FIPS PUB 47

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

1977 February 1

U.S. DEPARTMENT OF COMMERCE / National Bureau of Standards

FEDERAL STANDARD COBOL POCKET GUIDE

Category: Software
Subcategory: Programming Language

FOREWORD

The Federal Information Processing Standards Publication Series of the National Bureau of Standards is the official publication relating to standards adopted and promulgated under the provisions of Public Law 89-306 (Brooks Bill) and under Part 6 of Title 15, Code of Federal Regulations. These legislative and executive mandates have given the Secretary of Commerce important responsibilities for improving the utilization and management of computers and automatic data processing systems in the Federal Government. To carry out the Secretary's responsibilities, the National Bureau of Standards, through its Institute for Computer Sciences and Technology, provides leadership, technical guidance, and coordination of government efforts in the development of guidelines and standards in these areas.

The establishment of COBOL as a Federal Standard (FIPS PUB 21) is an effort to assist the Federal Government ADP user in stating data processing applications in such a way that the programs and data can be developed and ma[illegible]tained with a minimum of time and effort. [illegible] further assist in the programming task, FIPS Task Group 9 made a recommendation to the National Bureau of Standards that a companion FIPS PUB be published that could be used as a condensed programmer's reference guide of the standard language. Accordingly, the National Bureau of Standards is pleased to have the opportunity to make this reference material available for use by Federal agencies.

R. M. DAVIS, Director
Institute for
Computer Sciences
and Technology

ABSTRACT

This document contains a composite language skeleton of Federal Standard COBOL. It is intended to display complete and syntactically correct formats for the High Level of the standard. In addition, the document contains other selected prompts for the COBOL programmer to assist in expediting the programming task.

Key Words: COBOL; COBOL programming aids; Federal Standard COBOL; programming aids; programming languages.

Federal Information
Processing Standards Publication 47

1977 February 1

Announcing The

Federal Standard COBOL Pocket Guide

Federal Information Processing Standards Publications are issued by the National Bureau of Standards pursuant to the Federal Property and Administrative Services Act of 1949, as amended, Public Law 89-306 (79 Stat. 1127), Executive Order 11717 (38 FR 12315, dated May 11, 1973), and Part 6 of Title 15 Code of Federal Regulations (CFR).

NAME OF PUBLICATION. Federal Standard COBOL Pocket Guide.

CATEGORY. Software, Programming Language.

EXPLANATION. The purpose of this publication is to provide a handy prompt for COBOL programmers. The document contains a complete language skeleton for the high level of Federal Standard COBOL. Although not a part of Federal Standard COBOL, the Report Writer facility has been included for those having access to the American National Standard COBOL Report Writer facility.

APPROVING AUTHORITY. Department of Commerce, National Bureau of Standards (Institute for Computer Sciences and Technology).

MAINTENANCE AGENCY. Department of Commerce, National Bureau of Standards (Institute for Computer Sciences and Technology).

CROSS INDEX. FIPS PUB 21-1, COBOL.

WHERE TO OBTAIN COPIES OF THIS PUBLICATION. Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, Virginia 22161. When ordering, refer to Federal Information Processing Standards Publication 47(NBS-FIPS-PUB-47), title, and Accession Number. When microfiche is desired, this should be specified. Payment may be made by check, money order, or deposit account.

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 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.

This pocket guide was initially prepared by the Defense Communications Agency, Command and Control Technical Center. The Institute for Computer Sciences and Technology wishes to thank the Defense Communications Agency and, in particular, James J. Pottmyer, for the preparation of the material presented herein.

COBOL

POCKET GUIDE

NOTATIONS FOR SYNTAX DIAGRAM USAGE

- *WORDS*—Underlined uppercase words are required when the function of which they are a part is used, e.g., **<u>PICTURE</u>**
  Uppercase words which are not underlined are optional when the function of which they are a part is used, e.g., **<u>CURRENCY</u> SIGN <u>IS</u>**
  Lowercase words in a syntax diagram are generic terms used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or complete syntactical entries that must be supplied. Where a generic term appears more than once in a format, a number or letter appendage to the term serves to identify it for explanation or discussion in American National Standard X3.23-1974. See Definition topic for a list of these lowercase words.

- *BRACKETS* [ ]—Used to show words or phrases which are optional entries, e.g., **[<u>VALUE</u> IS literal]**

- *BRACES* { } —Used to show a mutually exclusive choice of contents, e.g., $\left\{\begin{matrix}\underline{\textbf{PICTURE}}\\ \underline{\textbf{PIC}}\end{matrix}\right\}$

- *ELLIPSES* •••—Used to represent the position in a syntax diagram at which repetition may occur at the user's option. They occur immediately following a right bracket or right brace and indicate that everything between that bracket or brace and its *paired* left bracket or left brace may be repeated.

- *COMMA* , *and SEMICOLON* ; —These symbols may appear where shown in a format and are interchangeable. Their inclusion is optional as desired by the programmer for readability

- *PERIOD* . —When one is shown in a syntax diagram, it is required.

- *SPECIAL CHARACTERS* + – / * ** > = < —Where one appears in a syntax diagram (although not underlined), it is required.

DEFINITIONS

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.

arithmetic-expression—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. See topic on Arithmetic Expressions.

cd-name—A user-defined **word** that names a Message Control System interface area described in a **communication-description-entry** within the **COMMUNICATION SECTION** of the **DATA DIVISION**.

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

comment-entry—An entry in the **IDENTIFICATION DIVISION** that may be any combination of characters from the computer character set.

communication-description-entry—An entry in the **COMMUNICATION SECTION** of the **DATA DIVISION** that describes the interface between the Message Control System (MCS) and the COBOL program. See syntax diagrams in Data Division topic.

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. 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. See syntax diagrams in Condition Format topic.

