

The IBM logo, consisting of the letters "IBM" in a bold, white, sans-serif font, set against a solid black rectangular background.

Systems Reference Library

**IBM System/360
Disk and Tape Operating Systems
COBOL Language Specifications**

COBOL (Common Business Oriented Language) is similar to English. It was developed by the Conference of Data Systems Languages (CODASYL). COBOL provides a convenient method of coding programs to handle commercial data processing problems.

This publication provides the programmer with rules for writing programs in COBOL for IBM System/360 Disk and Tape Operating Systems. Users unacquainted with COBOL should first familiarize themselves with the publication: COBOL: General Information Manual, Form F28-8053-2.

The titles and abstracts of related publications are listed in the IBM System/360 Bibliography, Form A22-6822.



Fourth Edition, November 1966

This edition, Form C24-3433-3, is a major revision of Form C24-3433-2 and obsoletes it and all earlier editions. It should be reviewed in its entirety.

Changes are indicated by a vertical line to the left of the affected text and to the left of affected parts of figures. A dot (●) next to a figure title or page number indicates that the entire figure or page should be reviewed.

Significant changes and additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for readers' comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Endicott, New York 13760.

PREFACE

This reference publication describes COBOL as implemented for the IBM System/360 Disk and Tape Operating Systems, discusses the four divisions of a COBOL program, and describes the following special features of IBM System/360 Disk and Tape Operating Systems COBOL:

1. Source Program Library Facility
2. Specifications for the Sterling Currency Feature and International Considerations
3. COBOL Debugging Language

Five appendixes are included:

1. A list of definitions of terms in COBOL formats
2. A COBOL word list
3. A discussion of intrarecord slack bytes, and record alignment within block files.
4. A discussion of intermediate results in arithmetic operations
5. Example of a COBOL calling program and a subprogram.

All disk I/O operations can be compiled on the disk system only.

The following features are not currently available for Disk or Tape Operating Systems, and are so identified at the appropriate places in this publication with an asterisk.

1. The RERUN clause
2. The Sterling Currency feature

The following features are not currently available for Tape Operating System, and are so identified at the appropriate places in this publication with two asterisks.

1. The APPLY WRITE ONLY clause
2. The USE AFTER STANDARD ERROR clause

The restrictions described under the PERFORM...VARYING option that are identified by three asterisks apply to Tape Operating System only.

The following features are IBM extensions to COBOL for IBM System/360 Disk and Tape Operating Systems, and are marked adjacent to the applicable features in this publication by the word "Ext" at their first appearance in the publication.

1. The ORGANIZATION clause
2. Internal and external floating-point items and floating-point literals
3. The overflow-name test-condition
4. The RECORD-KEY clause

5. The Linkage Section of the Data Division
6. Options 1 and 2 of the USE sentence
7. The REWRITE statement
8. The TRANSFORM statement
9. The Debugging Language
10. Sterling Currency feature

ACKNOWLEDGEMENT

The following extract from Government Printing Office Form Number 1962-0668996 is presented for the information and guidance of the user:

"This publication is based on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organizations participating in the original development were:

Air Material Command, United States Air Force
Bureau of Standards, United States Department of Commerce
Burroughs Corporation
David Taylor Model Basin, Bureau of Ships, United States
Navy
Electronic Data Processing Division,
Minneapolis-Honeywell Regulator Company
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
UNIVAC Division of Sperry Rand Corporation

"In addition to the organizations listed above, the following other organizations participated in the work of the Maintenance Group:

Allstate Insurance Company
The Bendix Corporation, Computer Division
Control Data Corporation
E. I. du Pont de Nemours and Company
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
The National Cash Register Company
Philco Corporation
Royal McBee Corporation
Standard Oil Company (New Jersey)
United States Steel Corporation

"This COBOL-61 manual is the result of contributions made by all of the above-mentioned organizations. No warranty, expressed or implied, is made by any contributor or by the 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.

"It is reasonable to assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programming. However, this protection can be positively assured only by individual implementors.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC,* Programming for the UNIVAC* I and II, Data Automation Systems 1958, 1959, Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, have specifi-

cally 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.

"Any organization interested in reproducing the COBOL report and initial specifications, in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgement of the source, but need not quote this entire section."

* Trademark of Sperry Rand Corporation

CONTENTS

SECTION 1: BASIC FACTS.	9	General Description.	31
Machine Requirements	9	Organization Of The Data Division.	31
Character Set.	10	File Section.	31
Punctuation.	11	Working Storage Section	32
Word Formation.	11	Linkage Section	32
Types of Names	12	Concepts of Data Description	32
Data-Names.	12	Levels of Data Items.	32
External-Names.	12	Condition-Names	33
Procedure-Names	12	Data-Names.	34
Paragraph-Names	13	Literals.	35
Qualification of Names	13	Figurative Constants.	36
COBOL Program Sheet.	13	Types of Data Items	37
Sequence Number: (Columns 1-6).	14	Alignment of Data Fields	41
Continuation Indicator: (Column 7).	14	File Section	41
Source Program Statements: (Columns		File and Record Handling.	41
8-72).	14	File Section Entries.	43
Program Identification Code:		Record Description Entry	46
(Columns 73-80).	14	Group Item.	47
Margin Restrictions	14	Elementary Items.	47
Continuation of Non-Numeric		Item Information Clauses.	50
Literals	15	Working-Storage Section.	61
Format Notation.	15	Linkage Section.	61
SECTION 2: COBOL PROCESSING		SECTION 6: PROCEDURE DIVISION	63
CAPABILITIES.	17	Purpose.	63
Input/Output Processing.	17	Syntax	63
Data Organization	17	Sections.	63
Access Methods.	18	Paragraphs.	63
Keys	18	Sentences	64
Accessing a Direct File Randomly.	18	Expressions	64
Accessing an Indexed File Randomly.	19	Statements.	64
Accessing an Indexed or Direct File		Conditionals.	65
Sequentially	19	Compiler-Directing Declaratives	74
Creation of an Indexed File	20	Continued Processing of File.	76
Creation of a Direct File	20	COBOL Verbs.	77
SECTION 3: IDENTIFICATION DIVISION	22	Input/Output Statements	77
SECTION 4: ENVIRONMENT DIVISION.	23	Data Manipulation Statements.	85
General Description.	23	Arithmetic Statements and Options	92
Configuration Section.	23	Procedure Branching Statements.	96
Input-Output Section	24	Compiler-Directing Statements	104
File-Control Paragraph.	25	SECTION 7: SOURCE PROGRAM LIBRARY	
I-O Control Paragraph	28	FACILITY.	107
SECTION 5: DATA DIVISION	31	Copy Clause	107
		INCLUDE Statement.	108
		SECTION 8: STERLING CURRENCY FEATURE	
		AND INTERNATIONAL CONSIDERATIONS.	109

Sterling Currency Feature.109	Intrarecord Slack Bytes.126
Sterling Non-Report110	Coding for a Usage Clause:.127
Sterling Sign Representation.110	Coding of an OCCURS Clause.128
Sterling Report111	Record Alignment Within Block Files.129
International Considerations.114	Block Files Example Coding.130
SECTION 9: COBOL DEBUGGING LANGUAGE.115	Block File Example Coding Showing	
TRACE115	the Filler for Alignment (Repeated	
EXHIBIT115	Here for Clarity).131
ON (Count-Conditional Statement).116	Some Rules to Remember.131
Compile-Time Debugging Packet.117	APPENDIX D: INTERMEDIATE RESULTS IN	
APPENDIX A: GLOSSARY OF LOWER-CASE		ARITHMETIC OPERATIONS133
WORDS IN COBOL FORMATS.118	Intermediate Results.133
APPENDIX B: DISK AND TAPE OPERATING		Compiler Treatment of Intermediate	
SYSTEMS COBOL WORD LIST124	Results.135
APPENDIX C: INTRARECORD SLACK BYTES		APPENDIX E136
AND RECORD ALIGNMENT IN BLOCK FILES126	INDEX.140

This section defines the minimum machine requirements for COBOL, the COBOL character set, and describes the formation of COBOL words. It also includes special topics such as punctuation, name qualification, and rules for writing COBOL source programs on a program sheet.

MACHINE REQUIREMENTS

Disk Operating System COBOL operates in a 32K byte environment if the disk compiler is allocated 14K bytes of storage.

Tape Operating System COBOL operates in a 16K byte environment if the compiler is allocated 10K bytes of storage.

A summary of all the devices supported by function, including intermediate (work) storage, is as follows:

UNITS USED
BY COBOL
PROCESSOR

	<u>FUNCTIONS</u>			
	Input	Work	Output	List
1403				X
1404*				X
1442	X		X	
1443				X
2501	X			
2520	X		X	
2540	X		X	
2311**	D	D	D	D
2400-series**	X	X	X	X

Notes:

D-Indicates Disk Operating System COBOL only.

* - For continuous forms only.

** - For work files three logical files are required and they must be the same device type.

Compile and execute is provided in all systems if sufficient intermediate storage is available.

Intermediate (work) storage devices may not be mixed. Where 2400 series magnetic tape units are used, a minimum of three (3) units are required.

Expanded instruction sets may be required depending on the specific requirements of the language program utilized as follows:

For COBOL:

SYSTEM REQUIRES: Standard instruction set, decimal arithmetic set.
(Floating-point option is required if floating-point literals are used.)

OBJECT PROGRAM
REQUIRES: Standard instruction set, decimal arithmetic option.
(Floating-point option is required if non-integer exponents or floating-point numbers are used.)

The device type dependency of problem programs is established at compile time. Problem programs compiled by COBOL support the following units:

1403
1404*
1442
1443
1445
2501
2520
2540
2400-series (7 or 9 track)
2311**
2321**

*For continuous forms only.

**For Disk Operating System only.

CHARACTER SET

The complete COBOL character set consists of the following 51 characters:

Digits 0 through 9

Letters A through Z

Special characters:

Blank or space

+ Plus sign

- Minus sign or hyphen

* Check protection symbol, asterisk

/ Slash

= Equal sign

> Inequality sign (greater than)

< Inequality sign (less than)

\$ Dollar sign

, Comma

. Period or decimal point

' Quotation mark

(Left parenthesis

) Right parenthesis

; Semicolon

Of the previous set, the following characters are used for words:

0 through 9

A through Z

- (hyphen)

The following characters are used for punctuation:

' Quotation mark
(Left parenthesis
) Right parenthesis
, Comma
. Period
; Semicolon

The following characters are used in arithmetic expressions:

+ Addition
- Subtraction
* Multiplication
/ Division
** Exponentiation

The following characters are used in relation tests:

> Greater than
< Less than
= Equal to

All of the preceding characters are contained in the COBOL character set. In addition, the programmer can use, as characters in non-numeric literals, any characters (except the quotation mark) included in the IBM Extended Binary-Coded-Decimal Interchange Code; however, such characters may be unacceptable to COBOL for other computers.

PUNCTUATION

The following general rules of punctuation apply in writing COBOL source programs:

1. When any punctuation mark is indicated in a format in this publication, it is required.
2. A period, semicolon or comma, when used, must not be preceded by a space, but must be followed by a space.
3. A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded immediately by a space.
4. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except in non-numeric literals.
5. When an arithmetic operator or an equal sign is used, it must be preceded by a space and followed by another space.
6. When the period or comma, or arithmetic operator characters are used in the PICTURE clause as editing characters, they are governed by rules for report items only.
7. A comma may be used as a separator between successive operands of a statement. A comma or semicolon may be used to separate a series of clauses , and a semicolon or the word THEN may be used to separate a series of statements.

WORD FORMATION

A word is composed of a combination of not more than 30 characters, chosen from the following set of 37 characters:

0 through 9 (digits)
A through Z (letters)
- (hyphen)

A word must not begin or end with a hyphen. A word is ended by a space, or by proper punctuation. Embedded hyphens are permitted. All words in COBOL are either reserved words, which have preassigned meanings in COBOL, or programmer-supplied names. Each type of name is discussed in the section of this publication in which it is first mentioned.

TYPES OF NAMES

There are a number of name types used in writing a COBOL source program; each of which must conform to specific format requirements to be compatible. The name types and their respective formats are:

DATA-NAMES

A data-name must contain at least one alphabetic character, and must be formed according to the rules for word formation.

EXTERNAL-NAMES

An external-name consists of quotation marks enclosing no more than eight alphabetic and numeric characters, the first of which must be an alphabetic character.

PROCEDURE-NAMES

Procedure-names follow the rules for word formation. They may be composed solely of numeric characters, in which case they are equivalent only if they are composed of identical digits and have the same numeric value.

PARAGRAPH-NAMES

Paragraph-names are procedure-names and therefore follow the rules for formation of procedure-names.

Other Names

The following name types take the same format used in the formation of data-names:

- FILE-NAMES
- CONDITION-NAMES
- RECORD-NAMES
- OVERFLOW-NAMES

QUALIFICATION OF NAMES

Every name used in a COBOL source program must be unique within the source program, either because no other name has the identical spelling, or because the name exists within a hierarchy of names (so that the name can be made unique by mentioning one or more of the higher levels of the hierarchy). The higher levels are called qualifiers when used in this way, and the process is called qualification.

