustment of LINE-COUNTER.

TERMINATE

The TERMINATE statement causes the Report Writer Control System (RWCS) to complete processing the specified reports.

General Format:

```
TERMINATE { report-name } ...
```

Syntax Rule:

Each report-name given in a TERMINATE statement must be defined by an RD entry in the Report Section of the Data Division.

General Rules:

1. The TERMINATE statement causes the RWCS to produce all the CONTROL FOOTING report groups beginning with the minor CONTROL FOOTING report group. Then the REPORT FOOTING report group is produced. The RWCS makes the prior set of control data item values available to the CONTROL FOOTING and REPORT FOOTING SOURCE clauses and USE procedures, as though a control break had been sensed in the most major control data-name.

2. If no GENERATE statements have been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for that report, the TERMINATE statement does not cause the RWCS to produce any report groups or perform any of the related processing.

3. During report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS must advance the report to a new page for the purpose of presenting a body group. Refer to the Report Group Presentation Rules Tables in this section.

4. The TERMINATE statement cannot be executed for a report unless the TERMINATE statement was chronologically preceded by an INITIATE statement for that report and for which no TERMINATE statement has yet been executed.

5. If more than one report-name is specified in a TERMINATE statement, the result of executing the TERMINATE statement is the same as if a separate TERMINATE statement had been written for each report-name in the same order as specified in the TERMINATE statement.

6. The TERMINATE statement does not close the file with which the report is associated; a CLOSE statement for the file must be executed. Every report in a file that is in an initiated condition must be terminated before a CLOSE statement is executed for that file.

USE BEFORE REPORTING

The USE BEFORE REPORTING statement specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

General Format:

USE BEFORE REPORTING data-name.

Syntax Rules:

1. A USE statement, when present, must immediately follow a section header in the declarative 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. Data-name, which may be qualified, represents a report group and must not appear in more than one USE statement.

   The GENERATE, INITIATE, or TERMINATE statements must not appear in a paragraph within a USE BEFORE REPORTING procedure.

   A USE BEFORE REPORTING procedure must not alter the value of any control data item.

3. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

General Rules:

1. The designated procedures are executed by the RWCS just before the named report group is produced. Refer to the TYPE clause.

2. Within a USE procedure, no reference must be made to any nondeclarative procedures. Conversely, in the nondeclarative portion, no reference must be made to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE BEFORE REPORTING statement or to the procedures associated with such a USE statement.

APPENDIX A

RESERVED WORDS

Reserved words are terms that may be used in COBOL source programs but must not appear in the programs as user-defined words or system names. They can be used only as indicated in the general formats. The seven types of reserved words are:

1. Key Words - Words whose presence are required when the format in which the words appear are used in a source program.
2. Optional Words - Within each format, uppercase words that are not underlined are called optional words and may be specified at the discretion of the user.
3. Connectives - Qualifier connectives (OF, IN); series connectives (separator comma or separator semicolon); and logical connectives (AND, OR).
4. Special Registers - Certain reserved words used to name and reference special registers (areas used to store data produced in conjunction with specific COBOL features).
5. Figurative Constants - Certain reserved words used to name and reference specific constant values.
6. Special-Character Words - The arithmetic operators and relation characters.

The following reserved word list includes words that are reserved for Honeywell computers other than the Series 60 Level 68 Multics systems. They are included to help the user avoid problems when using a source program written for Multics COBOL with COBOL compilers on other non-Multics Honeywell systems.

```
=
(
)
*
**
+
-
.
/
<
==
>

ACCEPT
ACCESS
ADD
ADDRESS
ADVANCING
AFTER
AGGREGATE
ALL
ALPHABET
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTER
ALTERING
ALTERNATE
AND
ANY
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
ATTACH
ATTACH-OPTIONS
AUTHOR

BECOMES
BEFORE
BEGINNING
BIT
BITS
BLANK
BLOCK
BOOLEAN
BOTTOM
BY

CALL
CANCEL
CATALOG-NAME
CATALOG-NAMES
CATALOGUE-NAME
CATALOGUE-NAMES
CATALOGUED
CD
CF
CH
CHAR
CHARACTER
CHARACTERS
CHECK
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMP-5
COMP-6
COMP-7
COMP-8
COMPL
COMPLEMENTARY
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTATIONAL-5
COMPUTATIONAL-6
COMPUTATIONAL-7
COMPUTATIONAL-8
COMPUTE
CONCURRENT
CONFIGURATION
CONNECT
CONSTANT
CONTAINS
CONTINUE
CONTROL
CONTROLS
CONVERSION
COPIES
COPY
CORR
CORRESPONDING
COUNT
CS-BASIC
CS-GENERAL
CURRENCY
CURRENT

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DB
DB-EXCEPTION
DB-KEY
DB-PRIVACY-KEY
DB-REALM-NAME
DB-RECORD-NAME
DB-SET-NAME
DB-STATUS
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-NUMERIC-CONTENTS
DEBUG-SUB 1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DEFAULT
DELETE
DELIMITED
DELIMITER
DENSITY
DEPENDING
DESCENDING
DESCRIPTORS
DESTINATION
DETACH
DETAIL
DEVICE
DIRECT-REFERENCE
DISABLE
DISCONNECT
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATE
DUPLICATES
DYNAMIC

EGI
ELSE
EMI
EMPTY
ENABLE
END
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DIVIDE
END EVALUATE
END-IF
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END SUBTRACT
END-UNSTRING
END-WRITE
ENDING
ENTER
ENTRY
ENVIRONMENT
EOP
EQUAL
EQUALS
ERASE
ERROR
ESI
EVALUATE
EVERY
EXAMINE
EXCEEDS
EXCEPTION
EXCLUSIVE
EXIT
EXT
EXTEND
EXTERNAL

FD
FILE
FILE-CONTROL
FILES
FILLER
FINAL
FIND
FINISH
FIRST
FOOTING
FOR
FORMAT
FROM

GENERATE
GENERATION
GET
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INCLUDING
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSIDE
INSPECT
INSTALLATION
INTERCHANGE
INTO
INVALID
```