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 an implementor-defined switch or device.

data-description-entry—An entry in the **DATA DIVISION** that is composed of a **level-number** followed by a **data-name**, if required, and then followed by a set of data clauses, as required. See syntax diagrams in Data Division topic.

data-name—A user-defined **word** that names a data item described in a **data-description-entry** in the **DATA DIVISION**. When used in the general formats, **data-name** represents a **word** which can neither be suscripted, indexed, nor qualified unless specifically permitted by the rules for that format, i.e., there are some restrictions on using the syntax diagram for an **identifier**.

declarative-sentence—A compiler-directing sentence consisting of a single **USE** statement terminated by the separator period. See syntax diagrams in Procedure Division topic.

file-control-entry—An entry in the **FILE-CONTROL** paragraph of the **ENVIRONMENT DIVISION** by which a data file is declared. See syntax diagrams in Environment Division topic.

file-name—A user-defined **word** that names a file described in a file description entry or sort-merge file description entry within the **FILE SECTION** of the **DATA DIVISION**.

identifier—A **data-name**, followed as required by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item. See syntax diagrams in Identifier Format topic.

imperative-statement—A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements. An imperative verb is any except for **IF**, **ENTER**, **USE**, **COPY**, or which contain the optional phrases **SIZE ERROR**, **INVALID KEY**, **ON OVERFLOW**, **NO DATA**, **AT END**, or **END-OF-PAGE**.

implementor-name—A system-name that refers to a particular feature available on that implementor's computing system.

index-name—A user-defined **word** that names a computer storage position or register associated with a specific table, the contents of which identify a particular element in the table.

integer—A numeric **literal** without a decimal point which must neither be signed nor zero unless explicitly allowed by the rules of that format.

language-name—A system-name that specifies a particular programming language.

level-number—A user-defined **word** which indicates the position of a data item in the hierarchical structure of a logical record or which 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** may be written either as a single digit or as a zero followed by a significant digit. **level-numbers 66**, **77**, and **88** identify special properties of a **data-description-entry**.

library-name—A user-defined **word** that names a COBOL library that is to be used by the compiler for a given source program compilation.

literal—A **character-string** whose value is implied by the ordered set of characters comprising the string or by specification of a reserved word which references a figurative constant. Every **literal** is one of two types, non-numeric or numeric. Rules for particular format sometime constrain the type or length of a **literal**.

mnemonic-name—A user-defined **word** that is associated in the **ENVIRONMENT DIVISION** with a specified **implementor-name**.

paragraph-name—A user-defined **word** that identifies and begins a paragraph in the **PROCEDURE DIVISION**. A **paragraph-name** need not contain any alphabetic characters.

procedure-name—A user-defined **word** which is used to name a paragraph or section in the **PROCEDURE DIVISION**. It consists of a **paragraph-name** (which may be qualified), or a **section-name**.

program-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 (two contiguous characters ==).

record-description-entry—The total set of **data-description-entries** associated with a particular record. The first **data-description-entry** in the set must have a **level-number** of **1**.

record-name—A user-defined **word** that names a record described in a **record-description-entry** in the **DATA DIVISION**.

relation-condition—The proposition, for which a truth 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 syntax diagrams in Condition Format topic.

relational-operator—The permissible operators are:

```
IS [NOT] GREATER THAN
IS [NOT] >
IS [NOT] LESS THAN
IS [NOT] <
IS [NOT] EQUAL TO
IS [NOT] =
```

report-group-description-entry—In the **REPORT SECTION** of **DATA DIVISION**, an **01 level-number** entry and its subordinate entries. See syntax diagrams.

report-name—A user-defined **word** that names a report described in a **report-description-entry** within the **REPORT SECTION** of the **DATA DIVISION**.

routine-name—A user-defined **word** that identifies a procedure written in a language other than COBOL.

section-name—A user-defined **word** which names a section in the **PROCEDURE DIVISION**. A **section-name** need not contain any alphabetic characters.

segment-number—A user-defined **word** which classifies sections in the **PROCEDURE DIVISION** for purposes of segmentation. **segment numbers** may be expressed either as a one- or two-digit number.

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

simple-condition—Any single **condition** chosen from the set:

Relation-Condition
Class Condition
Condition-Name Condition
Switch-Status Condition
Sign Condition

or a **simple-condition** enclosed in parentheses. See syntax diagrams in Condition Format topic.

statement—A syntactically valid combination of **words** and symbols written in the **PROCEDURE DIVISION** beginning with a verb.

subscript—An **integer** or a numeric data item (with no digits to the right of the assumed decimal point) whose value identifies a particular element in a table.

text-name—A user-defined **word** which identifies a particular sequence of **character-strings** within a COBOL library.

word—A **character-string** of 1 to 30 characters which forms a user-defined **word**, system-name, or a reserved word.

77-level-description-entry—A **data-description-entry** that describes a non-contiguous data item with the **level-number 77**. See syntax diagrams for **data-description-entry** in Data Division topic.

FIGURATIVE CONSTANTS

A figurative constant is a value referenced by the following reserved words. A figurative constant may be used wherever **literal** appears in a syntax diagram, subject to contraints in particular formats or the type (numeric or non-numeric).

ZERO, ZEROS, ZEROES—Represents numeric value "0", or one or more of the character "0", depending on the context in which used. When a **literal** must be of numeric type, these are the only figurative constants which can be used.

SPACE, SPACES—Represents one or more of the character space from the computer's character set.

HIGH-VALUE, HIGH-VALUES—Represents one or more of the character which has the highest ordinal position in the program collating sequence. For the **STANDARD-1** collating sequence, this is the DEL character, ASCII 7/15.

LOW-VALUE, LOW-VALUES—Represents one or more of the character which has the lowest ordinal position in the program collating sequence. For the **STANDARD-1** collating sequence, this is the NUL character, ASCII 0/0.

QUOTE, QUOTES—Represents one or more of the character ("). This figurative constant cannot be used in place of a quotation mark in a source program to bound a non-numeric **literal**. i.e., **QUOTE ABC QUOTE** cannot be used for **"ABC"**.

ALL literal—Represents one or more of the string of characters comprising the **literal**. The **literal** must be either a nonnumeric **literal** or any other figurative constant. Cannot be used with the **DISPLAY**, **INSPECT** or **STOP** statements.

Notes:
1. The singular and plural forms of the figurative constants are equal and may be used interchangeably.
2. Figurative constants may not be bounded by quotation marks.
3. When a figurative constant is not associated with another data item, it is assumed to be one character long, otherwise it assumes the length of the data item with which it is associated.