The following rules apply to the qualification of names:

1. The word OF or IN must precede each qualifying name, and the names must appear in ascending order of hierarchy.
2. A qualifier must be of a higher level and within the same hierarchy as the name it is qualifying.
3. The same name must not appear at two levels in a hierarchy in such a manner that it would appear to qualify itself.
4. The highest level qualifier must be unique. Each qualifying name must be unique at its own level within the hierarchy of the immediately higher qualifier.
5. Qualification when not needed is permitted.
6. Qualifiers must not be subscripted, although the entire qualified name may be subscripted.
7. The total number of characters x cannot exceed 300 where:
$$x = T + 4N$$

T is the number of characters in all names, and
 N is the number of data names, including their qualifiers.
8. Regardless of qualification, procedure names and data names must not be the same.

COBOL PROGRAM SHEET

The purpose of the program sheet is to provide a standard way of writing COBOL source programs.

The Identification, Environment, Data, and Procedure Divisions which constitute a COBOL source program are written in the stated order. This

program sheet, despite its necessary restrictions, is of a relatively free form. The programmer should note, however, that the rules for using it are precise and must be followed exactly. These rules take precedence over any other rules, with respect to spacing.

SEQUENCE NUMBER: (COLUMNS 1-6)

The sequence number must consist only of digits; letters and special characters should not be used. The sequence number has no effect on the source program and need not be written. If the programmer supplies sequence numbers in each program card, the compiler will check the source program cards and will indicate any errors in their sequence. If these columns are blank, no sequence error will be indicated.

CONTINUATION INDICATOR: (COLUMN 7)

See Continuation of Non-Numeric Literals.

SOURCE PROGRAM STATEMENTS: (COLUMNS 8-72)

These columns are used for writing the COBOL source program.

PROGRAM IDENTIFICATION CODE: (COLUMNS 73-80)

These columns can be used to identify the program. Any character from the COBOL character set may be used, including the blank. The program identification code has no effect on the object program or the compiler.

MARGIN RESTRICTIONS

There are two margins on the COBOL program sheet: Margin A (columns 8-11), and Margin B (columns 12-72).

A division-name must begin in Margin A, and be followed by a space, the word DIVISION, and a period. This entry must appear on a line by itself.

A section-name must begin in Margin A, and be followed by a space, the word SECTION, and then a period. This entry must appear on a line by itself, except in declarations and the INCLUDE verb.

A paragraph-name must also begin in Margin A, and must be followed immediately by a period and a space. Statements within a paragraph may start on the same line as the paragraph-name. Succeeding lines of the paragraph must begin in Margin B.

When a statement spans more than one line, and column 72 of the line-to-be-continued is used, the continuation line may begin at the Margin B column (no space is required).

The FD level indicator in the Data Division, must begin at Margin A. Names and clauses within these entries must not begin before column 12. The level numbers (01-49, 77, 88) of data description entries may begin in Margin A; however, the names and/or clauses of this entry (data-names and/or clauses) must not begin before column 12.

CONTINUATION OF NON-NUMERIC LITERALS

When a non-numeric literal is of a length such that it cannot be contained on one line of a coding sheet, the following rules apply:

1. On every line containing a portion of a literal to be continued, the portion of the literal that is to be continued must not be terminated with a quotation mark.
2. On every line containing a portion of a literal being continued, the portion of the literal being continued must be immediately preceded by a quotation mark. This quotation mark may appear anywhere in Margin B, and may not be preceded by anything but spaces.
3. A hyphen must be punched in column 7 of each line in which the literal is being continued.

FORMAT NOTATION

Throughout this publication, basic formats are prescribed for various elements of COBOL. These generalized descriptions are intended to guide the programmer in writing his own statements. They are presented in a uniform system of notation, explained in the following paragraphs. This notation is useful in describing COBOL, although it is not part of COBOL.

1. All words printed entirely in capital letters are reserved words. These are words which have preassigned meanings in the COBOL language and are not to be used for any other purpose. In all formats, words written in capital letters selected for use must be duplicated.
2. All underlined reserved words are required unless the portion of the format containing them is itself optional. These are key words. If any such word is missing or is incorrectly spelled, it is considered an error in the program. Reserved words not underlined may be included or omitted at the option of the programmer. These words are used only for the sake of readability, and are called optional words.
3. All punctuation and special characters (except those symbols cited in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted, according to the rules for punctuation specified in this publication.
4. Lower-case words in formats represent information that must be supplied by the programmer. All lower-case words that appear in a format are defined in the accompanying text or in Appendix A.
5. In order to facilitate references to them in text, some lower-case words are followed by a hyphen and a digit, or letter. This modification does not change the syntactical definition of the word.
6. Certain hyphenated words in the formats consist of capitalized portions followed by lower-case portions. These designate clauses or statements that are described in other formats, in appropriate sections of the text.

7. Square brackets ([]) are used to indicate that the enclosed item may be used or omitted, depending on the requirements of the particular program. When two or more items are stacked within brackets, one or none of them may occur.
8. Braces ({ }) enclosing vertically stacked items indicate that one of the enclosed items is obligatory.
9. The ellipsis (...) indicates that the immediately preceding unit may occur once, or any number of times in succession. A unit means either a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If a term is enclosed in brackets or braces, the entire unit, of which it is a part, must be repeated when repetition is specified.
10. Comments, restrictions, and clarifications on the use and meaning of every format are contained in the appropriate portions of the text.

INPUT/OUTPUT PROCESSING

IBM System/360 Disk and Tape Operating Systems COBOL support various data organizations, record formats, and access methods. The facilities available to the COBOL user are specified in this section.

In this publication, the term IOCS (Input/Output Control System) can be considered equivalent to the term "Data Management Routines" used in other IBM System/360 Disk and Tape Operating Systems publications.

DATA ORGANIZATION

IBM System/360 Disk and Tape Operating Systems COBOL provide three types of data organization: standard sequential, indexed, and direct.

The number and type of control fields used to locate logical records in a file differ, depending on which of these three types of data organization is used. Consequently, each type of data organization is incompatible with the other two. For example, records created on a standard sequential file cannot also be read as an indexed file. That is, organization of an input file must be the same as the organization of the file at creation time.

Standard Sequential Data Organization

When standard sequential data organization is used, the logical records in a file are positioned sequentially in the order in which they are created, and are read sequentially in the order in which they were created (or in sequentially reversed order if the REVERSED option of the OPEN statement is written for tape files). This type of data organization must be used for tape or unit-record files and may be assigned to direct-access devices.

Indexed Data Organization

When indexed data organization is used, the position of each logical record in a file is determined by indexes maintained by the system and created with the file. The indexes are based on symbolic keys provided by the user. Indexed files must be assigned to direct-access devices.

Direct Data Organization

When direct data organization is used, the positioning of the logical records in a file is determined by keys supplied by the user. ACTUAL keys are used to specify the track. SYMBOLIC keys are used in conjunction with actual keys to identify a record on a track.

On each track, records are positioned in the order in which they are written. Direct files must be assigned to direct-access devices.

ACCESS METHODS

There are two access methods provided by System/360 COBOL:

SEQUENTIAL ACCESS: This is the method of reading and writing records of a file in a serial manner, the order of references is implicitly determined by the position of a record in the file, except when indexed data organization is specified.

RANDOM ACCESS: This is the method of reading and writing records of a file in a non-sequential manner; the control of successive references to the files is determined by specifically defined keys supplied by the user.

KEYS

When accessing indexed or direct files randomly, the user must provide information to identify the specific record desired. For both organizations, direct and indexed sequential, the user must provide a key to identify the desired record. These keys are defined as follows:

SYMBOLIC KEY: The SYMBOLIC KEY is a unique storage resident value that distinguishes a record from all other records in the file (for example, a stock-number in an inventory file or an employee's name or man-number in a payroll file).

RECORD KEY: The RECORD KEY is a unique value within the record that distinguishes it from all other records in the file.

ACTUAL KEY: The ACTUAL KEY is the location on the disk at which the record is located. Thus it is the actual track address.

These values are used by IOCS to determine where the record is located or where it should be placed. For a randomly accessed file, the values of the data-names for the symbolic and actual keys are never automatically modified by IOCS. The user has complete responsibility for ensuring that the correct values are in the data-names before reading, writing, or rewriting.

Depending on the type of random file organization, identification of a record is accomplished through the use of the SYMBOLIC, RECORD, or ACTUAL keys as follows:

ACCESSING A DIRECT FILE RANDOMLY

When accessing a direct file, the ACTUAL and SYMBOLIC KEYS are required.

IOCS uses the value of the ACTUAL KEY as the actual track address. After locating the track, for a read or rewrite operation, IOCS searches the track for a record that is preceded by a "key area" equal to the SYMBOLIC KEY. When a match is found, the data portion of the record is read or, if a rewrite operation, replaced by the new record. If, for a

read, the desired record cannot be found on the specified track, IOCS searches the entire cylinder for the record. When APPLY RESTRICTED SEARCH option is used, the search is limited to the specified track.

For a write operation, after locating the actual track, IOCS searches for the last record on the track, and writes the new record (with control fields including a key field equal to the SYMBOLIC KEY provided).

For a direct organization file, before a read, write, or rewrite, the track number must be moved into the data-name specified by the ACTUAL KEY clause, and the symbolic key must be moved into the SYMBOLIC KEY clause.

ACCESSING AN INDEXED FILE RANDOMLY

When accessing an indexed file, the RECORD and SYMBOLIC KEYS are required.

For blocked files, a "key area" precedes the block that IOCS uses to determine which record keys are in the block. For unblocked files, a "key area" precedes each record that IOCS uses to identify the RECORD KEY within the record.

When reading or writing records, the track containing the record desired is determined by using the SYMBOLIC KEY and searching the file's index table. When the track has been determined for unblocked records, the record is identified by comparing the SYMBOLIC KEY to the key area preceding the record. When the track has been determined for blocked records, the record is identified by comparing the SYMBOLIC KEY to the key area preceding the block, and then to the RECORD KEY of the record itself.

Before reading or rewriting an indexed file, the symbolic and record keys for the desired record must be moved into the data-names specified by the SYMBOLIC and RECORD KEY clauses.

ACCESSING AN INDEXED OR DIRECT FILE SEQUENTIALLY

When creating an indexed file sequentially, a RECORD KEY is required, and the SYMBOLIC KEY is optional.

When creating a direct file sequentially, the ACTUAL and SYMBOLIC KEYS are required.

Processing indexed or direct files sequentially is similar to that for a standard sequential file. Thus, IOCS determines where a record is to be found based solely upon the logical sequence in which records were placed in the file previously. For direct files, this logical sequence corresponds exactly to the physical sequence of the records; for indexed files, this logical sequence corresponds to the sequence of keys, which must be in collating sequence. If the user accesses an indexed file sequentially, and specifies binary zeros in the SYMBOLIC KEY, retrieval begins with the first record of the file.

It should be noted that the preceding discussion applies specifically to files accessed or created by a COBOL program. It is possible in lower level languages to create other types of files. In general, such files may not be used by COBOL programs.

CREATION OF AN INDEXED FILE

Indexed files may be created as follows:

1. Describe files with the following clauses:
 - ORGANIZATION IS INDEXED
 - [ACCESS IS SEQUENTIAL]
 - ASSIGN TO DIRECT ACCESS
 - [SYMBOLIC KEY IS]
 - RECORD KEY IS
2. Open the file-name as OUTPUT, and WRITE the records in ascending key sequence; close the file.
3. This file name may not be opened in any other OPEN statement in the program.

CREATION OF A DIRECT FILE

A direct file may be created sequentially by:

1. Describing files with the following clauses:
 - ORGANIZATION IS DIRECT
 - [ACCESS IS SEQUENTIAL]
 - ASSIGN TO DIRECT ACCESS
 - SYMBOLIC KEY IS
 - ACTUAL KEY IS
2. Opening a file as an output file
3. Writing each record sequentially, specifying its SYMBOLIC KEY

An end-of-file record is automatically placed on the last track of the file at CLOSE.

Figure 1 summarizes the clause and statement specifications allowed for each of the three data organizations. Also, each file-name must be specified in a SELECT clause in the Environment Division and must be defined by an FD entry in the File Section of the Data Division.