INVOKING
IS

JUST
JUSTIFIED

KEY
KEY-LOCATION
KEYED

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

MAXIMUM
MEMBER
MEMBERS
MEMBERSHIP
MEMORY
MERGE
MESSAGE
MODE
MODIFICATION
MODIFY
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
NOT
NULL
NUMBER
NUMERIC
NUMERIC-EDITED

OBJECT
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPERATIONAL
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OUTPUT_MODE
OVERFLOW
OWNER

PADDING
PAGE
PAGE-COUNTER
PATH
PERFORM
PERMANENT
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PREATTACHED
PREFIX
PRINT
PRINTING
PRIOR
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PURGE

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
READY
REALM
REALM-ID
REALM-NAME
REALMS
RECEIVE
RECORD
RECORD-NAME
RECORDING
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACE
REPLACEMENT
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETAIN
RETAINING
RETRIEVAL
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SCALAR
SD
SEARCH
SECONDARY
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SELECTION
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SETS
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPANNED
SPECIAL-NAMES
SS
SSF
STANDARD
STANDARD-1
STANDARD-2
START
STATION
STATUS
STOP
STORE
STREAM
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUB-SCHEMA
SUBSTITUTE
SUBSTITUTION
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SUSPEND
SYMBOLIC
SYNC
SYNCHRONIZED
SYSTEM

TABLE
TALLY
TALLYING
TAPE
TAPE-OPTIONS
TEMP
TEMPORARY
TENANT
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIMES
TITLE
TO
TOP
TRAILING
TRANSFORM
TYPE

UNEQUAL
UNIT
UNSTRING
UNTIL
UP
UPDATE
UPON
USAGE
USAGE-MODE
USE
USING

VALUE
VALUES
VARYING
VIA
VIRTUAL

WHEN
WITH
WITHIN
WORDS
WORKING-STORAGE
WRITE

APPENDIX B

GLOSSARY

Appendix B contains terms that are defined in accordance with their meaning as used in this manual describing Multics COBOL and may not have the same meaning in other languages.

These definitions are intended for either reference material or introductory material to be reviewed prior to reading the preceding sections in this manual. Therefore, the definitions are generally brief and do not include detailed syntactical rules.

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.

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

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

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

Alphabetic Character. A character that belongs to the following set of both uppercase and lowercase 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; 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 ASCII 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 or 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 caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

Arithmetic Operator. A single character, or a fixed two-character combination, that belongs to the following set:

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

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

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

AT END Condition. A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.
2. During the execution of a RETURN statement, when no next logical record exists for the associated sort file.
3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

Block. A physical unit of data that is normally composed of one or more logical records. For random mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical records that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Body Group. Generic name for a report group of TYPE DETAIL, CONTROL HEADING, or CONTROL FOOTING.

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 that executes a CALL to another program.

CD-Name. A user-defined word that names a CMCS interface area described in a communication description entry within the Communication Section of the Data Division.

Character. The basic indivisible unit of the language.

Character Data Item. A data item consisting entirely of Standard Data Format characters.

Character Position. Amount of physical storage required to store a single standard data format character described with usage DISPLAY.

Character-String. A sequence of contiguous characters that form 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 is wholly numeric.

Clause. An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

CMCS. (See COBOL Message Control System.)

COBOL Character Set. The complete COBOL character set consisting of the 85 characters listed below:

Character	Meaning
0,1,...,9	Digit
A,B,...,Z	Uppercase letter
a,b,...,z	Lowercase letter
	Space (blank)
+	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 symbol
<	Less than symbol
!	Exclamation point
#	Number sign
%	Percent
&	Ampersand
'	Apostrophe
:	Colon
?	Question mark
@	Commercial at

NOTE: Certain of the characters comprising the COBOL character set may not be graphically represented in definitions of National and International Standard character sets. In these instances, a substitute graphic may be specified to replace the characters not represented.

COBOL Message Control System (CMCS). The Multics title for the Message Control System in COBOL. The different naming convention is used to distinguish it from other Multics software (Multics Communication System, MCS).

COBOL Word. (See Word)

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

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

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

Comment-Entry. An entry in the Identification Division that may be any combination of characters from the host computer's character set.

Comment Line. A source program line represented by an asterisk in the indicator area of the line and any characters from the host computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line, and by any characters in area A and area B of that line, causes page ejection prior to printing the comment.

Communication Description Entry. An entry in the Communication Section of the Data Division that is composed of the level indicator CD, followed by a cd-name, and then followed by a set of clauses as required. It describes the interface between the COBOL Message Control System (CMCS) and the COBOL program.

Communication Device. A mechanism (hardware or hardware/software) capable of sending data to a queue and/or receiving data from a queue. This mechanism may be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

Communication Section. The section of the Data Division that describes the interface areas between the CMCS and the program, composed of one or more CD description entries.

Compilation Activity. The activity (subdivision of a job) during which the source program is being compiled.

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 in which 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.

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

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 set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression. A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement. (See Simple Condition and Complex Condition.)

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

Conditional Variable. A data item, one or more values of which has a condition-name assigned to it.

Configuration Section. A section of the Environment Division that describes overall specifications of source and object computers.

Connective. A reserved word that is 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.)

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

Control Break. A change in the value of a data item that is referred to in the CONTROL clause. More generally, a change in the value of a data item that is used to control the hierarchical structure of a report.

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

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

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

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

Control. A set of body groups that is presented for a given value of a control data item or FINAL. Each control group may begin with a CONTROL HEADING, end with a CONTROL FOOTING, and contain DETAIL report groups.

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

Control Hierarchy. A designated sequence of report divisions defined by the positional order of FINAL and the data-names within a CONTROL phrase.

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

Currency Sign. The character $ of 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 that is available in the record area associated with the file.

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

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

Data Description Entry. A Data Division entry consisting of a level-number followed by a data-name, if required, and followed by a set of data clauses, as required.

Data Item. A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL program.

Data-Name. A user-defined word that names a data item described in a data description entry in the Data Division. When used in the formats, "data-name"

represents a word that can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

Debugging Line. Any source line with D or d in the indicator area of the line.