225-116 O - 77 - 2

PICTURE CHARACTER STRING

A **PICTURE character-string** contains 1 to 30 characters describing the characteristics and editing requirements of an elementary data item. An unsigned 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, e.g., **X(5)** is equivalent to **XXXXX**. (Note that **S V . CR** and **DB** may appear only once.) The rules for forming **PICTURE character-strings** for the different categories of data are:

SYMBOL	REPRESENTS	MAY APPEAR WITH	RESTRICTIONS	NOTES
		ALPHABETIC		
A	Alphabetic character	**A B**	At least one **A** must be present	
B	Space character insertion	**A B**		
		NUMERIC		
9	Numeric character	**9 P V S**	At least one **9** must be present	1
P	Assumed decimal scaling position	**9 P V S**	Either first or last except for **S** and **V**	1, 2, 3
V	Location of assumed decimal point within item	**9 P S**	Only one **V** allowed	2, 3
S	Presence of operational sign	**9 P V**	Must be leftmost; only one **S** allowed	4
		ALPHANUMERIC		
X A 9	Any allowable character in the computer character set	**X A 9**	Either at least one **X** must be present or else both **A** and **9** must be present	

Notes:

1. The total number of digit positions in a numeric or numeric edited item must be between 1 and 18. The symbols **9 P Z *** and the second and following occurrences of **+ - $** count as digit positions.
2. The symbol **V** used in conjunction with **P** is redundant and is not required–e.g., **VPP99** is equivalent to **PP99**, and **99PV** is equivalent to **99P**.
3. The symbols **P** and **V** do not count in the size of an item in standard data format.
4. The symbol **S** is counted in the size of an item in standard data format only if **SIGN . . . SEPARATE** has been specified.
5. A numeric edited item must contain either at least one **9 Z *** or else at least two **+ - $**. A numeric edited item cannot consist entirely of **9 P V** symbols (which would be numeric category).
6. If all digits are represented by **Z** or floating **+ - $** and the data has the value zero, the entire data item will be spaces. If all digits are represented by ***** and the data has the value zero, the data item will be all asterisks except for the actual decimal point. Otherwise, replacement will occur left of either the decimal point or the first non-zero digit represented by an insertion symbol, whichever is farther to the left.
7. Any **, B 0 /** insertion characters embedded in **Z** or ***** zero suppression symbols will be replaced by space or asterisks, respectively, if the digit position to the left has a leading zero suppressed by inserting space or asterisk.
8. The second floating character from the left represents the leftmost limit of numeric data that can be stored. A single floating character is inserted immediately to the left of the first non-zero digit (or the decimal point) in a position represented by floating **+ - $** or by **, B 0 /**; and any other positions back to the first floating **+ - $** are replaced with spaces.
9. If the **CURRENCY SIGN** clause is specified (**SPECIAL-NAMES** paragraph), the character specified as the currency symbol is used instead of **$** in the **PICTURE character-string**. It may be any character in the computer character set except **0 1 2 3 4 5 6 7 8 9 A B C D L P R S V X Z * + - , . ; ( ) " / =** or space.
10. If the **DECIMAL-POINT IS COMMA** clause is stated (**SPECIAL-NAMES** paragraph), the rules for period (**.**) and comma (**,**) are exchanged.

SYMBOL	REPRESENTS	MAY APPEAR WITH	RESTRICTIONS	NOTES
		ALPHANUMERIC EDITED		
X A 9	Any allowable character in the computer character set	X A 9 B 0 /	At least one X or else at least one A must be present	
B	Space character insertion	X A 9 B 0 /	At least one B 0 or / must be present; cannot consist entirely of A and B (which would be alphabetic category)	
0	Zero (0) character insertion			
/	Slash (/) character insertion			
		NUMERIC EDITED		
9	Numeric character	any other except A X	May not precede Z * $ or floating + – ; may not appear if Z * or floating $ + – occurs to right of decimal point position	1, 5
Z	Numeric character; replace leading zeros with spaces	9 Z P . V CR DB , B 0 / and single $ + –	May not follow 9 ; if it occurs right of decimal point position, all digits must be represented by P or itself	1, 5, 6, 7
*	Numeric character; replace leading zeros with asterisk (*) characters	9 * P . V CR DB , B 0 / and single $ + –		
floating $	Numeric character; insert currency symbol left of first non-zero digit.	9 $ P . V CR DB , B 0 / and single + –	May not follow 9 ; if it occurs right of decimal point position, all digits must be represented by P or $	1, 5, 6, 8, 9
floating +	Numeric character; to left of first non-zero character insert minus (–) if negative, else insert plus (+)	9 + P . V , B 0 / and single $	May not follow 9 ; if it occurs right of decimal point position, all digits must be represented by P or itself	1, 5, 6, 8
floating –	Numeric character; to left of first non-zero character insert minus (–) if negative, else insert space	9 – P . V , B 0 / and single $		
P	Assumed decimal scaling position	9 Z * + – $ P V , B 0 /	Must either precede or follow all digit positions represented by 9 Z * or floating + – $	1, 2, 3
single $	Insert currency symbol	9 Z * + – P CR DB . V , B 0 /	Leftmost except for single + –	9
single +	Insert minus (–) if negative; else insert plus (+)	9 Z * $ P . V , B 0 /	Either leftmost or rightmost	
single –	Insert minus (–) if negative; else insert space character			
CR	Insert two characters "CR" if negative; else insert two spaces	9 Z * $ P . V , B 0 /	Rightmost	
DB	Insert two characters "DB" if negative; else insert two spaces			
.	Actual decimal point	9 Z * $ + – CR DB , B 0 /	May not be rightmost; only one . allowed	10
V	Location of assumed decimal point within item	9 Z * $ + – P CR DB , B 0 /	Only one V allowed	2, 3
,	Comma (,) character insertion	any other except A X	May not be rightmost	7, 8, 10
B	Space character insertion	any other except A X		7, 8
0	Zero (0) character insertion			
/	Slash (/) character insertion			

SPECIAL REGISTERS

Special registers are compiler generated storage areas into which automatically stored information is produced in conjunction with the use of certain COBOL features.

DEBUG-ITEM—Provides information about the condition that caused the execution of a debugging section with the following items implicitly described:

- **DEBUG-LINE**—Implementor-defined means of identification of particular source statement.

- **DEBUG-NAME**—Contains first 30 characters of the name (**file-name**, **identifier**, **procedure-name** or **cd-name**) that caused the debugging section to be executed.

- **DEBUG-SUB-1**, **DEBUG-SUB-2**, **DEBUG-SUB-3**—If the referenced data item is **subscripted**, the occurrence number of each level is entered in these items respectively as necessary.

- **DEBUG-CONTENTS**—Contains information concerning where the debug is taking place, e.g., "START PROGRAM," "SORT OUTPUT," the entire contents of a record which is read, etc.

The implicit description of **DEBUG-ITEM** is:

```
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).
```

LINAGE-COUNTER—Register(s) generated by the presence of a **LINAGE** clause in an **FD** entry. It points to the line at which the device is positioned within the current page body. It may be referenced (qualified by **file-name** if more than one used) but not modified by **PROCEDURE DIVISION** statements. It can represent a range of 1 through the value in **data-name-5** or **integer-5** in the **FD** syntax diagram.