Figure 1. Permissible Data Organization Clauses and Statements

DISK AND TAPE OPER. SYSTEMS ORGANIZATION	DEVICE TYPE	ACCESS	OPEN	ORGANIZATION	RECORDING MODE	BLOCK CONTAINS	RESERVE ALTERNATE AREA	LABEL RECORDS	READ WRITE REWRITE	APPLY	KEY	CLOSE	ASSIGN
DTFCD	READER	{SEQUENTIAL}	INPUT	-	F	-	[1 NO]	OMITTED	READ {INTO} AT END	-	-	CLOSE	UNIT- RECORD
DTFCD	PUNCH	{SEQUENTIAL}	OUTPUT	-	F	-	[1 NO]	OMITTED	WRITE {FROM} {ADVANCING}	-	-	CLOSE	UNIT- RECORD
DTFPR	PRINTER	{SEQUENTIAL}	OUTPUT	-	F	-	[1 NO]	OMITTED	WRITE {FROM} {ADVANCING}	-	-	CLOSE	UNIT- RECORD
DTFMT	TAPE	{SEQUENTIAL}	{INPUT {NO-REWIND} {REVERSED} OUTPUT {NO-REWIND}}	-	[F U V]	{n (except for U)}	[1 NO]	{STANDARD OMITTED d - name}	READ AT END WRITE {FROM} {ADVANCING}	{WRITE ONLY} ²	-	{NO REWIND} {LOCK} {UNIT}	UTILITY
DTFSD	DISK	{SEQUENTIAL}	{INPUT OUTPUT I-O}	-	[F U V]	{n (not with U)}	[1 NO]	{STANDARD d - name}	READ AT END WRITE {FROM} REWRITE	{WRITE ONLY} ²	-	CLOSE {UNIT}	UTILITY DIRECT- ACCESS
DTFIS	DISK	{SEQUENTIAL}	{INPUT OUTPUT I-O}	INDEXED	F	{n}	{NO}	STANDARD	READ AT END REWRITE 1 INVALID KEY WRITE ²	-	{SYMBOLIC} RECORD	CLOSE	DIRECT- ACCESS
DTFDA	DISK	{SEQUENTIAL}	INPUT OUTPUT	DIRECT	{F U}	-	{NO}	{STANDARD d - name}	READ AT END WRITE	-	ACTUAL SYMBOLIC	CLOSE	DIRECT- ACCESS
DTFIS	DISK	RANDOM	{INPUT I-O}	INDEXED	F	{n}	{NO}	STANDARD	READ INVALID KEY WRITE {INVALID KEY} REWRITE {INVALID KEY}		SYMBOLIC RECORD	CLOSE	DIRECT- ACCESS
DTFDA	DISK	RANDOM	{INPUT I-O}	DIRECT	{F U}	-	{NO}	{STANDARD d - name}	READ INVALID KEY WRITE {INVALID KEY} REWRITE {INVALID KEY}	{RESTRICTED SEARCH}	SYMBOLIC ACTUAL	CLOSE	DIRECT- ACCESS

1 - For I-O Option of OPEN clause
2 - For OUTPUT Option

SECTION 3: IDENTIFICATION DIVISION

The Identification Division is used to identify a program and to provide other pertinent information concerning the program. The format of the Identification Division is:

IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

[AUTHOR. sentence...]

[INSTALLATION. sentence...]

[DATE-WRITTEN. sentence...]

[DATE-COMPILED. sentence...]

[SECURITY. sentence...]

[REMARKS. sentence...]

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	001	IDENTIFICATION DIVISION.
001	002	PROGRAM-ID. 'CALLPRGM'.
001	003	REMARKS. EXAMPLE OF A CALLING PROGRAM.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

Program-name is an external-name and must follow the rules for external name formation. Program-name identifies the object program to the Control Program.

IDENTIFICATION and the other COBOL words in the Identification Division must begin in Margin A. If sentences are written, they must be contained within Margin B. They may consist of any characters in the EBCDIC set.

GENERAL DESCRIPTION

The function of the Environment Division is to centralize the aspects of the total data processing problem that are dependent upon the physical characteristics of a specific computer. It provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

The Environment Division must begin in Margin A with the heading ENVIRONMENT DIVISION followed by a period.

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

The Configuration Section, which deals with the over-all specifications of computers, is divided into two paragraphs. They are: the Source-Computer paragraph, which defines the computer on which the COBOL compiler is to be run, and the Object-Computer paragraph, which defines the computer on which the program produced by the COBOL compiler is to be run.

The Input-Output Section deals with the definition of the external media (input/output devices) and information needed to create the most efficient transmission and handling of data between the media and the object program. This section is divided into two paragraphs. They are the File-Control paragraph, which names and associates the files with the external media; and the I-O-Control paragraph, which defines special input-output techniques.

CONFIGURATION SECTION

The format of the Configuration Section is:

```
[  
CONFIGURATION SECTION.  
[SOURCE-COMPUTER.  IBM-360 [model-number].]  
[OBJECT-COMPUTER.  IBM-360 [model-number].]  
]
```

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7	8 12
001	004	ENVIRONMENT DIVISION.
001	005	CONFIGURATION SECTION.
001	006	SOURCE-COMPUTER. IBM-360 D30.
001	007	OBJECT-COMPUTER. IBM-360 D30.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

INPUT-OUTPUT SECTION

The format of the Input-Output Section is:

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  [SELECT file-name ASSIGN-clause  
  [RESERVE-clause]...  
  [ACCESS-clause]  
  [ORGANIZATION-clause]  
  [SYMBOLIC KEY-clause]  
  [ACTUAL KEY-clause]  
  [RECORD KEY-clause].]  
I-O-CONTROL.  
  [SAME-clause.] ...  
  [RERUN-clause.] ...  
  [APPLY-clause.] ...
```


An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	004	ENVIRONMENT DIVISION. . .
001	008	INPUT-OUTPUT SECTION.
001	009	FILE-CONTROL.
001	010	SELECT FILEA ASSIGN TO 'SYS004' UTILITY 2400 UNITS.
001	011	SELECT FILEB ASSIGN TO 'SYS005' UNIT-RECORD 2540R RESERVE NO ALTERNATE AREA.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

The individual optional clauses that compose the File-Control and I-O-Control paragraphs may appear in any order within their respective sentences or paragraphs. They are described in the following text. The Input-Output Section may be omitted if there are no files used in the program.

I-O-CONTROL may be omitted if none of the clauses in the paragraph are written. A period must follow the last clause in each SELECT sentence written in the File-Control paragraph, and must follow each clause written in the I-O-Control paragraph.

FILE-CONTROL PARAGRAPH

SELECT Sentence

The SELECT sentence must begin with the words SELECT file-name and must be given for each file named in the File-Control paragraph.

The name of each file must be unique within a program and must have a File Description (FD) in the Data Division of the source program. Conversely, every file named in an FD entry must be named in a SELECT sentence.

ASSIGN Clause

The format of the ASSIGN clause is:

ASSIGN TO external-name { DIRECT-ACCESS
UTILITY
UNIT-RECORD } device-number UNIT [S]

The ASSIGN clause is used to assign a file to a particular device.

External-name specifies the name by which the file is known to the Control Program.

External-name for files in the assign clause must be of the format 'SYSnnn' where nnn is a 3-digit number between 001 and 254.

DIRECT-ACCESS, UNIT-RECORD, and UTILITY specify device classes. Each file must be assigned to a device class. Files assigned to UTILITY or UNIT-RECORD have sequential access only, and data contained on these files is organized in the standard sequential fashion.

Files assigned to DIRECT-ACCESS may have standard sequential, indexed, or direct organization. When organization is indexed or direct, access may be either sequential or random.

Device-number is used to specify a particular device type within a device class and is required.

The allowable device-numbers are:

UNIT-RECORD
1442R, 1442P, 1403, 1404 (continuous forms only), 1443, 1445, 2501,
2520R, 2520P, 2540R, 2540P

"R" indicates reader.

"P" indicates punch.

UTILITY
2400, 2311, 2321

DIRECT ACCESS
2311, 2321

ACCESS Clause

The format of the ACCESS clause is:

[ACCESS is { SEQUENTIAL
RANDOM }]

The ACCESS clause indicates the manner in which the records of a file are read or written.

If this clause is not written, ACCESS IS SEQUENTIAL is assumed. If ACCESS IS RANDOM is written, the file must be assigned to a DIRECT-ACCESS device and must be indexed or direct.

Ext | ORGANIZATION Clause

The format of the ORGANIZATION clause is:

[ORGANIZATION IS { INDEXED
DIRECT }]

This clause may only be written for files assigned to direct-access devices in a SELECT sentence.

If the ORGANIZATION clause is omitted, a standard sequential file is assumed.

INDEXED specifies indexed data organization.

DIRECT specifies direct data organization.

RESERVE Clause

The format of the RESERVE clause is:

[RESERVE { ^{NO}
1 } ALTERNATE AREA[S]]

This clause specifies the number of buffers reserved for a sequential file in addition to the standard minimum of one required for a file. If this clause is omitted, one additional buffer is assumed. If NO is written, no additional buffer will be reserved.

SYMBOLIC KEY Clause

The format of the SYMBOLIC KEY clause is:

[SYMBOLIC KEY IS data-name]

The symbolic key identifies a record.

This clause is allowed only when the ORGANIZATION clause is specified, and is required if ACCESS IS RANDOM is specified.

Data-name must not be defined in the file for which it is the symbolic key.

If the SYMBOLIC KEY clause is used for an ACCESS SEQUENTIAL file having an ORGANIZATION IS DIRECT clause, the symbolic identity of the record will be placed into data-name whenever a READ statement is executed for the file. Any changes the programmer may make to data-name will not affect the order in which records are read from the file.

If the file is specified as ACCESS IS RANDOM, the symbolic identity of the desired record to be read or written must be placed in data-name before the READ or WRITE statement for the record is executed. The symbolic identity will be used by IOCS to determine the physical location of the record.

Data-name may be any fixed length working storage item less than 256 bytes in length, with the exception of floating-point or report items. A discussion of data items is contained in Section 5.

ACTUAL KEY Clause

The format of the ACTUAL KEY clause is:

[ACTUAL KEY IS data-name]

The actual key specifies the track address at which the record is to be placed, or at which the search for the record is to start.

This clause is required for a file when ORGANIZATION IS DIRECT is specified for it. This clause must not be specified for a file under any other circumstances.

The functions of this clause are similar to those of the SYMBOLIC KEY clause, except that this clause specifies a data item that will contain the track address on which a record is to be found or placed.

Data-name must be defined as an 8-byte data item.

The actual key in operating System/360 specifies a relative track address. In Disk and Tape Operating Systems it specifies the actual track or hardware address. The actual key field must be 8 bytes long and contain the track address as specified in the Data Management Concepts publication for the system.

Ext | RECORD KEY Clause

The format of the RECORD KEY clause is:

[RECORD KEY IS data-name]

This clause is used with files whose organization is indexed. Data-name specifies the item within the data record that contains the key for the record.

The item specified by data-name must be defined to exclude the first byte of the record in the following types of files:

1. Files whose records are unblocked
2. Files from which records are to be deleted
3. Files any one of whose keys might start with a delete-code character (high-value).

With these exceptions, the item specified by data-name may appear anywhere within the record.

When more than one record description is associated with a file, a similar field must appear in each description, and must be in the same relative position from the beginning of the record although the same name need not be used.

| Data-name may be any fixed-length item less than 256 bytes in length, with the exception of floating-point or report items.

I-O CONTROL PARAGRAPH

SAME Clause

The format of the SAME clause is:

[SAME AREA FOR file-name-1 file-name-2 [file-name-3...].]

The SAME clause is used to specify that two or more files are to use the same main storage area for buffers.

Only one of the files named in this clause may be open at any time.

More than one SAME clause may appear in a COBOL program, but any one file-name may appear in only one SAME clause.

* RERUN Clause

The format of the RERUN clause is:

[RERUN ON external-name EVERY END OF $\left. \begin{array}{c} \text{REEL} \\ \text{UNIT} \end{array} \right\}$ of file-name.]

This clause specifies that checkpoint records are to be written on the unit specified by an external name. Only one checkpoint device is permitted in any given program.

A checkpoint record is a recording of the status of the computer at a given point in the execution of the object program. It contains all of the information necessary to restart the program from that point.

Checkpoint records will be written whenever a change of volume occurs for the file named by file-name. File-name must be the name of a standard sequential file. The term volume is defined in the Supervisor and Input/Output Macros publication for the system.

Format of external-name is the same as that in the ASSIGN clause.

APPLY Clause

The formats of the APPLY clause are:

Option 1

[APPLY overflow-name TO FORM-OVERFLOW ON file-name.]

This option is used to specify overflow-name, which may be used in tests for form-overflow of a printer to which the file named by file-name is assigned. The condition is true when channel 12 is sensed by an on line printer.

Overflow-names follow the rules for data-name formation. Data name formation is discussed in Section 1. Data names are discussed in Section 5; overflow tests are discussed in Section 6.

An overflow-name may be written in conjunction with a WRITE statement with an ADVANCING option in order to control spacing of printed records. Thus, the following statement could be written (with a programmer-supplied overflow-name):