Debugging Section. A section that contains a USE FOR DEBUGGING statement.

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.

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

Delimited Scope Statement. Any statement that includes its explicit scope terminator.

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

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

Destination. The symbolic identification of the receiver of a transmission from a queue.

Digit Position. The amount of physical storage required to store a single digit. This amount may vary depending on the usage of the data item describing the digit position. If the Data Description entry specifies that usage is DISPLAY, then a digit position is synonymous with a character position.

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

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

```
CONTROL DIVISION.
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [ USING data-name-1 [ data-name-2 ] ... ] .
```

Dynamic Access. An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access) during the scope of the same OPEN statement.

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

Character	Meaning
b,B	Space
0	Zero

+	Plus
-	Minus
cr, CR	Credit
db, DB	Debit
z,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.

End of Procedure Division. The physical position in a COBOL source program after which no further procedures appear.

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

Environment Clause. A clause that appears as part of an Environment Division entry.

Execution Time. (See Object Time)

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

Expression. An arithmetic or conditional expression.

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

External Switch. A hardware or software device used to indicate that one of two alternate states exist.

Figurative Constant. A compiler-generated value referenced using certain reserved words.

File. A collection of records.

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

File Description (FD)
Saved Area Description (SA)
Sort file description (SD)

FILE-CONTROL. The name of an Environment Division paragraph in which the data files for a given source program are declared.

File Control Entry. A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

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

File-Name. A user-defined word that names a file described in a file description entry or a sort 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 sort file description entries together with their associated record descriptions.

Fixed Length Record. A record associated with a file whose FD or SD entry requires that all records contain the same number of character positions.

Footing Area. The portion of the page body adjacent to the bottom margin.

Format. A specific arrangement of a set of data.

Group Item. A named contiguous set of elementary or group 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 for that file.

Identifier. A data-name, followed as required by the syntactically correct combination of qualifiers, subscripts, and indexes necessary to make unique reference to a data item.

Imperative Statement. A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement can consist of a sequence of imperative statements.

Implementor Name. A system-name that refers to a particular feature available on that implementor's computing system.

Implicit Scope Terminator. A separator period which terminates the scope of preceding unterminated statements, or a phrase of a statement which indicates the end of the scope of any statement contained within the preceding phrase.

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

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

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

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

Indexed File. A file with indexed organization.

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

Input File. A file that is opened in the input mode.

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

Input-Output Control System. The set of run-time subroutines that perform physical and logical record processing in the execution of input/output commands.

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 transmitting and handling data during execution of the object program.

Input-Output-Technique. A system-name that specifies an input/output technique.

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

Integer. A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term "integer" appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero, unless explicitly allowed by the rules of that format.

Intermediate Data Item. A signed numeric data item that contains the results developed in the course of an arithmetic operation prior to when the final result is moved to the resultant-identifier, if any.

INVALID KEY Condition. A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

Key. A data item that identifies the location of a record, or a set of data items that serve to identify the ordering of data.

Key of Reference. The key used to access records within an indexed file.

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

Level Indicator. Two alphabetic characters that identify a specific type of file or a position in a hierarchy.

Level-Number. A user-defined word that indicates the position of a data item in the hierarchical structure of a logical record or that indicates special properties of a data description entry. A level-number is expressed as a one- or two-digit number. 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 can 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 that names a COBOL library that is accepted by the compiler for documentation purposes only.

Library Text. A sequence of character-strings and/or separators in an include file.

LINAGE-COUNTER. A special register whose value points to the current position within the page body.

Line. (See Report Line)

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 data items can be referred to by both the calling and called program.

Literal. A character-string whose value is implied by the ordered set of characters of which the string is composed.

Logical Operator. One of the reserved words AND, OR, or NOT. In the formation of a condition, both or either of AND and OR 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.

Low-Order End. The rightmost character of a string of characters.

MCS. (See Message Control System.)

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

Message. Data associated with an end-of-message indicator or an end-of-group indicator. (See Message Indicators.)

Message Control System (MCS). A communication control system that supports the processing of messages. See also COBOL Message Control System (CMCS).

Message Count. The count of the number of complete messages that exist in the designated queue of messages.

Message Indicators. EGI (end-of-group indicator), EMI (end-of-message indicator), and ESI (end-of-segment indicator) are conceptual indications that serve to notify the CMCS that a specific condition exists (end of group, end of message, end of segment).

Within the hierarchy of EGI, EMI, and ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI. An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by an ESI, EMI, or EGI. A message may be terminated by an EMI or EGI.

Message Segment. Data that forms a logical subdivision of a message normally associated with an end-of-segment indicator. (See Message Indicators.)

Mnemonic-Name. A user-defined word that is associated in the Environment Division with a specified hardware or operating system feature.

Native Character Set. The ASCII character set.

Native Collating Sequence. The ASCII collating sequence.

Negated Combined Condition. The NOT logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition. The NOT logical operator immediately followed by a simple condition.

Next Executable Sentence. The next sentence to which control is transferred after execution of the current statement is complete.

Next Executable Statement. The next statement to which control is transferred after execution of the current statement is complete.

Next Record. The record that logically follows the current record of a file.

Noncontiguous Items. Elementary data items in the Working-Storage and Linkage Sections that bear no hierarchic relationship to other data items.

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

Nonnumeric Literal. A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

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

Numeric Item. A data item whose description restricts its contents to a value represented by characters chosen from the digits 0 through 9; if signed, the item can also contain a +, -, or other representation of an operational sign.

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

OBJECT-COMPUTER. The name of an Environment Division paragraph that describes the computer environment within which the object program is executed.

Object of Entry. A set of operands and reserved words within a Data Division entry that immediately follows the subject of the entry.

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 can be used in place of "object program."

Object Time. The time at which an object program is executed.

Open Mode. The state of a file after execution of an OPEN statement for that file and before execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

Operand. Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this manual, any lowercase word that appears in a statement or entry format may be considered to be 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 File. An input file that is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the optional file.

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

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

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

Output Procedure. A set of statements to which control is given during execution of a SORT statement after the sort function is completed.