LINE-COUNTER—Register(s) generated for each **RD** entry. It is used to determine the vertical positioning of the report. The Report Writer Control Section maintains the value of this register(s) which may be accessed but not modified by **PROCEDURE DIVISION** statements. It can represent a range of 0 through 999999 and has an implicit description of PICTURE 9(6).

PAGE-COUNTER—Register(s) generated for each **RD** entry, that is used by the program to number the pages of a report. The Report Writer Control Section maintains the value of this register(s) but it *may* be altered by a **PROCEDURE DIVISION** statement. It can represent a range of 1 through 999999 and has an implicit description of PICTURE 9(6).

IDENTIFIER FORMAT

FORMAT 1

$$\text{data-name-1}\left[\begin{Bmatrix}\underline{\text{OF}}\\\underline{\text{IN}}\end{Bmatrix}\text{data-name-2}\right]\left[(\text{subscript-1}\left[,\text{subscript-2}\left[,\text{subscript-3}\right]\right])\right]$$

FORMAT 2

$$\text{data-name-1}\left[\begin{Bmatrix}\underline{\text{OF}}\\\underline{\text{IN}}\end{Bmatrix}\text{data-name-2}\right]\ldots\left[(\begin{Bmatrix}\text{index-name-1}\left[\{\pm\}\text{ literal-2}\right]\\\text{literal-1}\end{Bmatrix}\right.$$
$$\left.\left[,\begin{Bmatrix}\text{index-name-2}\left[\{\pm\}\text{ literal-4}\right]\\\text{literal-3}\end{Bmatrix}\left[,\begin{Bmatrix}\text{index-name-3}\left[\{\pm\}\text{ literal-6}\right]\\\text{literal-5}\end{Bmatrix}\right]\right])\right]$$

COPY STATEMENT FORMAT

$$\underline{\text{COPY}}\text{ text-name}\left[\begin{Bmatrix}\underline{\text{OF}}\\\underline{\text{IN}}\end{Bmatrix}\text{library-name}\right]$$
$$\left[\underline{\text{REPLACING}}\left\{\begin{Bmatrix}\text{==pseudo-text-1==}\\\text{identifier-1}\\\text{literal-1}\\\text{word-1}\end{Bmatrix}\underline{\text{BY}}\begin{Bmatrix}\text{==pseudo-text-2==}\\\text{identifier-2}\\\text{literal-2}\\\text{word-2}\end{Bmatrix}\right\}\ldots\right]$$

CONDITION FORMAT

RELATION CONDITION

```
                           ⎧ IS [NOT] GREATER THAN ⎫
⎧ identifier-1           ⎫ ⎪ IS [NOT] LESS THAN    ⎪ ⎧ identifier-2           ⎫
⎪ literal-1              ⎪ ⎪ IS [NOT] EQUAL TO     ⎪ ⎪ literal-2              ⎪
⎨ arithmetic-expression-1⎬ ⎨ IS [NOT] >            ⎬ ⎨ arithmetic-expression-2⎬
⎩ index-name-1           ⎭ ⎪ IS [NOT] <            ⎪ ⎩ index-name-2           ⎭
                           ⎩ IS [NOT] =            ⎭
```

CLASS CONDITION

```
                   ⎧ NUMERIC    ⎫
identifier IS [NOT]⎨            ⎬
                   ⎩ ALPHABETIC ⎭
```

SIGN CONDITION

```
                              ⎧ POSITIVE ⎫
arithmetic-expression is [NOT]⎨ NEGATIVE ⎬
                              ⎩ ZERO     ⎭
```

CONDITION-NAME CONDITION

condition-name

SWITCH-STATUS CONDITION

condition-name

NEGATED SIMPLE CONDITION

NOT simple-condition

COMBINED CONDITION

```
          ⎧⎧ AND ⎫           ⎫
condition ⎨⎨     ⎬ condition ⎬ ...
          ⎩⎩ OR  ⎭           ⎭
```

ABBREVIATED COMBINED RELATION CONDITION

```
                   ⎧⎧ AND ⎫                                    ⎫
relation-condition ⎨⎨     ⎬ [NOT] [relational-operator] object ⎬ ...
                   ⎩⎩ OR  ⎭                                    ⎭
```

Note:

When parentheses are not used or when parenthesized conditions are at the same level of inclusiveness, the following order of evaluation is observed:

1. Values are established for any arithmetic expression.

2. Truth values for simple conditions are established.

3. Truth values for negated simple conditions are established.

4. Truth values for combined conditions are established with all combinations of **AND** evaluated first followed by **OR**.

5. Truth values for negated combined conditions are established. (A negated combined condition is **NOT** followed by a combined condition in parentheses.)

ARITHMETIC EXPRESSIONS

ARITHMETIC OPERATORS

BINARY	*MEANING*
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

UNARY	
+	Effect of multiplication by numeric literal +1
-	Effect of multiplication by numeric literal −1

FORMATION RULES

1) Arithmetic expressions may only begin with the symbols (+ - or a variable (identifier or literal) and may only end with) or a variable. There must be a one-to-one correspondence between left and right parenthesis, with each left parenthesis to the left of its right parenthesis.
2) Parentheses may be used to specify the order in which elements of the expression are to be evaluated or they may be used to eliminate the ambiguities in logic.
3) 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 the order of execution of consecutive operations of the same hierarchical level is from left to right with the following hierarchical order implied:

   1st – Unary + -
   2nd – Exponentiation**
   3rd – Multiplication and division * /
   4th – Addition and subtraction + -
4) Allowable combinations of operators, variables, and parentheses in arithmetic expressions are:

FIRST SYMBOL	SECOND SYMBOL				
	VARIABLE	BINARY	UNARY	(	)
VARIABLE	NO	YES	NO	NO	YES
BINARY	YES	NO	YES	YES	NO
UNARY	YES	NO	NO	YES	NO
(	YES	NO	YES	YES	NO
)	NO	YES	NO	NO	YES