Padding Character. An alphanumeric character used to fill 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 media.

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

Page Footing. The report group that is presented at the end of a report page as determined by the Report Writer Control System.

Page Heading. A report group that is presented at the beginning of a report page as determined by the Report Writer Control System.

Paragraph. In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header. A reserved word, followed by a period and a space, 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. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record. (See Block)

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

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

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

Procedure-Name. A user-defined word that names a paragraph or section in the Procedure Division. It consists of a paragraph-name (which can be qualified) or a section-name.

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

Pseudo-Text. A sequence of character strings and/or separators bounded by, but not including, pseudo-text delimiters.

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

Punctuation Character. A character that belongs to the following set:

Character	Meaning
,	Comma
;	Semicolon
.	Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
	Space
=	Equal sign

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

Qualifier.

1. A data-name that is used in a reference together with another data-name at a lower level in the same hierarchy.

2. A section-name that is used in a reference together with a paragraph-name specified in that section.

3. A library-name that is used in a reference together with a text-name associated with that library.

Queue. A logical collection of messages awaiting transmission or processing.

Queue-Name. A symbolic name that indicates to the CMCS the logical path by which a message or a portion of a completed message may be accessible in a queue.

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

Record. (See Logical Record)

Record Area. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description. (See record description entry.)

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

Record Key. A 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.

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 specify the ordinal position of a logical record in a relative file.

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

Report Clause. A clause in the Report Section of the Data Division which appears in a report description entry, or a report group description entry.

Report Description Entry. An entry in the Report Section of the Data Division which is composed of the level indicator RD, followed by a report name, followed by a set of report clauses as required.

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

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

Report Group. In the Report Section of the Data Division, an 01 level-number entry and its subordinate entries.

Report Group Description Entry. An entry in the Report Section of the Data Division that is composed of the level-number 01, the optional data-name, a TYPE clause, and an optional set of report clauses.

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

Report Line. A division of a page representing one row of horizontal character positions. Each character position of a report line is aligned vertically beneath the corresponding character position of the report line above it. Report lines are numbered from 1, by 1, starting at the top of the page.

Report-Name. A user-defined word that names a report described in a report description entry within the Report Section of the Data Division.

Report Section. The section of the Data Division that contains one or more report description entries and their associated report group description entries.

Report Writer Control System (RWCS). An object time control system, provided by the implementor, that accomplishes the construction of reports.

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

Reserved Word. A COBOL word specified in the list of words that can be used in COBOL source programs, but that must not appear in the programs as user-defined words or system-names. (See Appendix A.)

Resultant-Identifier. A user-defined data item that is to contain the result of an arithmetic operation.

Run-Unit. A set of one or more object programs that function, at object time, as a unit to provide problem solutions.

RWCS. (See Report Writer Control System)

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 period and a space 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 a period and a space. The permissible section headers are:

In the Environment Division:

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

In the Data Division:

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

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

Section-Name. A user-defined word that names a section in the Procedure Division.

Segment-Number. A user-defined word that classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers can contain only

the characters 0, 1, ..., 9. A segment-number can be expressed as either a one- or two-digit number.

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

Separator. A punctuation character used to delimit character-strings.

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

Sequential File. A file with sequential organization.

Sequential Organization. The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

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

Simple Condition. Any single condition chosen from the set:

Relation condition
Class condition
Condition-name condition
Switch-status condition
Sign condition
Message condition
(Simple condition)

Sort File. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

Sort-Merge File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

Source. The symbolic identification of the originator of a transmission to a queue.

SOURCE-COMPUTER. The name of an Environment Division paragraph that describes the computer environment within which the source program is compiled.

Source Item. An identifier designated by a SOURCE clause that provides the value of a printable item.

Source Program. Although it is recognized that a source program can be represented by other forms and symbols, in this manual it always refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word "program" alone can be used in place of "source program." The Control Division may optionally be placed before the Identification Division to supply default values to the compiler, but it is not considered part of the 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 Multics names are related to user-specified mnemonic-names.

Special Registers. 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 the characteristics of 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 of infinite length and breadth, 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 written in the Procedure Division, beginning with a verb.

Sub-Queue. A logical hierarchical division of a queue.

Subject of Entry. An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

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.

Sum Counter. A signed numeric data item established by a SUM clause in the Report Section of the Data Division. The sum counter is used by the Report Writer Control System to contain the result of designated summing operations that take place during production of a report.

Switch-Status Condition. The proposition, for which a truth value can be determined, that a switch, capable of being set to an "ON" or "OFF" status, has been set to a specific status.

System-Name. A COBOL word that is used to communicate with the operating environment.

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

Table Element. A data item that belongs to the set of repeated items that compose a table.

Terminal. The origination of a transmission to a queue, or the receiver of a transmission from a queue.

Text-Name. A user-defined word that identifies library text.

Text-Word. Any character-string or separator, except a space, in a COBOL source program or 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, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1, respectively.

Unit. Random mass storage space allocated to the program in the execution activity under a single operating system file code.

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 can be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Variable-Length Data Item. A data item that, although physically fixed in size, contains a logically variable number of characters. Such a data item must contain the PICTURE symbol L in its data description entry.

Variable-Occurrence Data Item. A table element that is repeated a variable number of times. Such a data 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.

Volume. A unit of storage media, such as a reel of tape, or the mass storage file space allocated to a program by the operating system under one file code.

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

Working-Storage Section. The section of the Data Division that describes working-storage data items and constants, composed either of noncontiguous items or of working-storage records, or of both.

66-Level-Description-Entry. A data description entry that identifies RENAMES entries and can be used only as described by Format 2 of the Data Description Skeleton.

77-Level-Description-Entry. A data description entry that describes a noncontiguous data item with the level-number 77 and can be used only as described by Format 1 of the Data Description Skeleton.

88-Level-Description-Entry. A data description entry that defines condition-names associated with a conditional variable and can be used only as described by Format 3 of the Data Description Skeleton.

APPENDIX C

COLLATING SEQUENCE AND BIT CONFIGURATION CHARTS

Table C-1. ASCII Collating Sequence (Hexadecimal) and Bit Configuration

$b_4b_5b_6b_7$ (HEX)	$b_0b_1b_2b_3$ 0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000 0	NUL	DLE	SP	0	@	P	`	p								
0001 1	SOH	DC1	!	1	A	Q	a	q								
0010 2	STX	DC2	"	2	B	R	b	r								
0011 3	ETX	DC3	#	3	C	S	c	s								
0100 4	EOT	DC4	$	4	D	T	d	t								
0101 5	ENQ	NAK	%	5	E	U	e	u								
0110 6	ACK	SYN	&	6	F	V	f	v								
0111 7	BEL	ETB	'	7	G	W	g	w								
1000 8	BS	CAN	(	8	H	X	h	x								
1001 9	HT	EM	)	9	I	Y	i	y								
1010 A	NL	SUB	*	:	J	Z	j	z								
1011 B	VT	ESC	+	;	K	[	k	{								
1100 C	NP	FS	,	<	L	\	l	|							
1101 D	CR	GS	-	=	M	]	m	}								
1110 E	RRS	RS	.	>	N	^	n	~								
1111 F	BRS	US	/	?	O	_	o	DEL								EO

AS44-02

Table C-2. EBCDIC Collating Sequence

$b_4b_5b_6b_7$ / $b_0b_1b_2b_3$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000 0	NUL	DLE	DS		SP	&	-						{	}	\	0
0001 1	SOH	DC1	SOS				/		a	j	~		A	J		1
0010 2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0011 3	ETX	TM							c	l	t		C	L	T	3
0100 4	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101 5	HT	NL	LF	RS					e	n	v		E	N	V	5
0110 6	LC	BS	ETB	UC					f	o	w		F	O	W	6
0111 7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
1000 8		CAN							h	q	y		H	Q	Y	8
1001 9	RLF	EM						`	i	r	z		I	R	Z	9
1010 A	SMM	CC	SM		¢	!	¦	:								
1011 B	VT	CU1	CU2	CU3	.	$	,	#								
1100 C	FF	IFS		DC4	<	*	%	@								
1101 D	CR	IGS	ENQ	NAK	(	)	_	'								
1110 E	SO	IRS	ACK		+	;	>	=								
1111 F	SI	IUS	BEL	SUB	|	¬	?	"								EO

NOTE: The following substitutions are made on some equipment.

a. ASCII environment
- ¢ (4A) prints as [(left bracket)
- ! (5A) prints as] (right bracket)
- | (4F) prints as ! (exclamation mark)
- ¬ (5F) prints as ^ (circumflex accent)

and in ISO environment only
- ¦ (6A) prints as | (vertical line)

b. 200-2000 Series environment
- | (4F) prints as C_R (credit sign)
- \ (E0) prints as ≠ (unequal sign)
- ¬ (5F) prints as ■ (filled square)
- _ (6D) prints as □ (square)

c. 100 - 400 - 600 - 6000 Series environment
- | (4F) prints as] (right bracket)
- ¢ (4A) prints as [(left bracket)
- ¬ (5F) prints as ↑ (upper arrow)
- _ (6D) prints as ← (left arrow)

APPENDIX D

FIPS LEVELING

Federal Information Processing Standard publication 21-1 announced the adoption of American National Standard COBOL, X3.23-1974, as a Federal Standard. The modules defined in X3.23-1974 are combined into four levels identified as: Low, Low-Intermediate, High-Intermediate, and High. Each Federal Standard COBOL level is composed of either the high or low levels of the nucleus and ten of the eleven Functional Processing Modules (FPM's) defined in X3.23-1974. The four Federal Standard COBOL levels are reflected in the following table. The numbers in the table refer to the level within the FPM or nucleus as designated in X3.23-1974, and a dash in the table denotes the corresponding FPM is omitted.

	Low Level	Low Intermediate Level	High Intermediate Level	High Level
NUCLEUS	1	1	2	2
FPM's				
TABLE HANDLING	1	1	2	2
SEQUENTIAL I-O	1	1	2	2
RELATIVE I-O	-	1	2	2
INDEXED I-O	-	-	-	2
SORT-MERGE	-	-	1	2
REPORT WRITER	-	-	-	-
SEGMENTATION	-	1	1	2
LIBRARY	-	1	1	2
DEBUG	-	1	2	2
INTER-PROGRAM COMMUNICATION	-	1	2	2
COMMUNICATION	-	-	2	2

NOTE: The "REPORT WRITER" module is not mandatory in any Federal Level. However, the specifications contained in X3.23-1974 should be used to the extent practical, consistent with the requirements.

The COBOL user may restrict use to a subset of the Multics COBOL language by using the -level control argument. A reference to this argument must take the form:

-level N or -level NM

where N and M are integers and

$1 \leq N \leq 5$ and $1 \leq M \leq 3$

The integer N specifies one of the Federal Standard COBOL levels according to the following table:

Integer	Level
1	low
2	low-intermediate
3	high-intermediate
4	high
5	full Multics COBOL

Syntactical and semantics constructs which do not belong to the specified Federal Standard COBOL level are diagnosed. If N = 4 is specified, then constructions which are extensions to ANSI COBOL-74 are diagnosed. If N = 5, then no leveling diagnostics appear. The default for N is 5.

The integer M specifies the severity associated with the labeling diagnostics. The default for M is 3.

Further details for the -level control argument may be found in the Multics COBOL User's Guide, (Order No. AS43).