IDENTIFICATION DIVISION

```
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

GENERAL FORMAT

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. computer-name [WITH DEBUGGING MODE]

OBJECT COMPUTER. computer-name

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

    [ , PROGRAM COLLATING SEQUENCE IS alphabet-name]

    [ , SEGMENT-LIMIT IS segment-number]

[ SPECIAL-NAMES. [ , implementor-name

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

    [                     { STANDARD-1                                                 } ]
    [                     { NATIVE                                                     } ]
    [                     { implementor-name                                           } ]
    [ , alphabet-name IS  {             [ { THROUGH }                    ]             } ] ...
    [                     { literal-1   [ { THRU    } literal-2          ]             } ]
    [                     {             [ ALSO literal-3 [ , ALSO literal-4] ...       } ]
    [                     {   [             [ { THROUGH }                    ]  ]      } ]
    [                     {   [ , literal-5 [ { THRU    } literal-6          ]  ]      } ]
    [                     {   [             [ ALSO literal-7 [ , ALSO literal-8] ...] ] } ]

    [ , CURRENCY SIGN IS literal-9]

    [ , DECIMAL-POINT IS COMMA] . ]

[ INPUT-OUTPUT SECTION.
  FILE CONTROL.

      { file-control-entry } ...

  [ I-O-CONTROL.

      [ ; RERUN [ ON { file-name-1       } ]
      [              { implementor-name  } ]

      [          { { [END OF] { REEL } } OF file-name-2 } ]
      [          { {          { UNIT } }                } ]
      [   EVERY  { { integer-1 RECORDS }                } ] ...
      [          { integer-2 CLOCK-UNITS                } ]
      [          { condition-name                       } ]

      [          [ RECORD     ]                                             ]
      [ ; SAME   [ SORT       ] AREA FOR file-name-3 { , file-name-4 } ...  ] ...
      [          [ SORT-MERGE ]                                             ]

      [ ; MULTIPLE FILE TAPE CONTAINS file-name-5 [ POSITION integer-3]
           [ , file-name-6 [POSITION integer-4] ] ... ] ... ]
```

file-control-entry

FORMAT 1:

```
SELECT [OPTIONAL] file-name

    ASSIGN TO implementor-name-1 [ , implementor-name-2] ...

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

    [ ; ORGANIZATION IS SEQUENTIAL]

    [ ; ACCESS MODE IS SEQUENTIAL]

    [ ; FILE STATUS IS data-name-1] .
```

FORMAT 2:

```
SELECT file-name

    ASSIGN TO implementor-name-1 [ , implementor-name-2] ...

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

    ; ORGANIZATION IS RELATIVE

    [                 {SEQUENTIAL [ , RELATIVE KEY IS data-name-1]}]
    [ ; ACCESS MODE IS{                                           }]
    [                 {{RANDOM }                                  }]
    [                 {{DYNAMIC}, RELATIVE KEY IS data-name-1     }]

    [ ; FILE STATUS IS data-name-2] .
```

FORMAT 3:

```
SELECT file-name

    ASSIGN TO implementor-name-1 [ , implementor-name-2] ...

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

    ; ORGANIZATION IS INDEXED

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

    ; RECORD KEY IS data-name-1

    [ ; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES] ] ...

    [ ; FILE STATUS IS data-name-3] .
```

FORMAT 4:

```
SELECT file-name ASSIGN TO implementor-name-1 [ , implementor-name-2] ...
```

DATA DIVISION

GENERAL FORMAT

```
DATA DIVISION.

[FILE SECTION.

{FD file-name

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

    [ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

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

    [ ; VALUE OF implementor-name-1 IS {data-name-1}
                                       {literal-1  }

        [ , implementor-name-2 IS {data-name-2} ] ...]
                                  {literal-2  }

    [ ; DATA {RECORD IS  } data-name-3 [ , data-name-4] ...]
             {RECORDS ARE}

    [ ; LINAGE IS {data-name-5} LINES [ , WITH FOOTING AT {data-name-6} ]
                  {integer-5  }                           {integer-6  }

        [ , LINES AT TOP {data-name-7} ] [ , LINES AT BOTTOM {data-name-8} ]]
                         {integer-7  }                       {integer-8  }

    [ ; CODE-SET IS alphabet-name]

    [ ; {REPORT IS  } report-name-1 [ , report-name-2] ... ].
        {REPORTS ARE}

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

[SD file-name

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

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

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

[WORKING-STORAGE SECTION.

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

 NKAGE SECTION.

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

[COMMUNICATION SECTION.

[communication-description-entry

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

[REPORT SECTION.

[RD report-name

    [ ; CODE literal-1]

    [ ; {CONTROL IS  } {data-name-1 [ , data-name-2] ...                 } ]
        {CONTROLS ARE} {FINAL [ , data-name-1 [ , data-name-2] ... ]     }

    [ ; PAGE [LIMIT IS  ] integer-1 [LINE ] [ , HEADING integer-2]
             [LIMITS ARE]           [LINES]

        [ , FIRST DETAIL integer-3] [ , LAST DETAIL integer-4]

        [ , FOOTING integer-5] ] .

{report-group-description-entry} ... ] ...]
```

data-description-entry

FORMAT 1:

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

    [ ; REDEFINES data-name-2]
    [ ; {PICTURE} IS character-string]
        {PIC    }
    [ ; [USAGE IS] {COMPUTATIONAL}]
                   {COMP         }
                   {DISPLAY      }
                   {INDEX        }
    [ ; [SIGN IS] {LEADING } [SEPARATE CHARACTER]]
                  {TRAILING}
    [ ; OCCURS {integer-1 TO integer-2 TIMES DEPENDING ON data-name-3}
               {integer-2 TIMES                                      }
        [ {ASCENDING } KEY IS data-name-4 [ , data-name-5] ...] ...
          {DESCENDING}
        [INDEXED BY index-name-1 [ , index-name-2] ...]]
    [ ; {SYNCHRONIZED} [LEFT ]]
        {SYNC        } [RIGHT]
    [ ; {JUSTIFIED} RIGHT]
        {JUST     }
    [ ; BLANK WHEN ZERO]
    [ ; VALUE IS literal] .
```

FORMAT 2:

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

FORMAT 3:

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

communications-description-entry

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-1, data-name-2, ... , data-name-11]              ]
```

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 [ , index-name-2] ... ] ]
    [ ; ERROR KEY IS data-name-4]
    [ ; SYMBOLIC DESTINATION IS data-name-5] .
```

report-group-description-entry

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] .
```

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     }

   [ ; SOURCE IS identifier-1                                  ]
   [ ; VALUE IS literal                                        ]
   [ { ; SUM identifier-2 [ , identifier-3] ...                ]
   [   [UPON data-name-2 [ , data-name-3] ... ] } ...          ]
   [   [ RESET ON { data-name-4 } ]                            ]
   [   [          { FINAL       } ]                            ]

   [ ; [USAGE IS] DISPLAY] .
```

PROCEDURE DIVISION

GENERAL FORMAT

FORMAT 1:

```
PROCEDURE DIVISION [USING data-name-1 [ , data-name-2] ... ] .
[DECLARATIVES.
{section-name SECTION [segment-number] .  declarative-sentence
[paragraph-name. [sentence] ... ] ... }...
END DECLARATIVES.]
{section-name SECTION [segment-number] .
[paragraph-name. [sentence] ... ] ... }...
```

FORMAT 2:

```
PROCEDURE DIVISION [USING data-name-1 [ , data-name-2] ... ] .
{paragraph-name. [sentence] ...} ...
```

declarative-sentence

FORMAT 1:

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

FORMAT 2:

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

FORMAT 3:

```
USE BEFORE REPORTING identifier.
```

FORMAT 4:

```
                         { cd-name-1                              }
                         { [ALL REFERENCES OF] identifier-1       }
USE FOR DEBUGGING ON     { file-name-1                            }
                         { procedure-name-1                       }
                         { ALL PROCEDURES                         }

   [ { cd-name-2                          } ]
   [ { [ALL REFERENCES OF] identifier-2   } ]
   [ { file-name-2                        } ] ... .
   [ { procedure-name-2                   } ]
   [ { ALL PROCEDURES                     } ]
```

VERBS

FORMAT 1:

$$\underline{\text{ACCEPT}}\ \text{identifier}\ [\underline{\text{FROM}}\ \text{mnemonic-name}]$$

FORMAT 2:

$$\underline{\text{ACCEPT}}\ \text{identifier}\ \underline{\text{FROM}} \left\{\begin{matrix}\underline{\text{DATE}}\\ \underline{\text{DAY}}\\ \underline{\text{TIME}}\end{matrix}\right\}$$

FORMAT 3:

$$\underline{\text{ACCEPT}}\ \text{cd-name MESSAGE}\ \underline{\text{COUNT}}$$

FORMAT 1:

$$\underline{\text{ADD}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} \left[\begin{matrix}\text{, identifier-2}\\ \text{, literal-2}\end{matrix}\right] \ldots\ \underline{\text{TO}}\ \text{identifier-m}\ [\underline{\text{ROUNDED}}]$$

$$[\text{, identifier-n}\ [\underline{\text{ROUNDED}}]]\ \ldots\ [\ ;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

FORMAT 2:

$$\underline{\text{ADD}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} , \left\{\begin{matrix}\text{identifier-2}\\ \text{literal-2}\end{matrix}\right\} \left[\begin{matrix}\text{, identifier-3}\\ \text{, literal-3}\end{matrix}\right] \ldots$$

$$\underline{\text{GIVING}}\ \text{identifier-m}\ [\underline{\text{ROUNDED}}]\ [\text{, identifier-n}\ [\underline{\text{ROUNDED}}]]\ \ldots$$

$$[\ ;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

FORMAT 3:

$$\underline{\text{ADD}} \left\{\begin{matrix}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{matrix}\right\} \text{identifier-1}\ \underline{\text{TO}}\ \text{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[\ ;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

$$\underline{\text{ALTER}}\ \text{procedure-name-1}\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ \text{procedure-name-2}$$

$$[\text{, procedure-name-3}\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ \text{procedure-name-4}]\ \ldots$$

$$\underline{\text{CALL}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} [\underline{\text{USING}}\ \text{data-name-1}\ [\text{, data-name-2}]\ \ldots]$$

$$[\ ;\ \text{ON}\ \underline{\text{OVERFLOW}}\ \text{imperative-statement}]$$

$$\underline{\text{CANCEL}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} \left[\begin{matrix}\text{, identifier-2}\\ \text{, literal-2}\end{matrix}\right] \ldots$$

FORMAT 1:

$$\underline{\text{CLOSE}}\ \text{file-name-1} \left[\begin{matrix}\left\{\begin{matrix}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{matrix}\right\} \left[\begin{matrix}\text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}}\\ \text{FOR}\ \underline{\text{REMOVAL}}\end{matrix}\right]\\ \text{WITH} \left\{\begin{matrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{matrix}\right\}\end{matrix}\right]$$

$$\left[\text{, file-name-2} \left[\begin{matrix}\left\{\begin{matrix}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{matrix}\right\} \left[\begin{matrix}\text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}}\\ \text{FOR}\ \underline{\text{REMOVAL}}\end{matrix}\right]\\ \text{WITH} \left\{\begin{matrix}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{matrix}\right\}\end{matrix}\right]\right] \ldots$$

FORMAT 2:

$$\underline{\text{CLOSE}}\ \text{file-name-1}\ [\text{WITH}\ \underline{\text{LOCK}}]\ [\text{, file-name-2}\ [\text{WITH}\ \underline{\text{LOCK}}]]\ \ldots$$

$$\underline{\text{COMPUTE}}\ \text{identifier-1}\ [\underline{\text{ROUNDED}}]\ [\text{, identifier-2}\ [\underline{\text{ROUNDED}}]]\ \ldots$$

$$= \text{arithmetic-expression}\ [\ ;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{imperative-statement}]$$

$$\underline{\text{DELETE}}\ \text{file-name RECORD}\ [\ ;\ \underline{\text{INVALID}}\ \text{KEY imperative-statement}]$$

$$\underline{\text{DISABLE}} \left\{\begin{matrix}\underline{\text{INPUT}}\ [\underline{\text{TERMINAL}}]\\ \underline{\text{OUTPUT}}\end{matrix}\right\} \text{cd-name WITH}\ \underline{\text{KEY}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\}$$

$$\underline{\text{DISPLAY}} \left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} \left[\begin{matrix}\text{, identifier-2}\\ \text{, literal-2}\end{matrix}\right] \ldots\ [\underline{\text{UPON}}\ \text{mnemonic-name}]$$

FORMAT 1:

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

    [ , identifier-3 [ROUNDED] ] ... [ , ON SIZE ERROR imperative-statement]
```

FORMAT 2:

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

    [ , identifier-4 [ROUNDED] ] ... [ ; ON SIZE ERROR imperative-statement]
```

FORMAT 3:

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

    [ , identifier-4 [ROUNDED] ] ... [ ; ON SIZE ERROR imperative-statement]
```

FORMAT 4:

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

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

FORMAT 5:

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

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

```
ENABLE {INPUT   [TERMINAL]}  cd-name WITH KEY {identifier-1}
       {OUTPUT            }                   {literal-1   }

ENTER language-name [routine-name]

EXIT [PROGRAM]

GENERATE {data-name  }
         {report-name}
```

FORMAT 1:

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

FORMAT 2:

```
GO TO procedure-name-1 [ , procedure-name-2] ... , procedure-name-n

    DEPENDING ON identifier

IF condition; {statement-1  }  {, ELSE statement-2    }
              {NEXT SENTENCE}  {; ELSE NEXT SENTENCE  }

INITIATE report-name-1 [ , report-name-2] ...
```