Following is a list of all elements in the American National Standard COBOL organized by the level in which each element is located. A level code is composed of, from left to right, a single digit and a three-character mnemonic code. The digits 1 and 2 denote the low and high levels, respectively, within a module. The three-character name defines the module as shown in the list:

Mnemonic Name	Meaning
NUC	Nucleus
TBL	Table Handling
SEQ	Sequential I-O
REL	Relative I-O
INX	Indexed I-O
SRT	Sort-Merge
RPW	Report Writer
SEG	Segmentation
LIB	Library
DEB	Debug
IPC	Inter-Program Communication
COM	Communication

ELEMENTS	LEVEL
Language Concepts	
Character set	
Characters used for words	
0,1,...,9,A,B,...,Z - (hyphen or minus).	1 NUC
Characters used for punctuation	
. " ( ) - space.	1 NUC
, ;	2 NUC
Characters used in arithmetic operations	
+ - * / **	2 NUC
Characters used in relations	
> < = ...	2 NUC
Characters used in editing	
B 0 + - CR DB Z * $, . /	1 NUC
Separators	1 NUC
Semicolon and comma not permitted	1 NUC
Semicolon and comma are allowed	2 NUC
Character-strings.	1 NUC
COBOL words	1 NUC
Not more than 30 characters	
User-defined words	1 NUC
cd-name	1 COM
condition-name.	2 NUC
data-name	
Must begin with an alphabetic character. . .	1 NUC
Need not begin with an alphabetic character	2 NUC
file-name	1 SEQ
index-name.	1 TBL
level-number.	1 NUC
library-name.	2 LIB
mnemonic-name	1 NUC
paragraph-name.	1 NUC
program-name.	1 NUC
record-name	1 SEQ
report-name	1 RPW
routine-name.	1 NUC
section-name.	1 NUC
segment-number.	1 SEG
text-name	1 LIB
System-names	1 NUC
computer-name	
implementor-name	
language-name	
Reserved words	1 NUC
Key words	1 NUC
Optional words.	1 NUC
Connectives	
Qualifier connectives: OF, IN	2 NUC
Series connectives: , (separator comma) and ; (separator semicolon)	2 NUC
Logical connectives: AND, OR, AND NOT OR NOT.	2 NUC

Reserved Words (continued)	
Special registers	
LINE-COUNTER, PAGE-COUNTER	1 RPW
LINAGE-COUNTER	2 SEQ
DEBUG-ITEM	1 DEB
Figurative constants	
ZERO	1 NUC
ZEROS, ZEROES	2 NUC
SPACE	1 NUC
SPACES	2 NUC
HIGH-VALUE, LOW-VALUE	1 NUC
HIGH-VALUES, LOW-VALUES	2 NUC
QUOTE	1 NUC
QUOTES	2 NUC
All literal	2 NUC
Special-character words	
Arithmetic operators	2 NUC
Relation characters	2 NUC
Literals	1 NUC
Nonnumeric literals have lengths from 1 through 120 characters	
Numeric literals have lengths from 1 through 18 digits	
PICTURE character-strings	1 NUC
Comment-entries	1 NUC
Qualification	2 NUC
No qualification permitted	1 NUC
Qualification permitted	2 NUC
Subscripting	
3 levels	1 TBL
Indexing	
3 levels	1 TBL
Identification Division	
The PROGRAM-ID paragraph	1 NUC
The AUTHOR paragraph	1 NUC
The INSTALLATION paragraph	1 NUC
The DATE-WRITTEN paragraph	1 NUC
The DATE-COMPILED paragraph	2 NUC
The SECURITY paragraph	1 NUC
Environment Division	
Configuration Section	
The SOURCE-COMPUTER paragraph	1 NUC
computer-name	1 NUC
WITH DEBUGGING MODE phrase	1 DEB
The OBJECT-COMPUTER paragraph	1 NUC
computer-name	1 NUC
MEMORY SIZE clause	1 NUC

ELEMENTS	LEVEL
The OBJECT-COMPUTER paragraph (continued)	
PROGRAM COLLATING SEQUENCE clause.	1 NUC
SEGMENT-LIMIT clause	1 SEG
The SPECIAL-NAMES paragraph	
implementor-name IS mnemonic-name.	1 NUC
ON STATUS.	1 NUC
OFF STATUS	1 NUC
implementor-name series.	1 NUC
alphabet-name clause	
STANDARD-1.	1 NUC
NATIVE.	1 NUC
implementor-name.	1 NUC
literal	2 NUC
CURRENT SIGN clause.	1 NUC
DECIMAL-POINT clause	1 NUC
Input-Output Section	
The FILE-CONTROL paragraph	
SELECT clause.	1 SEQ
	1 REL
	1 INX
	1 SRT
Optional phrase	2 SEQ
ASSIGN TO implementor-name clause.	1 SEQ
	1 REL
	1 INX
	1 SRT
RESERVE AREA(S) clause	2 SEQ
	2 REL
	2 INX
ORGANIZATION clause	
SEQUENTIAL.	1 SEQ
RELATIVE.	1 REL
INDEXED	1 INX
ACCESS MODE clause	
SEQUENTIAL.	1 SEQ
	1 REL
	1 INX
RANDOM.	1 REL
	1 INX
DYNAMIC	2 REL
	2 INX
RECORD KEY clause.	1 INX
ALTERNATE RECORD KEY clause.	2 INX
FILE STATUS clause	1 SEQ
	1 REL
	1 INX
The I-O-CONTROL paragraph	
RERUN clause	1 SEQ
	1 REL
	1 INX

ELEMENTS	LEVEL
The I-O-CONTROL paragraph (continued)	
SAME AREA clause	1 SEQ 1 REL 1 INX
SAME RECORD AREA clause	2 SEQ 2 REL 2 INX 2 SRT
SAME SORT/SORT-MERGE AREA clause	2 SRT
SAME series	1 SEQ 1 REL 1 INX
MULTIPLE FILE TAPE clause	2 SEQ
Data Division	
Communication Section	1 COM
File Section	1 SEQ 1 REL 1 INX 1 SRT 1 RPW
Linkage Section	1 IPC
Report Section	1 RPW
Working-Storage Section	1 NUC
The communication description entry	1 COM
The data description entry	1 NUC
The file description entry	1 SEQ 1 REL 1 INX 1 RPW
The record description entry	1 SEQ 1 REL 1 INX
The report description entry	1 RPW
The report group description entry	1 RPW
The sort-merge description entry	1 SRT
The BLANK WHEN ZERO clause	1 NUC
The BLOCK CONTAINS clause	
integer CHARACTERS/RECORDS	1 SEQ 1 REL 1 INX 1 RPW
integer-1 TO integer-2 CHARACTERS/RECORDS	2 SEQ 2 REL 2 INX 1 RPW
The CODE clause	1 RPW
The CODE-SET clause	1 SEQ 1 RPW
The COLUMN NUMBER clause	1 RPW
The CONTROL clause	1 RPW
The data-name clause	1 NUC 1 RPW

ELEMENTS	LEVEL
The DATA RECORDS clause.	1 SEQ
	1 REL
	1 INX
	1 SRT
FILLER	1 NUC
The GROUP INDICATE clause.	1 RPW
The JUSTIFIED clause (may be abbreviated JUST)	1 NUC
The LABEL RECORDS clause	
STANDARD/OMITTED.	1 SEQ
	1 REL
	1 INX
	1 RPW
Level-number	
01 through 10 (level-number must be 2 digits)	1 NUC
1 through 49 (level-number may be 1 digit).	2 NUC
66 or 88.	2 NUC
77.	1 NUC
The LINAGE clause.	2 SEQ
The LINE NUMBER clause	1 RPW
The NEXT GROUP clause.	1 RPW
The OCCURS clause	
integer TIMES	1 TBL
ASCENDING/DESCENDING data-name.	2 TBL
data-name series	2 TBL
INDEXED BY index-name	1 TBL
integer-1 TO integer-2 DEPENDING ON data-name	2 TBL
The PAGE clause.	1 RPW
The PICTURE clause (may be abbreviated PIC)	
Character-string may contain 30 characters.	1 NUC
Data characters: A X 9	1 NUC
Operational symbols: S V P	1 NUC
Fixed insertion characters: 0 B , . $ + - DB CR /.	1 NUC
Replacement or floating characters: + - Z *.	1 NUC
Currency sign substitution.	1 NUC
Decimal point substitution.	1 NUC
The RECORD CONTAINS clause	1 SEQ
	1 REL
	1 INX
	1 SRT
	1 RPW
The REDEFINES clause	
May not be nested	1 NUC
May be nested	2 NUC
The RENAMES clause	2 NUC
The REPORT clause.	1 RPW
The SIGN clause.	1 NUC
The SOURCE clause.	1 RPW
The SUM clause	1 RPW
The SYNCHRONIZED clause (may be abbreviated SYNC).	1 NUC
The TYPE clause.	1 RPW

ELEMENTS	LEVEL
The USAGE clause	
COMPUTATIONAL (may be abbreviated COMP)	1 NUC
DISPLAY	1 NUC
INDEX	1 TBL
The VALUE clause	
literal	1 NUC
literal series	2 NUC
literal THRU literal	2 NUC
literal range series	2 NUC
The VALUE OF clause	
implementor-name IS literal	1 SEQ 1 REL 1 INX 1 RPW
implementor-name IS data-name	2 SEQ 2 REL 2 INX 1 RPW
Procedure Division	
USING phrase in Procedure Division header	1 IPC
Declaratives	1 SEQ 1 REL 1 INX 1 RPW 1 DEB
Arithmetic expressions	2 NUC
Conditional expressions	1 NUC
Simple conditions	1 NUC
Relation condition	1 NUC
Relation operators	
[NOT] GREATER THAN	1 NUC
[NOT] >	2 NUC
[NOT] LESS THAN	1 NUC
[NOT] <	2 NUC
[NOT] EQUAL TO	1 NUC
[NOT] =	2 NUC
Comparison	
Numeric operands	1 NUC
Nonnumeric operands	
Operands must be of equal size	1 NUC
Operands may be unequal in size	2 NUC
Class condition	1 NUC
NOT option	
Switch-status condition	1 NUC
Condition-name condition	2 NUC
Sign condition	2 NUC
NOT option	
Complex condition	2 NUC
Logical operators AND, OR, and NOT	
Negated simple conditions	2 NUC
Combined and negated combined conditions	2 NUC
Abbreviated combined relation condition	2 NUC

ELEMENTS	LEVEL
The arithmetic statements	
Arithmetic operands limited to 18 digits.	1 NUC
Overlapping operands	1 NUC
	1 TBL
Multiple results in arithmetic statements.	2 NUC
The ACCEPT statement	
Only one transfer of data.	1 NUC
No restriction on the number of transfers of data	2 NUC
FROM phrase	2 NUC
MESSAGE COUNT phrase.	1 COM
The ADD statement	
identifier/literal series	1 NUC
TO identifier	1 NUC
TO identifier series.	2 NUC
GIVING identifier	1 NUC
GIVING identifier series.	2 NUC
ROUNDED phrase.	1 NUC
SIZE ERROR phrase	1 NUC
CORRESPONDING phrase.	2 NUC
The ALTER statement	
procedure-name.	1 NUC
procedure-name series	2 NUC
The CALL statement	
literal	1 IPC
identifier.	2 IPC
USING data-name	1 IPC
ON OVERFLOW phrase.	2 IPC
The CANCEL statement	2 IPC
The CLOSE statement	
Single file-name.	1 SEQ
file-name series.	2 SEQ
	1 REL
	1 INX
REEL	1 SEQ
UNIT	1 SEQ
NO REWIND.	2 SEQ
FOR REMOVAL.	2 SEQ
LOCK	2 SEQ
	1 REL
	1 INX
The COMPUTE statement.	2 NUC
The DELETE statement	1 REL
	1 INX
The DISABLE statement	
INPUT	1 COM
TERMINAL	2 COM
OUTPUT.	1 COM
KEY identifier/literal.	1 COM
The DISPLAY statement	
Only one transfer for data.	1 NUC
No restriction on the number of transfers of data	2 NUC
UPON phrase	2 NUC