FORMAT 1:

```
INSPECT identifier-1 TALLYING

  {                      {  {{ALL    } {identifier-3}}                                   }
  { , identifier-2 FOR   {  {{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   }]}                          }

MERGE file-name-1 ON  {ASCENDING } KEY data-name-1 [ , data-name-2] ...
                      {DESCENDING}
       [ON  {ASCENDING }   KEY data-name-3 [ , data-name-4] ... ] ...
       [    {DESCENDING}                                        ]
   [COLLATING SEQUENCE IS alphabet-name]

   USING file-name-2, file-name-3 [ , file-name-4] ...

   { OUTPUT PROCEDURE IS section-name-1      [{THROUGH}   section-name-2] }
   {                                         [{THRU   }                 ] }
   { GIVING file-name-5                                                   }
```

FORMAT 1:

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

FORMAT 2:

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

FORMAT 1:

```
 ULTIPLY {identifier-1}  BY identifier-2 [ROUNDED]
         {literal-1   }
   [ , identifier-3 [ROUNDED] ] ... [ ; ON SIZE ERROR imperative-statement ]
```

FORMAT 2:

```
MULTIPLY{identifier-1}  BY {identifier-2}  GIVING identifier-3 [ROUNDED]
        {literal-1   }     {literal-2   }
   [ , identifier-4 [ROUNDED] ] ... [ ; ON SIZE ERROR imperative-statement]
```

FORMAT 1:

```
       { INPUT file-name-1 [REVERSED       ]                    }
       {                   [WITH NO REWIND ]                    }
       {     [ , file-name-2  [REVERSED       ] ] ...           }
       {     [                [WITH NO REWIND ] ]               }
OPEN   { OUTPUT file-name-3 [WITH NO REWIND]                    } ...
       {     [ , file-name-4 [WITH NO REWIND] ] ...             }
       { I-O file-name-5 [ , file-name-6] ...                   }
       { EXTEND file-name-7 [ , file-name-8] ...                }
```

FORMAT 2:

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

FORMAT 1:

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

FORMAT 2:

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

FORMAT 3:

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

FORMAT 4:

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

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

       BY  {identifier-4}   UNTIL condition-1
           {literal-3   }

                                     {identifier-6  }
   [ AFTER  {identifier-5  }  FROM   {index-name-4  }
            {index-name-3  }         {literal-3     }

       BY  {identifier-7}  UNTIL condition-2

                                     {identifier-9  }
   [ AFTER  {identifier-8  }  FROM   {index-name-6  }
            {index-name-5  }         {literal-5     }

       BY  {identifier-10}  UNTIL condition-3 ] ]
           {literal-6    }
```

FORMAT 1:

READ file-name RECORD [INTO identifier] [; AT END imperative-statement]

FORMAT 2:

READ file-name [NEXT] RECORD [INTO identifier]

[; AT END imperative-statement]

FORMAT 3:

READ file-name RECORD [INTO identifier] [; INVALID KEY imperative-statement]

FORMAT 4:

READ file-name RECORD [INTO identifier]

[; KEY IS data-name]

[; INVALID KEY imperative-statement]

```
RECEIVE cd-name   {MESSAGE} INTO identifier-1 [ ; NO DATA imperative-statement]
                  {SEGMENT}
```

RELEASE record-name [FROM identifier]

RETURN file-name RECORD [INTO identifier] ; AT END imperative-statement

FORMAT 1:

REWRITE record-name [FROM identifier]

FORMAT 2:

REWRITE record-name [FROM identifier] [; INVALID KEY imperative-statement]

FORMAT 1:

```
SEARCH identifier-1 [VARYING {identifier-2  }] [ ; AT END imperative-statement-1]
                             {index-name-1  }

    ; WHEN condition-1  {imperative-statement-2}
                        {NEXT SENTENCE         }

    [ ; WHEN condition-2  {imperative-statement-3} ] ...
                          {NEXT SENTENCE         }
```

FORMAT 2:

```
SEARCH ALL identifier-1 [ ; AT END imperative-statement-1]

   ; WHEN {data-name-1  {IS EQUAL TO}  {identifier-3             } }
          {             {IS =       }  {literal-1                } }
          {                            {arithmetic-expression-1  } }
          {condition-name-1                                        }

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

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

FORMAT 1:

```
SEND cd-name FROM identifier-1
```

FORMAT 2:

```
                                      {WITH identifier-2}
SEND cd-name [FROM identifier-1]      {WITH ESI         }
                                      {WITH EMI         }
                                      {WITH EGI         }

   [ {BEFORE}  ADVANCING { {identifier-3}  [LINE ] } ]
     {AFTER }            { {integer     }  [LINES] } 
                         {                         }
                         { {mnemonic-name}         }
                         { {PAGE         }         }
```

FORMAT 1:

```
SET {identifier-1  [ , identifier-2]  ...}  TO  {identifier-3 }
    {index-name-1  [ , index-name-2]  ...}      {index-name-3 }
                                                {integer-1    }
```

RMAT 2:

```
ET index-name-4 [ , index-name-5] ... {UP BY  }  {identifier-4}
                                      {DOWN BY}  {integer-2   }
```

```
SORT file-name-1 ON {ASCENDING } KEY data-name-1 [ , data-name-2] ...
                    {DESCENDING}

     [ON {ASCENDING } KEY data-name-3 [ , data-name-4] ...] ...
         {DESCENDING}

   [COLLATING SEQUENCE IS alphabet-name]

   { INPUT PROCEDURE IS section-name-1 [{THROUGH} section-name-2] }
   {                                    {THRU   }                 }
   { USING file-name-2 [ , file-name-3] ...                       }

   { OUTPUT PROCEDURE IS section-name-3 [{THROUGH} section-name-4] }
   {                                     {THRU   }                 }
   { GIVING file-name-4                                            }
```

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

   [ ; INVALID KEY imperative-statement]
```

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

```
STRING {identifier-1} [, identifier-2] ... DELIMITED BY {identifier-3}
       {literal-1   } [, literal-2   ]                  {literal-3   }
                                                        {SIZE        }

   [ {identifier-4} [, identifier-5] ... DELIMITED BY {identifier-6} ]
     {literal-4   } [, literal-5   ]                  {literal-6   }
                                                      {SIZE        }

   INTO identifier-7 [WITH POINTER identifier-8]

   [ ; ON OVERFLOW imperative-statement]
```

FORMAT 1:

```
SUBTRACT {identifier-1} [, identifier-2] ... FROM identifier-m [ROUNDED]
         {literal-1   } [, literal-2   ]

    [, identifier-n [ROUNDED] ] ... [ ; ON SIZE ERROR imperative-statement]
```

FORMAT 2:

```
SUBTRACT {identifier-1} [, identifier-2] ... FROM {identifier-m}
         {literal-1   } [, literal-2   ]          {literal-m   }

    GIVING identifier-n [ROUNDED] [, identifier-o [ROUNDED] ] ...