ELEMENTS	LEVEL
The DIVIDE statement	
INTO identifier	1 NUC
INTO identifier series	2 NUC
BY identifier	1 NUC
GIVING identifier	1 NUC
GIVING identifier series	2 NUC
REMAINDER phrase	2 NUC
ROUNDED phrase	1 NUC
SIZE ERROR phrase	1 NUC
The ENABLE statement	
INPUT	1 COM
TERMINAL	2 COM
OUTPUT	1 COM
KEY identifier/literal	1 COM
The ENTER statement	1 NUC
The EXIT statement	1 NUC
The EXIT PROGRAM statement	1 IPC
The GENERATE statement	1 RPW
The GO TO statement	
procedure-name is required	1 NUC
procedure-name is optional	2 NUC
DEPENDING ON phrase	1 NUC
The IF statement	
Statements must be imperative statements	1 NUC
Nested statements	2 NUC
ELSE	1 NUC
The INITIATE statement	1 RPW
The INSPECT statement	
Only single character data item	1 NUC
Multi-character data item	2 NUC
The MERGE statement	2 SRT
The MOVE statement	
TO identifier	1 NUC
TO identifier series	1 NUC
CORRESPONDING phrase	2 NUC
The MULTIPLY statement	
BY identifier	1 NUC
BY identifier series	2 NUC
GIVING identifier	1 NUC
GIVING identifier series	2 NUC
ROUNDED phrase	1 NUC
SIZE ERROR phrase	1 NUC
The OPEN statement	
INPUT	
Single file-name	1 SEQ
file-name series	2 SEQ
	1 REL
	1 INX
REVERSED	2 SEQ
NO REWIND	2 SEQ

ELEMENTS	LEVEL
The OPEN statement (continued)	
I-O	
Single file-name	1 SEQ
file-name series	2 SEQ
	1 REL
	1 INX
EXTEND	
file-name series	2 SEQ
INPUT, OUTPUT, I-O, and EXTEND series	2 SEQ
INPUT, OUTPUT, and I-O series	1 REL
	1 INX
The PERFORM statement	
procedure-name.	1 NUC
THRU phrase	1 NUC
TIMES phrase.	1 NUC
UNTIL phrase.	2 NUC
VARYING phrase.	2 NUC
The READ statement	
file-name	1 SEQ
	1 REL
	1 INX
INTO identifier	1 SEQ
	1 REL
	1 INX
AT END phrase	1 SEQ
	1 REL
	1 INX
INVALID KEY phrase.	1 REL
	1 INX
NEXT RECORD	2 REL
	2 INX
KEY IS phrase	2 INX
The RECEIVE statement	
MESSAGE	1 COM
SEGMENT	2 COM
INTO identifier	1 COM
NO DATA phrase.	1 COM
The RELEASE statement	
record-name	1 SRT
FROM phrase	1 SRT
The RETURN statement	
file-name	1 SRT
INTO phrase	1 SRT
AT END phrase	1 SRT
The REWRITE statement	
FROM identifier	1 SEQ
	1 REL
	1 INX
INVALID KEY Phrase.	1 REL
	1 INX
The SEARCH statement	2 TBL

ELEMENTS	LEVEL
The SEND statement	
FROM identifier-1	2 COM
FROM identifier-1 WITH	1 COM
WITH identifier-2	2 COM
WITH EGI	1 COM
WITH EMI	1 COM
WITH ESI	2 COM
BEFORE/AFTER ADVANCING	1 COM
The SET statement	1 TBL
The SORT statement	
Only one SORT statement, a STOP RUN statement, and any associated input-output procedures allowed in the nondeclarative portion of a program	1 SRT
Program not limited to one SORT statement	2 SRT
COLLATING SEQUENCE phrase	2 SRT
The START statement	2 REL 2 INX
The STOP statement	1 NUC
The STRING statement	2 NUC
The SUBTRACT statement	
identifier/literal series	1 NUC
FROM identifier	1 NUC
FROM identifier series	2 NUC
GIVING identifier	1 NUC
GIVING identifier series	2 NUC
ROUNDED phrase	1 NUC
SIZE ERROR phrase	1 NUC
CORRESPONDING phrase	2 NUC
The SUPPRESS statement	1 RPW
The TERMINATE statement	1 RPW
The UNSTRING statement	2 NUC
The USE statement	
EXCEPTION/ERROR PROCEDURE	
ON file-name/INPUT/OUTPUT/I-O	1 SEQ 1 REL 1 INX
ON file-name series	2 SEQ 2 REL 2 INX
ON EXTEND	2 SEQ
BEFORE REPORTING	1 RPW
The USE FOR DEBUGGING statement	
procedure-name	1 DEB
procedure-name series	1 DEB
ALL PROCEDURES	1 DEB
ALL REFERENCES OF identifier series	2 DEB
file-name series	2 DEB
cd-name series	2 DEB

ELEMENTS	LEVEL
The WRITE statement	
record-name	1 SEQ
	1 REL
	1 INX
FROM identifier	1 SEQ
	1 REL
	1 INX
BEFORE/AFTER ADVANCING	
integer LINES	1 SEQ
PAGE	1 SEQ
identifier LINES	2 SEQ
mnemonic-name	2 SEQ
AT END-OF-PAGE phrase	2 SEQ
INVALID KEY phrase	1 REL
	1 INX
Segmentation	
Segment-number	1 SEG
Fixed segment-number range 0 through 49	1 SEG
Non-fixed segment-number range 50 through 99	1 SEG
SEGMENT-LIMIT clause	2 SEG
Library	
COPY statement	1 LIB
OF/IN library-name	2 LIB
REPLACING phrase	2 LIB
Reference format	1 NUC
Sequence numbers	1 NUC
Area A	1 NUC
Division header	1 NUC
Section header	1 NUC
Paragraph header	1 NUC
Data Division entries	1 NUC
Area B	1 NUC
Paragraphs	1 NUC
Data Division entries	1 NUC
Continuation of lines	1 NUC
Nonnumeric literals	1 NUC
Words and numeric literals	2 NUC
Comment lines	1 NUC
Asterisk (*) comment lines	1 NUC
Stroke (/) comment lines	1 NUC

INDEX

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE MULTICS COBOL
REFERENCE MANUAL

ORDER NO. AS44-02

DATED DECEMBER 1983

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE MULTICS COBOL
REFERENCE MANUAL

ORDER NO. AS44-02

DATED DECEMBER 1983

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

CUT ALONG LINE

FOLD ALONG LINE

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho, Chiyoda-ku, Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

40407, 2.5C583, Printed in U.S.A.