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

FORMAT 3:

```
SUBTRACT {CORRESPONDING} identifier-1 FROM identifier-2 [ROUNDED]
         {CORR         }

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

```
SUPPRESS PRINTING

TERMINATE report-name-1 [ , report-name-2] ...

UNSTRING identifier-1

    [DELIMITED BY [ALL] {identifier-2} [, OR [ALL] {identifier-3}] ... ]
    [                   {literal-1   } [           {literal-2   }]     ]

    INTO identifier-4 [ , DELIMITER IN identifier-5] [ , COUNT IN identifier-6]

        [ , identifier-7 [ , DELIMITER IN identifier-8] [ , COUNT IN identifier-9] ] ...

    [WITH POINTER identifier-10] [TALLYING IN identifier-11]

    [ ; ON OVERFLOW imperative-statement]
```

FORMAT 1:

```
WRITE record-name [FROM identifier-1]

    [                          {{identifier-2} [LINE ]} ]
    [ {BEFORE}  ADVANCING      {{integer     } [LINES]} ]
    [ {AFTER }                 {                      } ]
    [                          {{mnemonic-name}       } ]
    [                          {{PAGE         }       } ]

    [ ; AT {END-OF-PAGE} imperative-statement ]
    [      {EOP        }                      ]
```

FORMAT 2:

```
WRITE record-name [FROM identifier] [ ; INVALID KEY imperative-statement]
```

RESERVED WORDS

Reserved words are the following and may be used in COBOL programs as specified in the syntax diagrams.

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BLANK
BLOCK
BOTTOM
BY

CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EGI
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ESI
EVERY
EXCEPTION
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTERCHANGE
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

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

MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
RECEIVE
RECORD
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
[illegible]PRESS
[illegible]BOLIC
SYNC
SYNCHRONIZED

TABLE
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TYPE

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WITHIN
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

+
-
*
/
>
<
=

SYSTEM NAMES

For a specific implementation of COBOL, the implementor is expected to define certain system names for his compiler in accordance with American National Standard X3.23-1974. Such a system name is shown in the syntax diagrams as a **language-name**, a **computer-name**, or an **implementor-name**.

Make notes here on specific implementations:

language-name

computer-name

implementor-name

The words which can be used for **implementor-name** depend upon the entry in which the **implementor-name** is used.

□ The **SPECIAL-NAMES** paragraph of the **CONFIGURATION SECTION** of the **ENVIRONMENT DIVISION**

with **mnemonic-names** and/or **condition-names**

with **alphabet-names**

□ In the **file-control-entry** of the **FILE-CONTROL** paragraph of the **INPUT-OUTPUT SECTION** in the **ENVIRONMENT DIVISION** in a **SELECT** clause

□ The **RERUN** clause in the **I-O-CONTROL** paragraph of the **INPUT-OUTPUT SECTION** in the **ENVIRONMENT DIVISION**

□ The **VALUE OF** clause in the **FD** entry within the **FILE SECTION** of the **DATA DIVISION**

ASCII CHARACTER SET

The **STANDARD-1** alphabet consists of the following characters of the American Standard Code for Information Interchange, ASCII:

ASCII Character	Octal Value	Meaning
NUL	000	Null or time fill character
SOH	001	Start of heading
STX	002	Start of text
ETX	003	End of text
EOT	004	End of transmission
ENQ	005	Enquiry (who are you)
ACK	006	Acknowledge
BEL	007	Bell
BS	010	Backspace
HT	011	Horizontal tabulation
LF	012	Line feed (new line)
VT	013	Vertical tabulation
FF	014	Form feed
CR	015	Carriage return
SO	016	Shift out
SI	017	Shift in
DLE	020	Data link escape
DC1	021	Device control 1
DC2	022	Device control 2
DC3	023	Device control 3
DC4	024	Device control 4
NAK	025	Negative acknowledgement
SYN	026	Synchronous idle
ETB	027	End of transmission blocks
CAN	030	Cancel
EM	031	End of medium
SUB	032	Substitute
ESC	033	Escape
FS	034	File separator
GS	035	Group separator
RS	036	Record separator
US	037	Unit separator
SP	040	Space
!	041	Exclamation point
"	042	Quotation mark
#	043	Number sign
$	044	Currency symbol
%	045	Percent
&	046	Ampersand
'	047	Apostrophe or acute accent
(	050	Opening parenthesis
)	051	Closing parenthesis
*	052	Asterisk
+	053	Plus
,	054	Comma
–	055	Hyphen or minus
.	056	Period or decimal point
/	057	Slant
0	060	
1	061	
2	062	
3	063	
4	064	
5	065	
6	066	
7	067	
8	070	
9	071	
:	072	Colon
;	073	Semicolon
<	074	Less than

ASCII Character	Octal Value	Meaning
=	075	Equal
>	076	Greater than
?	077	Question mark
@	100	Commercial at
A	101	
B	102	
C	103	
D	104	
E	105	
F	106	
G	107	
H	110	
I	111	
J	112	
K	113	
L	114	
M	115	
N	116	
O	117	
P	120	
Q	121	
R	122	
S	123	
T	124	
U	125	
V	126	
W	127	
X	130	
Y	131	
Z	132	
[	133	Opening bracket
\	134	Reverse slant
]	135	Closing bracket
^	136	Circumflex
_	137	Underline
`	140	Grave accent
a	141	
b	142	
c	143	
d	144	
e	145	
f	146	
g	147	
h	150	
i	151	
j	152	
k	153	
l	154	
m	155	
n	156	
o	157	
p	160	
q	161	
r	162	
s	163	
t	164	
u	165	
v	166	
w	167	
x	170	
y	171	
z	172	
{	173	Opening brace
|	174	Vertical line
}	175	Closing brace
~	176	Tilde
DEL	177	Delete

COBOL CHARACTER SET

The COBOL character set consists of the 51 characters listed below.

Character	Meaning
0,1, . . . ,9	digit
A,B, . . . ,Z	letter
	space (blank)
+	plus sign
–	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	default currency sign
,	comma (optional decimal point)
;	semicolon
.	period (decimal point)
"	quotation marks
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

U. S. GOVERNMENT PRINTING OFFICE : 1977 O - 225-116