```
IF overflow-name WRITE X AFTER  
ADVANCING 0 LINES ELSE WRITE X  
AFTER ADVANCING 2 LINES
```

** Option 2

[APPLY WRITE-ONLY ON file-name...]

This option may be used for blocked V type records (which are only permitted on standard sequential files). Records must be built in a work area, and written with a WRITE...FROM clause.

The only reference to the record may be in a WRITE...FROM clause. This clause permits records to be added to a buffer even though the maximum size record cannot fit.

Subfields of these records may never be referenced.

Option 3

[APPLY RESTRICTED SEARCH OF integer TRACKS ON file-name...]

Integer can only be 1.

This clause is used to control the extent of the search made for a specified record. This option may only refer to files specified as ACCESS IS RANDOM and ORGANIZATION IS DIRECT. In normal operation, execution of a READ statement for a file causes the entire cylinder to be searched for the specified record when the record cannot be found on the specified track. When RESTRICTED SEARCH is written, the search is limited to the specified track. If the desired record cannot be found in the case of a READ or REWRITE, the INVALID KEY option of the READ, or REWRITE statement will be executed.

The INVALID KEY option will also be executed when the specified track is outside the limits of the last cylinder containing the file. It is the programmer's responsibility to determine which condition produced the invalid key condition.

GENERAL DESCRIPTION

The Data Division of a COBOL source program describes the information to be processed by the object program. This information falls into the following categories:

1. Data contained in files, entering or leaving the internal storage of the computer.
2. Data developed internally and placed in intermediate or working storage, and constant data defined by the user.
3. Linkage data descriptions for communication between main program and subprograms.

The Data Division must begin in Margin A with the header DATA DIVISION followed by a period. Each of the sections of the Data Division begins with a fixed section-name, and is followed by the word SECTION and a period, as follows:

DATA DIVISION.

FILE SECTION.

File Description entries

Record Description entries

WORKING-STORAGE SECTION.

Record Description entries

LINKAGE SECTION.

Record Description entries

The sections must appear in this order. If any section is not required, both it and its section-name may be omitted.

ORGANIZATION OF THE DATA DIVISION

The Data Division is subdivided into sections, according to types of data. Each section consists of entries, rather than sentences and paragraphs. An entry consists of a level indicator, a data-name or file-name, and a series of clauses which may be separated by commas or semicolons. The clauses may be written in any sequence (except the REDEFINES clause). Each entry must terminate with a period and a space.

FILE SECTION

The File Section describes the content and organization of files. Each such entry is followed by related Record Description entries.

The Record Description entries used in conjunction with a File describe the individual items contained in a data record of a file.

WORKING STORAGE SECTION

The Working-Storage Section consists solely of Record Description entries. These entries describe the areas of storage where intermediate results are stored at object-program execution time, and constants along with their values.

Ext LINKAGE SECTION

The Linkage Section is a required part of any COBOL subprogram that contains an ENTRY statement with USING option, and serves as a data-linking mechanism between the main program and the subprogram. It consists only of Record Description entries that provide dummy names for linkage to data in the main program. This is the only Data Division section whose entries do not cause object program data storage areas to be allocated.

When passing computational, computational-1 or computational-2 fields, refer to Appendix C for a discussion on alignment requirements.

CONCEPTS OF DATA DESCRIPTION

The following material defines the basic terms and concepts used in describing data. Rules which govern the writing of data descriptions appear later in this section.

LEVELS OF DATA ITEMS

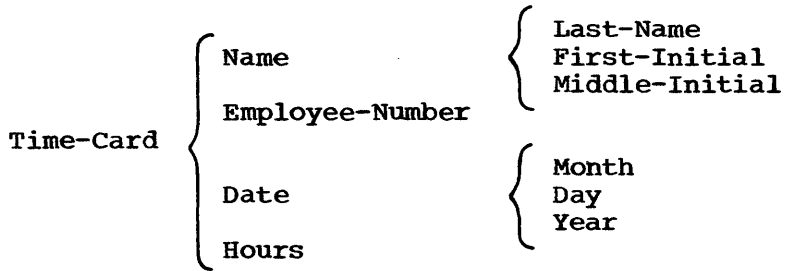
Level indicators are used to show how data items are related to each other. The most inclusive grouping of data is the file. The level indicator for a file is FD.

For purposes of processing, the contents of a file are divided into logical records, with level number 01 specifying a logical record. The object program locations of all logical records are assumed adjusted to double-word boundary. Subordinate data items that constitute a logical record are grouped in a hierarchy, and identified with level numbers 02 to 49.

Level number 77 identifies a record description entry in the Linkage Section or the Working-Storage Section. The level number 77 cannot appear in the file section.

Level number 88 is used to define a condition-name for a related conditional variable. A level number less than 10 may be written as a single digit preceded by a blank.

Levels, allowing specification of subdivisions of a record are necessary for referring to data. Once a subdivision is specified, it may be subdivided further to permit more detailed data reference. This may be illustrated by the following weekly time-card record, which is divided into four major items: name, employee-number, date, and hours, with more specific information appearing for name, and date.



Subdivisions of a record, that are not themselves further subdivided, are called elementary items. Data items that contain subdivisions are known as group items. When a Procedure Division statement makes reference to a group item, the reference applies to the area reserved for the entire group. Less inclusive groups are assigned higher level numbers. Level numbers of items within groups need not be consecutive. A group includes all groups and elementary items described under it until a level number less than or equal to the level number of the group is encountered. Separate entries are written in the source program for each level. To illustrate level numbers and group items, the weekly time-card record in the previous example may be described by Data Division entries having the following level numbers and data-names described in Figure 2.

```

01 TIME-CARD
04 NAME
06 LAST-NAME
06 FIRST-INITIAL
06 MIDDLE-INITIAL
04 EMPLOYEE-NUMBER
04 DATE
05 MONTH
05 DAY
05 YEAR
04 HOURS

```

Figure 2. Example of Data Levels

Only the level number and data-name of each entry have been given in Figure 2, data defining clauses were omitted..

Throughout the Data Division, level 01 items are adjusted to a double-word boundary; level 77 binary or internal floating-point items are adjusted to the next available half-word, full-word, or double-word boundary, as appropriate. For blocked files refer to the discussion on Intra-Record Slack bytes in Appendix C.

CONDITION-NAMES

The general form of a condition-name entry is:

88 condition-name VALUE IS literal.

Each level 88 entry must be preceded by either another level 88 entry (in the case of several consecutive condition-names pertaining to an elementary item), or by an elementary item.

Every condition-name pertains to an elementary item in such a way that the condition-name may be qualified by the name of the elementary item and the elementary item's qualifiers. A condition-name is used in the Procedure Division in place of a simple relational condition.

A condition-name may pertain to an elementary item (a conditional variable) requiring subscripts. In this case the condition-name, when written in the Procedure Division, must be subscripted according to the same requirements of the associated elementary item. Subscripting is discussed later in this text.

The literal in a condition-name entry must be consistent with the data type of the conditional variable.

Figure 3 is an example of Data Division entries and a Procedure Division statement that might be written using level 88 and the condition-name-test. (Details on the condition-name-test appear in Section 6 under the subsection Test Conditions.)

Data Division Portion:

```
01 TIME-CARD.
02 NAME, PICTURE X(20).
02 PAY-CODE, PICTURE 9.
    88 MONTHLY, VALUE IS 1.
    88 HOURLY, VALUE IS 2.
    88 SUBCONTRACTOR, VALUE 3.
02 SALARY, PICTURE 9999.
02 RATE-PER-HOUR, REDEFINES SALARY PICTURE 9V999,
   DISPLAY.
02 PER-DIEM, REDEFINES RATE-PER-HOUR PICTURE 99V99,
   DISPLAY.
```

Procedure Division Portion:

```
IF HOURLY COMPUTE GROSS = 40 * RATE-PER-HOUR,
ELSE IF MONTHLY COMPUTE GROSS = SALARY / 4.334,
ELSE IF SUBCONTRACTOR COMPUTE GROSS = 5 * PER-DIEM,
ELSE PERFORM ERROR-PROCESS.
```

Figure 3. Condition-name Example

DATA-NAMES

Data-names are names assigned by the programmer to identify data items used in a program. They always refer to a kind of data, not to a particular value, and the items they refer to usually assume a number of values during the course of a program.

A data-name or the key word FILLER must be the first word following the level number in each Record Description entry, as shown in the following general format:

level-number { data-name
 FILLER }

This data-name is the defining name of the entry, and is the means by which references to the associated data area (containing the value of a data item) are made.

If some of the characters in a record are not used in the processing steps of a program, then the data description of these characters need not include a data-name. In this case, FILLER is written in lieu of a data-name after the level number.

If the same data-name is assigned to more than one item in a program, it must be qualified in all references to it in the Procedure Division, Data Division, or Environment Division, except in the REDEFINES clause.

A data-name is qualified by writing either IN or OF after it, followed by the name of one or more groups, or the record or file in which it is contained. A highest level qualifier must, however, be unique.

Any combination of qualifiers that will ensure uniqueness may be used. More qualifiers may be used than are absolutely needed. In Figure 2, if YEAR OF DATE is needed to make YEAR unique, YEAR OF DATE IN TIME-CARD is also permitted.

A data-name cannot be subscripted when it is used as a qualifier. However, the entire qualified data-name may be subscripted.

LITERALS

A literal is a constant that is not identified by a data-name in a program, but is completely defined by its own identity. A literal is either non-numeric (alphabetic or alphanumeric), numeric, or floating-point.

Non-Numeric Literals

A non-numeric literal must be bounded by quotation marks and may consist of any combination of characters in the IBM EBCDIC set, except quotation marks. All spaces enclosed by the quotation marks are included as part of the literal. A non-numeric literal may not exceed 120 characters in length.

The following are examples of non-numeric literals:

'EXAMINE CLOCK NUMBER'
'12565'
'PAGE 144 MISSING'

Numeric Literals

A numeric literal must contain at least one and not more than 18 digits. A numeric literal may consist of the characters 0 through 9, the plus sign or the minus sign, and the decimal point. It may contain only one sign character and only one decimal point. The sign, if present, must appear as the leftmost character in the numeric literal. If a numeric literal is unsigned, it is assumed to be positive.

A decimal point may appear anywhere within the numeric literal, except as the rightmost character. If a numeric literal does not contain a decimal point, it is considered to be a whole number.

The following are examples of numeric literals:

1506798
+12572.6
-256.75
.16

Ext Floating-Point Literals

A floating-point literal must have the form:

$$\left[\begin{array}{c} + \\ - \end{array} \right] \text{mantissa } E \left[\begin{array}{c} + \\ - \end{array} \right] \text{exponent}$$

The plus or minus signs preceding the mantissa and exponent are the only optional characters within the format. The mantissa consists of one to 16 digits with a required decimal point.

Immediately to the right of the mantissa, the exponent is represented by the symbol E, followed by a plus or minus sign (if a sign is given), and one or two digits. The magnitude of the number represented by a floating-point literal must not exceed $.72 \times (10^{76})$. A zero exponent must be written as 0 or 00.

The value of the literal is the product of the mantissa and ten raised to the power given by the exponent. A floating-point literal must appear as a continuous string of characters with no intervening spaces.

The following are examples of floating-point literals:

12.3E2
-.34566E+17
+2.56E-6

FIGURATIVE CONSTANTS

Figurative constants are a special type of literal. They are values that have been assigned fixed data-names. Figurative constants must not be bounded by quotation marks.

ZERO may be used in many places in a program as a numeric literal. It may not, however, be used in an arithmetic statement. The use of ZERO as a non-numeric literal is permitted. All other figurative constants are considered non-numeric. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

ZERO	Represents one or more zeros.
ZEROS	
ZEROES	
SPACE	Represents one or more blanks or spaces.
SPACES	
HIGH-VALUE	Represents one or more appearances of
HIGH-VALUES	the highest value in the computer's collating sequence. (Hexadecimal 'FF')

LOW-VALUE LOW-VALUES	Represents one or more appearances of the lowest value in the computer's collating sequence. (Hexadecimal '00')
ALL 'character'	Represents one or more occurrences of the single character bounded by quotation marks. <u>Character</u> may not be a quotation mark.
QUOTE QUOTES	Represents the character '. Note that the use of the word QUOTE to represent the character ' at object time is not equivalent to the use of the symbol ' to bound a non-numeric literal.

When a figurative constant is used in such a way that the exact number of characters required cannot be determined, only one character is generated. For example, the statement DISPLAY ZEROES would produce one zero character since, in this case, the length of the sequence of zeros to be displayed cannot be determined.

TYPES OF DATA ITEMS

Several types of data items can be described in a COBOL source program. These data items are described in the following text. The format of the Record Description entry used to describe each of these items appears under the discussion of Record Description entries.

Group Items

A group item is defined as one having further subdivisions, so that it contains one or more elementary items. In addition, a group item may contain other groups. An item is a group item if, and only if, its level number is less than the level number of the immediately succeeding item, unless the latter level is 88. If an item is not a group item, then it is an elementary item, or, in the case of level 88, it is a condition-name.

The maximum length for any group item or elementary item in a tape system is 4092 bytes, whereas for a disk file it is 3625 bytes, except for a fixed-length Working-Storage group item, which may be as long as 32,767 bytes.

Elementary Items

Elementary items are items not further subdivided.

Alphabetic Item

An alphabetic item may contain any combination of the characters A through Z and space. Each alphabetic character is stored in a separate byte.

Alphanumeric Item

An alphanumeric item consists of any combination of characters in the IBM EBCDIC set. Each alphanumeric character is stored in a separate byte.

Report Item

A report item is an alphanumeric item containing only digits and/or special editing characters. It must not exceed 127 characters in length. A report item can be used only as a receiving field for numeric data. Each report character is stored in a separate byte. (See PICTURE and BLANK WHEN ZERO clauses.)

Fixed-Point Items

Fixed-point items may be defined as external decimal, internal decimal, or binary. External decimal corresponds to the form in which information is represented initially for card input, or finally for printed or punched output. Such items may be converted (by moving) to the internal machine formats described as internal decimal or binary. Except when an item is a single digit in length, these formats require less storage than the external decimal format and can be used to save space on volumes. The binary mode of representation is particularly efficient for data-names used as subscripts. Computational results are the same, regardless of the particular format selected provided the intermediate computational results do not require more than 18 digit positions.

EXTERNAL DECIMAL ITEM: Decimal numbers in the System/360 zoned format are external decimal items. Each digit of a number is represented by a single byte, with the four low-order bits of each eight-bit byte containing the value of a digit. The four high-order bits of each byte are zone bits; the zone bits of the low-order byte represent the sign of the item. The maximum length of an external decimal item is 18 digits. For items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration interpreted as positive but which does not represent an overpunch.

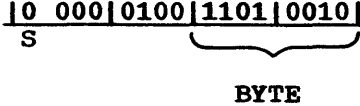
The codes used in the illustrations are as follows:

Z = Zone
Hexidecimal F = non-printing plus sign(1111)
9 = A digit
S = The sign position of a numeric field
b = A blank

If the item is used as an operand, it is assumed that the area contains a number less than or equal to that specified in the PICTURE clause.

An Example of binary represented decimal item is shown in Figure 6.

If the number is +1234 and,

<u>Picture and Usage Are:</u>	<u>Machine Representation Is:</u>
PICTURE S9999 COMPUTATIONAL.	0 000 0100 1101 0010 S
	

S=Sign a "1" in position S means number is negative.

 a "0" in position S means number is positive.

Figure 6. Example of Binary Represented Decimal Item

Ext Floating Point Items

External and internal floating-point formats define data items whose potential range of value is too great for fixed-point representation. The magnitude of the number represented by a floating-point item must not exceed .72 * (10 ** 76).

Ext EXTERNAL FLOATING-POINT ITEM: An external floating-point item consists of a combination of the characters +, -, blank, decimal point, and digits 0 through 9, appearing in a specific format which represents a number in the form of a decimal number followed by an exponent. The exponent specifies a power of ten that is used as a multiplier. External floating-point items (also called scientific decimal items) are scanned at object time for conversion to the equivalent internal floating-point value, when used as numeric operands. (See fp-form of PICTURE clause.) Each character of the PICTURE, except V, represents a single byte of storage reserved for the item.

An Example of an External Floating-Point item (literal) is shown in Figure 7.

If number is +12.34E+2 and,

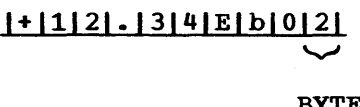
<u>Picture and Usage Are:</u>	<u>Machine Representation Is:</u>
PICTURE +99.99E-99 DISPLAY.	+ 1 2 . 3 4 E b 0 2 

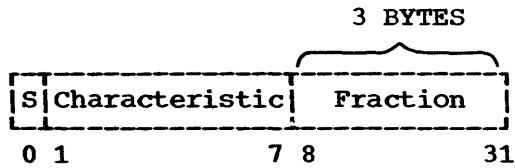
Figure 7. Example of External Floating-Point Item

Ext INTERNAL FLOATING-POINT ITEM: An internal floating-point item may be considered equivalent to an external floating-point item in capability and purpose. Internal floating-point numbers occupy four or eight bytes, depending on the length of the fractions.

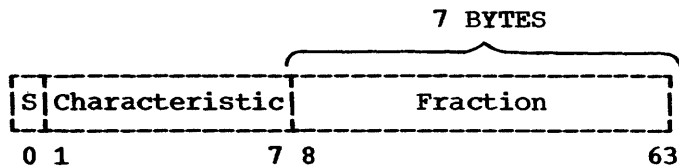
In the short-precision format, the fraction appears in the rightmost three bytes; in the long-precision format, the fraction appears in the rightmost seven bytes. The sign of the fraction is the leftmost bit in either format, and the exponent appears in bit positions 1 through 7.

Examples of internal floating-point items are shown in Figure 8.

For COMPUTATIONAL-1 (short form), the machine representation is:



For COMPUTATIONAL-2 (long form), the machine representation is:



For a discussion of internal representation of floating-point numbers, refer to IBM System/360 Principles of Operation Form A22-6821-1.

Figure 8. Examples of Internal Floating-Point Item

ALIGNMENT OF DATA FIELDS

The compiler assigns storage so that the starting byte of a binary or internal floating-point item is on the next available half-word, full-word, or double-word boundary, as appropriate. In this way, an internal floating-point or binary item is properly aligned at the storage location required by the computer.

If a data hierarchy contains binary or floating point items intermixed with other elementary items, there may exist "slack bytes" introduced to assure the necessary byte alignment (implicit synchronization).

Slack bytes exist in a record not only in main storage but on files. The compiler inserts slack bytes on output and expects them on input.

A further discussion of slack bytes is contained in Appendix C.

FILE SECTION

The File Section is used to define data that is contained in files.

FILE AND RECORD HANDLING

For purposes of processing, the contents of a file are divided into logical records. It is important to note that several logical records may occupy a block (i.e., a physical record).

In COBOL there are two classes of files;

1. A file for which there is only one 01-level record description subordinate to the FD entry, called a single-record file.
2. A file for which there is more than one 01-level record description subordinate to the FD entry, called a multiple-record file.

There are also two classes of records that may be contained in a file, fixed-length records and variable-length records. Variable-length records contain an OCCURS clause with a DEPENDING ON option; fixed-length records do not.

A SINGLE-RECORD file may contain either:

1. Fixed -length records, or
2. Variable-length records.

A MULTIPLE-RECORD file may contain records with three different characteristics:

1. EQUAL - Each record described is fixed in length and all the lengths are equal.
2. DIFFERING - Each record described is fixed in length, but at least two record descriptions have different lengths.
3. VARIABLE - One or more of the records described is variable in length.

Record Types

Three record types are available to COBOL users:

1. Fixed type (Format F): Can be used only when all logical records in a file are the same length; may be blocked, and may use the first character to control the printer carriage or punch stacker selection.
2. Variable type (Format V): Can be used for fixed, differing or variable records; records are preceded by a 4-byte count control field; may use the first character to control the printer carriage or punch stacker selection; may be blocked; may contain differing or variable length records. The control field is handled by COBOL and Data Management automatically.

The control field is not available to the user.

Figure 9 is a typical example of a control field used in format V records.

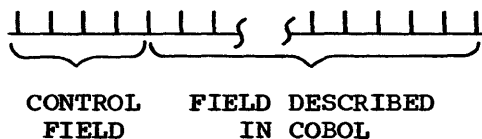


Figure 9. Typical Example of Control Field Used in Format V Records

3. Unspecified type (Format U), can be used for unblocked records.

If no recording mode is specified, records processed by COBOL object programs are format V records. If F or U recording mode is written, then either Format F or Format U records are assumed by the compiler.

FILE SECTION ENTRIES

FD file-name

[BLOCK CONTAINS integer {CHARACTERS
RECORDS }]

[RECORDING MODE IS model]

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

LABEL {RECORD IS
RECORDS ARE } {STANDARD
OMITTED
data-name }

DATA {RECORD IS
RECORDS ARE } record-name....

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	012	DATA DIVISION.
001	013	FILE SECTION.
001	014	FD FILEA, DATA RECORD IS RECORD-1, LABEL RECORDS
001	015	ARE STANDARD, BLOCK CONTAINS 5 RECORDS, RECORDING
		MODE IS F.
001	016	01 .
		.
		.
001	018	FD FILEB DATA RECORD IS RECORD-2, LABEL RECORDS
		ARE OMITTED.
001	019	01

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

Notes

1. The FD entry must describe each data file to be processed by the object program.
2. File-name is the highest level qualifier for its record description entries.

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause specifies the number of logical records of maximum length or the maximum number of characters (bytes) in a physical record. The format for this clause is:

BLOCK CONTAINS integer $\left\{ \begin{array}{l} \text{RECORDS} \\ \text{CHARACTERS} \end{array} \right\}$

The BLOCK CONTAINS clause must not be written if the UNIT-RECORD clause is specified in the Environment Division, or if U type records are used.

If CHARACTERS is written, integer must include the number of bytes occupied by slack bytes or control words contained in the physical records.

If this clause is omitted, it is assumed that records are not blocked.

RECORD CONTAINS Clause

The RECORD CONTAINS clause is used to specify the maximum size of logical records. The format for this clause is:

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

Integer-1 and integer-2 are used to specify minimum and maximum record sizes respectively. If the file contains only fixed-length records, integer-1 (if specified) and integer-2 must be equal to the sizes of the smallest and largest records described for the file respectively. If the file contains variable-length records, integer-1 is ignored and integer-2 is assumed to be the maximum size that any record in the file will have. Record lengths are determined by the compiler regardless of whether or not the clause is specified.

The RECORD CONTAINS clause is not necessary for a file having equal length records.

LABEL RECORDS Clause

This clause specifies the presence of standard or non-standard labels on a file, or the absence of labels. The format of this clause is:

LABEL RECORDS ARE $\left\{ \begin{array}{l} \text{STANDARD} \\ \text{OMITTED} \\ \text{data-name} \end{array} \right\}$

No labels	OMITTED
Non-standard labels	OMITTED
Standard labels	STANDARD
Standard and user labels	data-name

The OMITTED option must be specified for a file assigned to unit record devices. It may be specified for files assigned to magnetic tape units. Use of the OMITTED option does not result in automatic bypassing of non-standard labels on input. It is the user's responsibility to

either process or bypass non-standard labels on input, and create them on output.

The STANDARD option must be specified for files with indexed organization. It may be specified for any files except as noted in the preceding text.

The data-name option may be specified for files with standard sequential organization, with the exception of unit-record files, or for files with direct organization. The use of this option indicates that, in addition to standard labels, user labels are to be processed (see Options 1 and 2 of the Procedure Division USE section). Data-name, in this option, is a 01 or 77 level data-name in the Linkage Section of the Data Division which describes the label. This data-name is then available for reference by a declarative procedure written by the user for label processing. Label processing declarative procedures are discussed in Section 6. Data-name may not be subscripted.

A user label is 80 characters in length. A user header label is characterized by the appearance of UHL in character positions 1 through 3; a user trailer label has UTL in character positions 1 through 3. For both types, the relative position (1 through 8) of the label within the user label group is in character position 4. The remaining 76 characters are formatted according to user choice.

DATA RECORDS Clause

This clause specifies the names of the logical records in a file.

Its format is:

DATA { RECORD IS
RECORDS ARE } record-name...

Record-name is a data-name described with a 01 level-number in this section.

RECORDING MODE Clause

The RECORDING MODE clause specifies the format of the logical records comprising the file. The format for this is:

[RECORDING MODE IS mode]

Mode may be specified as either U,F or V.

The F mode (fixed-length format) may be specified when all the logical records in a file are the same length. This implies that no OCCURS DEPENDING ON clauses are associated with any entries in the data record descriptions. If more than one data record description is given allowing the FD entry, all record length calculated from the data record descriptions must be equal.

All UNIT-RECORD files must be F mode (fixed format).

The V mode (variable-length format) may be specified for any combination of record descriptions. A logical record of this format has a control field preceding it containing the length of the logical record.

The U mode (unspecified format) may be used with any combination of record descriptions. It may be compared to V mode records which are not blocked and without the preceding count control field.

The RECORDING MODE clause must be specified for files with F or U type records. If this clause is omitted, V type records are assumed.

RECORD DESCRIPTION ENTRY

A Record Description entry specifies the characteristics of each item in a data record. Every item must be described in a separate entry in the same order in which the item appears in the record. Each Record Description entry consists of a level-number, a data-name, and a series of independent clauses followed by a period.

The general format of a Record Description entry is:

```

level-number { data-name
              FILLER } [REDEFINES-clause]
              [PICTURE-clause] [BLANK-clause]
              [OCCURS-clause] [VALUE-clause]
              [JUSTIFIED RIGHT]
              [USAGE-clause].
  
```

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	012	DATA DIVISION. . .
001	015	01 RECORD-1.
001	016	02 SUB-FIELDA PICTURE IS X(68).
001	017	02 SUB-FIELDB PICTURE IS X(12). . .
001	019	01 RECORD-2 PICTURE IS X(80).

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

When this format is applied to the various specific items of data, it is limited by the nature of the data being described. The allowable format for the description of each data type appears below. Clauses

which are not shown in a format are specifically forbidden in that format. Clauses that are mandatory in the description of certain data items are written without brackets.

GROUP ITEM

Format:

level-number { data-name
 FILLER } [REDEFINES-clause]
 [OCCURS-clause] [USAGE-clause]

Example:

01 GROUP-NAME.
 02 FIELD-B PICTURE X.
 02 FIELD-C PICTURE 9.

ELEMENTARY ITEMS

Alphabetic Item

Format:

level-number { data-name
 FILLER } [REDEFINES-clause] [OCCURS-clause]
 PICTURE IS alpha-form [USAGE IS DISPLAY]
 [VALUE IS alphabetic-literal] [JUSTIFIED RIGHT].

Example:

02 EMPLOYEE-NAME PICTURE A(20).

Alphanumeric Item

Format:

level-number { data-name
 FILLER } [REDEFINES-clause] [OCCURS-clause]
 PICTURE IS an-form [USAGE IS DISPLAY]
 [VALUE IS non-numeric-literal] [JUSTIFIED RIGHT].

Examples:

02 MISC-1 PICTURE X(53).
02 MISC-2 PICTURE XXXXXXXX.

Report Item

Format:

level-number { data-name
 FILLER } [REDEFINES-clause] [OCCURS-clause]
PICTURE IS { numeric-form BLANK WHEN ZERO
 report-form [BLANK WHEN ZERO] }
[USAGE IS DISPLAY].

Example:

02 TOTAL PICTURE \$999,999.99-.
02 RLT PICTURE 999 BLANK ZERO.

External Decimal Item

Format:

level-number { data-name
 FILLER } [REDEFINES-clause] [OCCURS-clause]
[USAGE IS DISPLAY] PICTURE IS numeric-form
[VALUE IS numeric-literal].

Example:

02 HOURS-WORKED PICTURE 99V9, DISPLAY.
02 HOURS-SCHEDULED PICTURE 99V9.

Internal Decimal Item

Format:

level-number { data-name
 FILLER } [REDEFINES-clause] [OCCURS-clause]
PICTURE IS numeric-form USAGE IS COMPUTATIONAL-3
[VALUE IS numeric-literal].

Example:

02 YEAR-TO-DATE PICTURE S99999999V99 COMPUTATIONAL-3.

Binary Item

Format:

level-number { data-name
FILLER } [REDEFINES-clause] [OCCURS-clause]
PICTURE IS numeric-form USAGE IS COMPUTATIONAL
[VALUE IS numeric-literal].

Example:

03 SUBSCRIPT PICTURE S999 COMPUTATIONAL.

Ext External Floating-Point Item

Format:

level-number { data-name
FILLER } [REDEFINES-clause] [OCCURS-clause]
PICTURE IS fp-form [USAGE IS DISPLAY].

Example:

02 GAMMA PICTURE +.9(8)E+99.

Ext Internal Floating-Point Item

Format:

level-number { data-name
FILLER } [REDEFINES-clause] [OCCURS-clause]
[USAGE IS { COMPUTATIONAL-1
COMPUTATIONAL-2 }]
[VALUE IS floating-point-literal] .

Example:

02 DEVIATION COMPUTATIONAL-1.

ITEM INFORMATION CLAUSES

Each entry consists of one or more clauses that provide information about the item. Listed below are the options available.

USAGE Clause

The USAGE clause describes the form in which data is represented.

The USAGE clause may be written at any level. At a group level, it applies to each elementary item in the group. The usage of an elementary item must not contradict the usage explicitly stated for a group to which the item belongs. If USAGE is not specified, the usage of an item is assumed to be DISPLAY. The format of the USAGE clause is:

$$\left[\text{USAGE IS } \left\{ \begin{array}{l} \text{DISPLAY} \\ \text{COMPUTATIONAL} \\ \text{COMPUTATIONAL-1} \\ \text{COMPUTATIONAL-2} \\ \text{COMPUTATIONAL-3} \end{array} \right\} \right]$$

The DISPLAY option specifies that the item is stored in character form, one character per byte.

The COMPUTATIONAL option specifies a binary data item occupying two, four, or eight character positions corresponding to specified decimal lengths of 1-4, 5-9, and 10-18, respectively. The leftmost bit of the reserved area is the operational sign. Computational items are aligned at the next half-word or full-word boundary, as appropriate.

The COMPUTATIONAL-1 option specifies a data item stored in short-precision floating-point format.

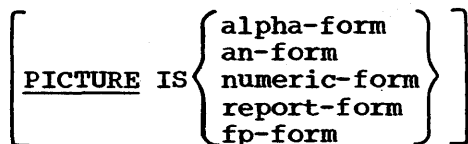
The COMPUTATIONAL-2 option specifies a data item stored in long-precision floating-point format.

The COMPUTATIONAL-3 option specifies that the item is stored in packed decimal format: two digits per character position, with the low-order half character containing the sign.

PICTURE Clause

The PICTURE clause specifies a detailed description of an elementary level data item and may include specification of special report editing.

The general format of the PICTURE clause is:



An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
003	008	DATA DIVISION.
		.
		.
003	013	02 A PICTURE X(68).
003	014	02 B PICTURE X(12).

Refer to Appendix E, Figure 35, for the relationship between the example above, and the sample program given therein.

The options are described in the following text.

ALPHA-FORM OPTION: This option represents an alphabetic item. The PICTURE of an alphabetic item can contain only the character A. An A indicates that the character position will always contain one of the 26 letters of the English alphabet or a space.

AN-FORM OPTION: This option applies to alphanumeric items. The PICTURE of an alphanumeric item can contain only the character X. An X indicates that the character position will always contain a character from the EBCDIC set.

NUMERIC-FORM OPTION: This option refers to a fixed-point numeric item. The PICTURE of a numeric item may contain a valid combination of the following characters:

CHARACTER	MEANING
9	The character 9 indicates that the actual or conceptual digit position contains a numeric character.
V	The character V indicates the position of an assumed decimal point. Since a numeric item cannot contain an actual decimal point, an assumed decimal point is used to provide the compiler with information concerning the decimal alignment of items involved in computations. Storage is never reserved for the character V.

- P** The character P represents a numeric digit position for which storage is never reserved and which always is treated as if it contained a zero. P (or a group of Ps) is used to indicate the location of an assumed decimal point. For example, an item composed of the digits 123 would be treated by an arithmetic procedure statement as 123000 if its PICTURE were 999PPP; or as .000123 if its PICTURE were VPPP999. The character V may be used or omitted as desired. When used, V must be placed in the position of the assumed decimal point, to the left or right of the P or Ps that have been specified.
- S** The character S indicates the presence of an operational sign. If used, it must be the leftmost character of the PICTURE. For a binary item a sign is always present in the item; hence, the presence of S in a numeric-form PICTURE is required. For internal and external decimal items used as input, an S must be written if and only if the item contains a sign. For internal and external decimal items developed by the execution of COBOL statements, the compiler will develop a sign if and only if an S is written in the PICTURE. If an S is not written, the sign position is occupied by a bit configuration interpreted as positive, but which does not represent an overpunch.

REPORT-FORM OPTION: This option refers to a report item. The editing characters that may be combined to describe a report item are: 9 V P . Z * CR DB , 0 B \$ + -. The characters 9, P, and V have the same meaning as for a numeric item. The meanings of the other allowable editing characters are described in the following text.

CHARACTER MEANING

- .** The decimal point character (.) specifies that an actual decimal point is to be inserted in the indicated position and the source item is to be aligned accordingly. Numeric character positions to the right of an actual decimal point in a PICTURE must consist of characters of one type (i.e., * or Z or 9 or \$ or + or -).
- Z** The character Z is the zero suppression character. Each Z in a PICTURE represents a digit position. Leading zeros to be placed in positions defined by Z are suppressed, leaving the position blank. Zero suppression also terminates upon encountering the decimal point (. or V). Z may appear to the right of the decimal point only if all digit positions are represented by Zs. (A Z cannot appear to the right of a 9 anywhere.) The PICTURE ZZZ.ZZ is equivalent to a combination of the BLANK clause and the PICTURE ZZZ.99.
- *** The asterisk is the "check protection" replacement character which is similar to Z, except that leading zeros are replaced by asterisks. An * must not appear anywhere to the right of a 9. The

BLANK WHEN ZERO clause may not be applied to an item having an * in its PICTURE.

CR DB CR and DB are called credit and debit symbols and may appear only at the right end of a picture. These symbols occupy two character positions and indicate that the specified symbol is to appear in the indicated positions if the value of a source item is negative. If the value is positive or zero, spaces will appear instead.

, 0 B The comma, zero, and B specify insertion of comma, zero, and space, respectively. Each insertion character is counted in the size of the data item, but does not represent a digit position. The presence of zero suppression(Z) or check protection (*) indicates that suppression of leading insertion characters also takes place with associated space or asterisk replacement. These characters may also appear in conjunction with a floating string, as described in the following text.

A floating string is defined as a leading, continuous series of either \$, + or -, or a string composed of one such character interrupted by Bs and/or commas and/or V or actual decimal point. For example:

```

$$, $$$, $$$
++++
---, ---, ---
$$$B$$$
+(8)V++
$$, $$$-$$

```

A floating string containing n+1 occurrences of \$, + or - defines n digit positions. When moving a numeric value into a report item, the appropriate character floats from left to right, so that the developed report item has exactly one actual \$, + or - immediately to the left of the most significant nonzero digit, in one of the positions indicated by \$, + or - in the PICTURE. Blanks are placed in all character positions to the left of the single developed \$, + or -. If the most significant digit appears in a position to the right of positions defined by a floating string, then the developed item contains \$, + or - in the rightmost position of the floating string, and nonsignificant zeros may follow. The presence of an actual or implied decimal point in a floating string is treated as if all digit positions to the right of the point were indicated by the PICTURE character 9, and a BLANK WHEN ZERO clause was written for the item. In the following examples, b represents a blank in the developed items.

PICTURE	Numeric Value	Developed Item
\$\$\$999	14	bb\$014
--, ---, 999	-456	bbbbbb-456

A floating string need not constitute the entire PICTURE of a report item, as shown in the preceding examples. Restrictions on characters that may follow a floating string are given later in this description.

When B, comma, or zero appear to the right of a floating string, the string character floats through these characters in order to be as close to the leading digit as possible.

The character B in a floating string indicates that an embedded blank is to appear in the indicated position, unless the position immediately precedes the nonzero, leading significant digit. Embedded Bs in a PICTURE need not be single characters. Thus, \$\$BB\$\$ is a valid PICTURE for a report item. The character comma in a floating string operates similarly, except that the appropriate character appears in the developed item instead of a blank.

The character V in a floating string serves merely to indicate alignment of the assumed decimal point.

\$	The character \$, + or - may appear in a PICTURE
+	either singly or in a floating string. As a fixed
-	sign control character, the + or - must appear as
	either the first or last symbol in the PICTURE,
	but not both. The plus sign indicates that the
	sign of the item is indicated by either a plus or
	minus placed in the character position, depending
	on the algebraic sign of the numeric value placed
	in the report field. The minus sign indicates
	that blank or minus is placed in the character
	position, depending on whether the algebraic sign
	of the numeric value placed in the report field is
	positive or negative, respectively. As a fixed
	insertion character, the dollar sign may appear
	only once in a PICTURE.

Other rules for a report item PICTURE are as follows:

1. The appearance of one type of floating string precludes any other floating string.
2. There must be at least one digit position character.
3. If there are no 9s, BLANK WHEN ZERO is implied unless all numeric positions are *.
4. The appearance of a floating sign string or fixed plus or minus insertion characters precludes the appearance of any other of the sign control insertion characters, namely, + - CR or DB.
5. The characters in a PICTURE to the left of an actual decimal point (or in the entire PICTURE if no decimal point is given), excluding the characters that comprise a floating string, are subject to the following restrictions:
 - a. Z may not follow * or 9 or a floating string.
 - b. * may not follow 9 or Z or a floating string.
6. The characters to the right of a decimal point up to the end of a PICTURE, excluding the fixed insertion characters + - CR DB (if present), are subject to the following restrictions:
 - a. Only one type of digit position character may appear. That is, asterisks, Zs, 9s, and floating string digit position characters \$ + - are mutually exclusive.
 - b. If any of the numeric character positions to the right of a decimal point is represented by + or - \$ or Z or *, then all the numeric character positions must be represented by the same characters.
7. A floating string must begin with at least two consecutive appearances of + or - or \$.
8. The PICTURE character 9 can never appear to the left of a floating or replacement character.
9. Floating or replacement characters + - Z \$ or * cannot be mixed in a PICTURE description. They may appear with fixed characters as follows:
 - a. * or Z with fixed \$
 - b. \$ (fixed or floating) with fixed rightmost + or -
 - c. * or Z with fixed leftmost + or -
 - d. * or Z with fixed rightmost + or -

Ext FP-FORM OPTION: This option refers to an external floating-point item. The PICTURE of an external floating-point item consists of all of the following:

1. + or - (+ indicates that a plus sign represents positive values and that a minus sign represents negative values; - indicates that a blank represents positive values and that a minus sign represents negative values).
2. One to sixteen 9s representing mantissa with a decimal point or V
3. The letter E
4. + or - (see note 1 above)
5. Two 9s representing the exponent

General Notes: The following considerations pertain to use of the PICTURE clause.

1. A PICTURE clause must only be used at the elementary level.
2. An integer enclosed in parentheses and following A X 9 Z * 0 P - B \$ or + indicates the number of consecutive occurrences of the PICTURE character.
3. All characters, except P V and S are counted in the total size of a data item. CR and DB occupy two character positions.
4. A maximum of 30 character positions is allowed in a PICTURE character string. For example, PICTURE A(79) consists of five PICTURE characters.
5. A PICTURE must consist of at least one of the characters A X 9 * Z or at least a pair of one of the characters + or - or \$.
6. The characters . S V CR and DB can appear only once in a picture. CR and DB may not both appear in the same PICTURE.
7. An item can possess only one sign.

The examples in Figure 10 illustrate the use of PICTURE to edit data. In each example a movement of data is implied, as indicated by the column headings.

Source Area		Receiving Area	
PICTURE	Data Value	PICTURE	Edited Data
S99999	-12345	-ZZ,ZZ9.99	-12,345.00
S99999V	00123	\$ZZ,ZZ9.99	\$ 123.00
S9(5)	00100	\$ZZ,ZZ9.99	\$ 100.00
9(5)	00000	\$ZZ,ZZZ.99	\$.00
9(5)	00000	\$ZZ,ZZZ.ZZ	
999V99	123.45	\$ZZ,ZZ9.99	\$ 123.45
V99999	.12345	\$ZZ,ZZ9.99	\$ 0.12
9(5)	12345	\$**,**9.99	\$12,345.00
9(5)	00123	\$**,**9.99	\$***123.00
9(5)	00000	\$**,**9.99	\$*****.00
9(5)	00000	\$**,**,**	*****
99V999	12.345	\$**,**9.99	\$***12.34
9(5)	12345	\$\$\$,\$\$9.99	\$12,345.00
9(5)	00123	\$\$\$,\$\$9.99	\$123.00
9(5)	00000	\$\$\$,\$\$9.99	\$0.00
9(4)V9	1234.5	\$\$\$,\$\$9.99	\$1,234.50
V9(5)	.12345	\$\$\$,\$\$9.99	\$0.12
S99999V	-12345	-ZZZZ9.99	-12345.00
S9(5)V	12345	-ZZZZ9.99	12345.00
S9(5)	-00123	-ZZZZ.99	- 123.00
S99999	12345	ZZZZ9.99-	12345.00
S9(5)	-12345	ZZZZ9.99-	12345.00-
S9(5)	01234	-----99	1234.00
S9(5)	-00001	-----99	-1.00
S9(5)	12345	+ZZZZZ.99	+12345.00
S9(5)	-12345	+ZZZZZ.99	-12345.00
S9(5)	12345	ZZZZZ.99+	12345.00+
S9(5)	-12345	ZZZZZ.99+	12345.00-
S9(5)	00123	+++++.99	+123.00
S9(5)	00001	-----99	1.00
9(5)	00123	+++++.99	+123.00
9(5)	00123	-----99	123.00
9(5)	12345	BB999.00	345.00
S9(5)	-12345	\$\$\$\$\$.99CR	\$12345.00CR
S9(5)	12345	\$\$\$\$\$.99CR	\$12345.00

Figure 10. Editing Applications of the PICTURE Clause

BLANK Clause

This clause specifies that the item being described is filled with spaces whenever the value of the item is zero. The BLANK clause may only be used for report items specified at an elementary level.

The format of the BLANK clause is:

[BLANK WHEN ZERO]

VALUE Clause

The VALUE clause defines condition-name values and specifies the initial value of Working-Storage items. The format of this clause is:

[VALUE IS literal]

The size of a literal given in a VALUE clause must be less than or equal to the size of the item as given in the PICTURE clause, with the provision that the literal must also include leading or trailing zeros to reflect Ps in the PICTURE. The positioning of the literal within a data area is the same as the positioning that would result from specifying a MOVE of the literal to the data area. The type of literal written in a VALUE clause depends on the type of data item.

When an initial value is not specified, no assumption should be made regarding the initial contents of an item in Working-Storage.

The VALUE clause can only be specified for elementary items other than report and external floating point.

In the File Section and Linkage Section the VALUE clause can only appear in conjunction with a level 88 item.

The VALUE clause must not be written in a Record Description entry that also has an OCCURS or REDEFINES clause, or in an entry that is subordinate to an entry containing an OCCURS or REDEFINES clause. In the latter case, an 88 level VALUE clause may be subordinate to the OCCURS or REDEFINES clause.

REDEFINES Clause

This clause specifies that the same area is to contain different data items, or provides an alternative grouping or description of the same data. The format of the REDEFINES clause is:

data-name-1 [REDEFINES data-name-2]

When written, the REDEFINES clause must be the first clause following the data-name that defines the entry.

When an area is redefined, all descriptions of the area remain in effect. Thus, if B and C are two separate items that share the same storage area due to redefinition, the procedure statements MOVE X TO B or MOVE Y TO C could be executed at any point in the program. In the first case, B would assume the value of X and take the form specified by the description of B. In the second case, the same physical area would receive Y according to the description of C. A redefinition does not cause any data to be erased and does not supersede a previous description.

The REDEFINES clause must not be used for logical records associated with the same file (i.e., it must not be used at the 01 level in the File Section), since implied redefinition exists. The level number of data-name-2 must be identical to that of the item containing the REDEFINES clause. Redefinition starts at data-name-2, and ends when a level-number less than or equal to that of data-name-2 is encountered.

The entries giving the new description of the area must immediately follow the entries describing the area being redefined. However, additional entries that redefine the same area may intervene.

If data-name-1 is described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2, then data-name-2 must start on a half-word, full-word, or double-word boundary, as appropriate.

A REDEFINES clause may be specified for an item within the scope of an area being redefined; that is, REDEFINES clauses may be specified for items subordinate to items containing REDEFINES clauses.

Except for condition-name entries, entries containing or subordinate to a REDEFINES clause must not contain any VALUE clauses.

The description of data-name-1 or of any item subordinate to data-name-1 may not contain an OCCURS clause with a DEPENDING ON option. Data-name-1 may not be subordinate to an item containing an OCCURS clause. Data-name-2 may not contain an OCCURS clause in its description nor may it be subordinate to an item described by an OCCURS clause. No item subordinate to data-name-2 may be described by an OCCURS clause with a DEPENDING ON option.

Between data-name-2 and data-name-1 there may be no entries having a lower level number than the level number of data-name-2 and data-name-1.

The length of data-name-1, multiplied by the number of occurrences of data-name-1, must be equal to the length of data-name-2.

Data-name-2 need not be written with qualifiers to ensure uniqueness.

Examples of the REDEFINES clause are contained in Figure 3.

OCCURS Clause

The OCCURS clause is used in defining related sets of repeated data, such as tables, lists, vectors, matrices, etc. It specifies the number of times that a data item with the same format is repeated. Record Description clauses associated with an item whose description includes an OCCURS clause apply to each repetition of the item being described. When the OCCURS clause is used, the data-name that is the defining name of the entry must be subscripted whenever it appears in the Procedure Division. If this data-name is the name of a group item, then all data-names belonging to the group must be subscripted whenever they are used.

The OCCURS clause must not be used in any Record Description entry having a level number 01 or 88. The OCCURS clause has the following formats:

Option 1

[OCCURS integer TIMES]

In Option 1, integer represents the exact number of occurrences.

Option 2

[OCCURS integer TIMES DEPENDING ON data-name]

In Option 2, integer refers to the maximum number of occurrences. The use of Option 2 does not imply that the length of the data item is variable, but that the number of occurrences of the item may vary. The record containing the variable number of occurrences of the item is, however, of variable length, as is any group containing the variable number of occurrences.

In Option 2, the actual number of occurrences is equal to the value at object-time of the elementary item called data-name. This value must be a positive integer. Hence, the PICTURE for data-name must describe an integer. Data-name must be an internal decimal, external decimal, or binary item. If data-name appears within the record in which the current Record Description entry also appears, then data-name must precede the variable portion of the record which depends on it. Data-name should be qualified, when necessary, but subscripting is not permitted.

Option 2 has the following restrictions.

1. Only one such clause per logical record is allowed.
2. The clause must appear in the description of either a group that contains the last elementary item of the record, or in the description of the last elementary item itself.
3. The item having an OCCURS clause with a DEPENDING ON option may not itself be contained in a group having any OCCURS clause.

Subscripting: Subscripting provides the facility for referring to data items in a table or list that have not been assigned individual data-names. Subscripting is determined by the appearance of an OCCURS clause in a data description. If an item has an OCCURS clause or belongs to a group having an OCCURS clause, it must be subscripted whenever it is used.

A subscript is a positive nonzero integer whose value determines to which element a reference is being made within a table or list. The subscript may be represented either by a literal or a data-name that has an integral value. Whether the subscript is represented by a literal or a data-name, the subscript is enclosed in parentheses and appears after the terminal space of the name of the element. A subscript must be an internal decimal, external decimal, or binary item.

Binary subscripting of data-names, in general, results in more efficient coding.

Tables may be defined so that more than one level of subscripting is required to locate an element within them. Such a case exists when a group item described with an OCCURS clause contains one or more items also described with OCCURS clauses. A maximum of three levels of subscripting is permitted by COBOL. Multilevel subscripts are always written from left to right, in decreasing order of inclusiveness of the groupings in the table. Subscripts are written within a single pair of parentheses and are separated by a comma followed by a space. For example:

```
01  ARRAY.
   02  VECTOR, OCCURS 2 TIMES.
      03  ELEMENT, OCCURS 3, PICTURE S9(9).
         USAGE IS COMPUTATIONAL.
```

The preceding example would be allocated storage, as shown in Figure 11.

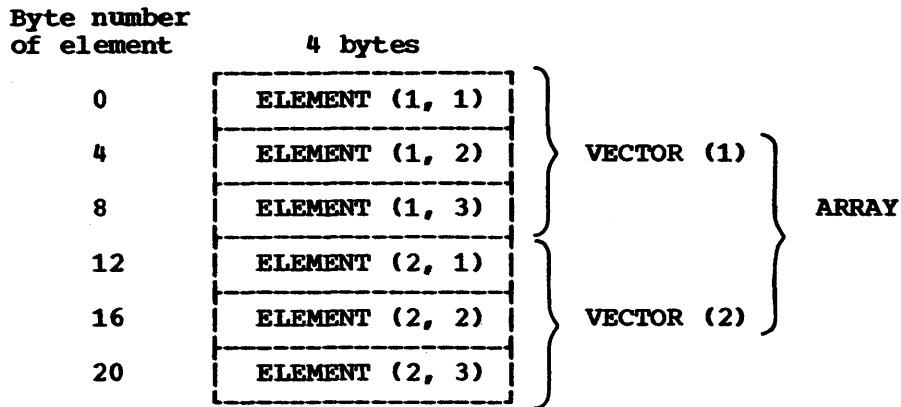


Figure 11. An Example of Subscripting for a Defined Array

A data-name may not be subscripted under the following circumstances:

1. When it is being used as a subscript.
2. When it is being used as a qualifier.
3. When it appears as the defining name of a record description entry.
4. When it appears as data-name-2 in a REDEFINES clause.
5. When it appears as data-name in the DEPENDING ON option of the OCCURS clause or GO TO clause.
6. When it is data-name in a SYMBOLIC KEY, ACTUAL KEY, and RECORD KEY clause.
7. When it is data-name in a LABEL RECORDS clause.

JUSTIFIED RIGHT Clause

This clause may be written only for an elementary alphabetic or alphanumeric item. Its format is:

[JUSTIFIED RIGHT]

When non-numeric data is moved to a field for which JUSTIFIED RIGHT has been specified, the rightmost character of the source field is placed in the rightmost position of the receiving field. The moving of characters continues from right to left until the receiving field is filled. If the length of the source field is greater than that of the receiving field, truncation terminates the move after the leftmost position of the receiving field is filled. If the source field is shorter, the remaining leftmost positions of the receiving field are filled with spaces.

WORKING-STORAGE SECTION

The Working-Storage Section is used to describe areas of storage reserved for intermediate processing of data. This section consists of a series of Record Description entries, each of which describes an item in a work area.

An independent Working-Storage entry describes a single item that is not subdivided and is not itself a subdivision of some other item. Each of these items is defined in a separate Record Description entry, which begins with the special level number 77. All independent Working-Storage entries must precede any items having any of the level numbers 01 through 49.

Data items in the Working-Storage Section that bear a definite relationship to each other must be grouped into records according to the rules for formation of record descriptions. All clauses that are used in Record Description entries may be used in Working-Storage record descriptions. Each data-name in the Working-Storage Section that identifies a record (01 or 77 level) must be unique, since it cannot be qualified by a file-name. Subordinate data-names need not be unique, if they can be made unique by qualification.

No assumption should be made about the initial values of Working-Storage items when these items have not had their initial values defined in a VALUE clause.

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
003	008	DATA DIVISION.
003	009	WORKING-STORAGE SECTION.
003	010	77 MODIFICATION PICTURE X(12), VALUE IS 'PUT ANY DATA'.

Refer to Appendix E, Figure 35, for the relationship between the example above, and the sample program given therein.

Ext LINKAGE SECTION

The Linkage Section describes data passed from another program, or user label record areas.

Record description entries in the Linkage Section provide names and descriptions but storage within the program is not reserved, since the data exists elsewhere. Any Record Description clause may be used to describe items in the Linkage Section, with one exception: the VALUE clause may not be specified for other than level 88 items. In the Linkage Section, level 01 items are assumed to start on a double-word boundary. Refer to Appendix C for a discussion of record alignment.

The Linkage Section is required in any program in which a LABEL RECORDS clause with a data-name option or an ENTRY statement with a USING option appears. A complete discussion of ENTRY is contained in Section 6.

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
003	008	DATA DIVISION. . . .
003	011	LINKAGE SECTION.
003	012	01 PASS FIELD.
003	013	02 A PICTURE X(68).
003	014	02 B PICTURE X(12).

Refer to Appendix E, Figure 35, for the relationship between the example above, and the sample program given therein.

PURPOSE

The Procedure Division of a source program specifies those procedures needed to solve a given problem. These steps (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English, which employ the concept of verbs to denote actions, statements and sentences to describe procedures. The Procedure Division must begin in Margin A with the header PROCEDURE DIVISION followed by a period on a line by itself.

SYNTAX

The format of the Procedure Division is straight forward and exacting. Its constituent parts, in order of hierarchy, are:

- Section
- Paragraph
- Sentence
- Expression
- Statement

The discussion that follows describes the units of expression that constitute the Procedure Division and the way in which they may be combined.

SECTIONS

A section is composed of one or more successive paragraphs and must begin with a section-header beginning in Margin A. A section-header consists of a unique section-name conforming to the rules for procedure-name formation, followed by the word SECTION and a period. A section header must appear on a line by itself, except in the Declaratives portion of the Procedure Division, where it may only be followed immediately by a USE sentence or an INCLUDE statement. The INCLUDE statement is discussed in Section 7. A section-name need not immediately follow the words PROCEDURE DIVISION or END DECLARATIVES. A section ends at the next section-name or at the end of the Procedure Division, or, in the case of Declaratives, at the next section-name or at END DECLARATIVES.

PARAGRAPHS

Paragraphs are logical entities consisting of one or more sentences. Each paragraph must begin with a paragraph-name which must start in Margin A.

A paragraph-name must not be duplicated within the same section. When used as operands in Procedure Division statements, nonunique paragraph-names may be uniquely qualified by writing IN or OF after the paragraph-name, followed by the name of the section in which the paragraph is contained. A paragraph ends at the next paragraph-name or section-name, or at the end of the Procedure Division. In the case of Declaratives, a paragraph ends at the next paragraph-name, section-name, or at END DECLARATIVES.

SENTENCES

A sentence is a single statement or a series of statements terminated by a period and followed by a space. A single comma or semicolon or the word THEN may be used as a separator between statements. A sentence must be contained within Margin B.

EXPRESSIONS

An expression may be defined as a meaningful combination of names, literals, COBOL words, and/or operators which may be reduced to a single value. This definition will become clear after the reader has studied the two types of expressions employed in COBOL, the "arithmetic" expression and the "conditional" expression.

STATEMENTS

A statement consists of a COBOL verb or the word IF or ON, followed by any appropriate operands (data-names, file-names, or literals) and other COBOL words that are necessary for the completion of the statement. The three types of statements are: compiler-directing, imperative, and conditional.

Types of Statements

COMPILER-DIRECTING STATEMENT: A compiler-directing statement directs the compiler to take certain actions at compilation time. A compiler-directing statement contains one of the compiler-directing verbs and its operands. Compiler-directing statements (except for NOTE, COPY, and INCLUDE) must appear as separate single sentences.

IMPERATIVE STATEMENT: An imperative statement specifies an unconditional action to be taken by the object program. An imperative statement consists of a COBOL verb and its operands, excluding the Compiler-Directing verbs and the conditional statements. An imperative statement may also consist of a series of imperative statements.

CONDITIONAL STATEMENT: A conditional statement is a statement containing a condition that is tested to determine which of the alternate paths of program flow is to be taken.

The following are conditional statements:

1. A READ statement
2. A WRITE statement with the INVALID KEY option
- Ext | 3. A REWRITE statement with the INVALID KEY option
4. An arithmetic statement with the SIZE ERROR option
- Ext | 5. An ON statement
6. An IF statement

Although IF and ON are not verbs in the grammatical sense, they are regarded as such in COBOL, inasmuch as they are the key words associated with a particular statement form.

The conditions evaluated in conditional statements are:

1. AT END or INVALID KEY in a READ statement
2. INVALID KEY in a WRITE or REWRITE statement
3. SIZE ERROR in a arithmetic statement
4. The count-condition in an ON statement
5. One of four tests in an IF statement

The conditions in 1 to 4 above are called 'event-conditions.' The conditions in 5 above are called 'test-conditions.'

The formats for the conditions named in 1 to 4 above are discussed in the text for their respective statements. The types of conditions evaluated in an IF statement are discussed in the section "Test-Conditions."

CONDITIONALS

IF Statement

The format of the IF statement is:

```

IF condition [THEN] { statement-1...
                        NEXT SENTENCE }
      [ { ELSE
        { OTHERWISE } { statement-2...
                        NEXT SENTENCE } ]

```

ELSE (or OTHERWISE) NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

Examples of IF Statement:

```

IF SALES ARE NOT EQUAL TO SALES-QUOTA COMPUTE STANDARD-RATE = SALES *
BASE
IF AMOUNT IS LESS THAN 200000 MOVE ' INVENTORY-COUNT' TO PRINTER-AREA.
IF MONTH EQUAL TO 100 GO TO HIT ELSE GO TO LOOP.

```

EVALUATION OF CONDITIONAL STATEMENTS: When a condition is evaluated the following action is taken:

1. If the condition is true, the statements immediately following the condition are executed.
2. If the condition is false the next sentence or the statements following ELSE or OTHERWISE (or the next sentence) are executed.

The AT END, INVALID KEY, and SIZE ERROR conditions are followed by a series of imperative statements. In an ON count-conditional statement, the count- condition is followed by a series of imperative statements (or NEXT SENTENCE) and may be followed by the words ELSE or OTHERWISE followed by a series of statements (or NEXT SENTENCE). The formats of the IF statement describe what may follow the condition in the IF statement.

A series of imperative statements is terminated by one of the following:

1. A period.
2. ELSE or OTHERWISE associated with a previous IF or ON.

In a series of imperative statements executed if a condition is true, only the last statement may be an Option 1 GO TO statement or a STOP RUN statement; otherwise the series of statements would contain statements to which control cannot flow.

For example, in the following paragraph, the statement MOVE A TO B could never be executed whether or not the AT END condition were found to be false.

W. READ PAYROLL-RECORD AT END GO TO Y MOVE A TO B.

Figure 12 is a flowchart showing how an IF or ON conditional statement is evaluated.

Figure 13 is a flowchart showing how a conditional statement other than IF or ON is evaluated.

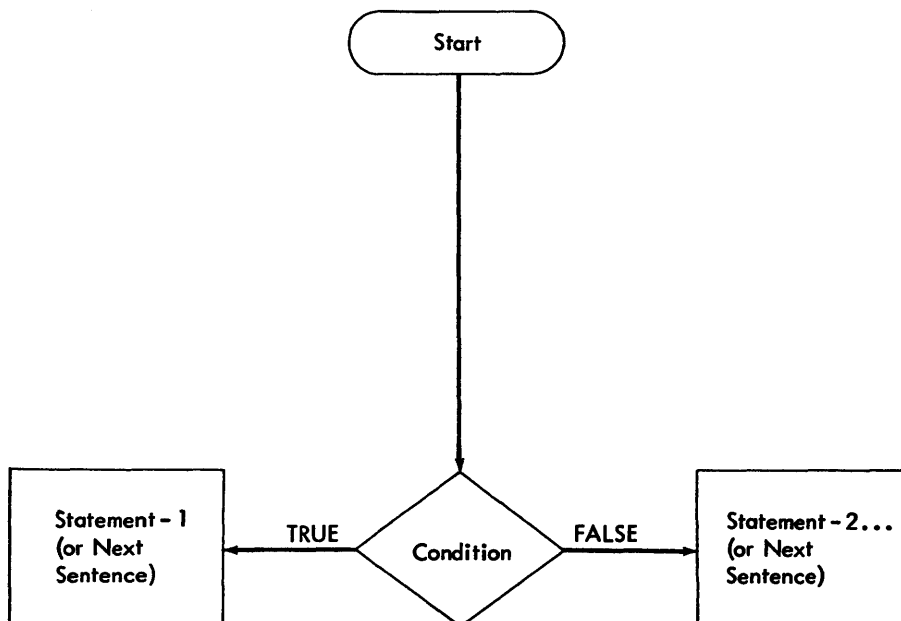


Figure 12. Evaluation of IF or ON Conditional Statement

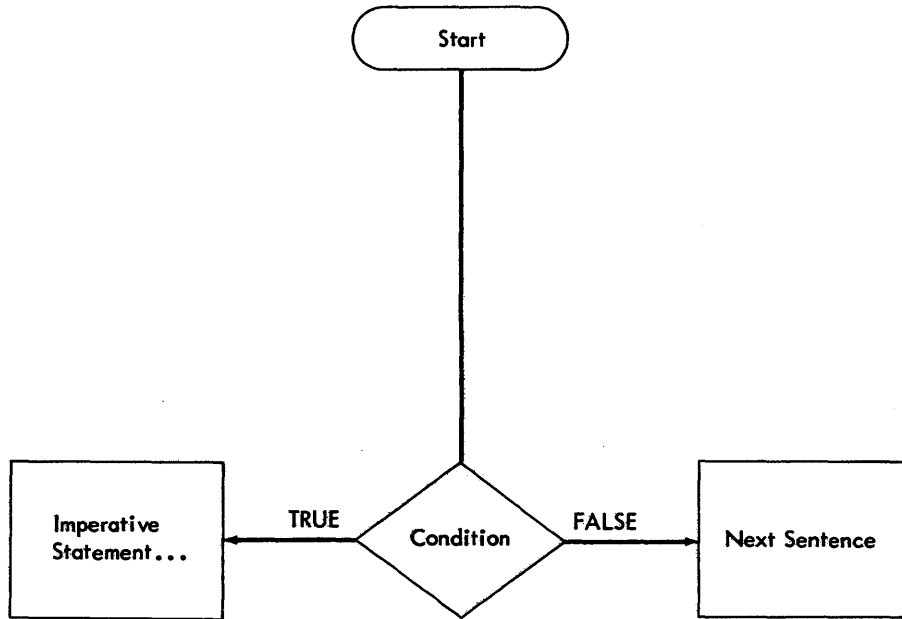
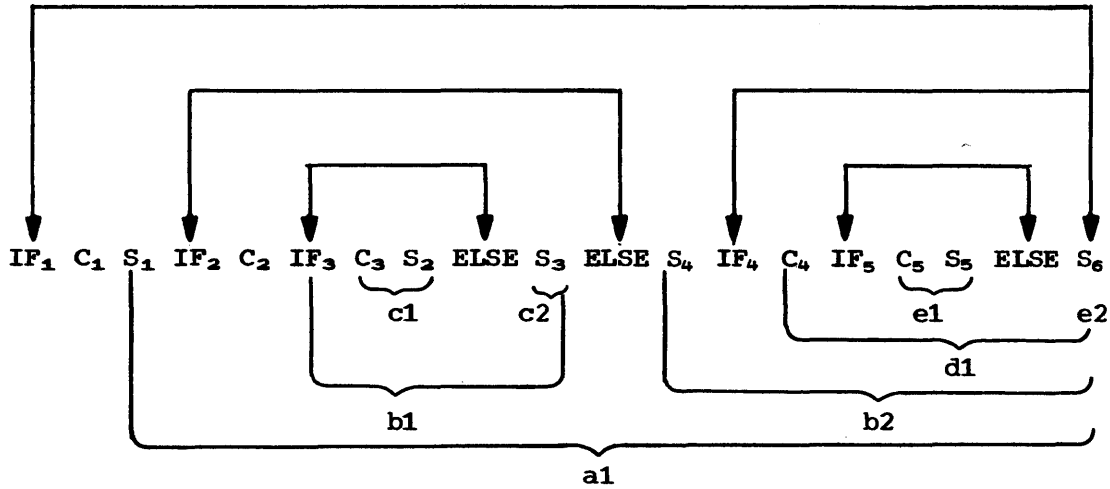


Figure 13. Evaluation of Conditional Statement other than IF or ON

NESTED IF STATEMENTS: Statement-1 and statement-2 in IF statements may consist of one or more imperative statements and/or a conditional statement. If a conditional statement appears as statement-1 or as part of statement-1, it is said to be nested. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already been paired with an ELSE. In the conditional statement in Figure 14, C stands for condition; S stands for any number of imperative statements; and the pairing of IF and ELSE is shown by the lines connecting them.

Figure 15 is a flowchart indicating the logical flow of the conditional statement in Figure 14.



- a1 - Statement-1 for IF₁
(If C₁ is false, the next sentence is executed, since there is no ELSE for it.)
- b1 - Statement-1 for IF₂
- b2 - Statement-2 for IF₂
- c1 - Statement-1 for IF₃
- c2 - Statement-2 for IF₃
- d1 - Statement-1 for IF₄
(If C is false, the next sentence is executed, since there is no ELSE for it.)
- e1 - Statement-1 for IF₅
- e2 - Statement-2 for IF₆

Figure 14. Conditional Statements with Nested IF Statements

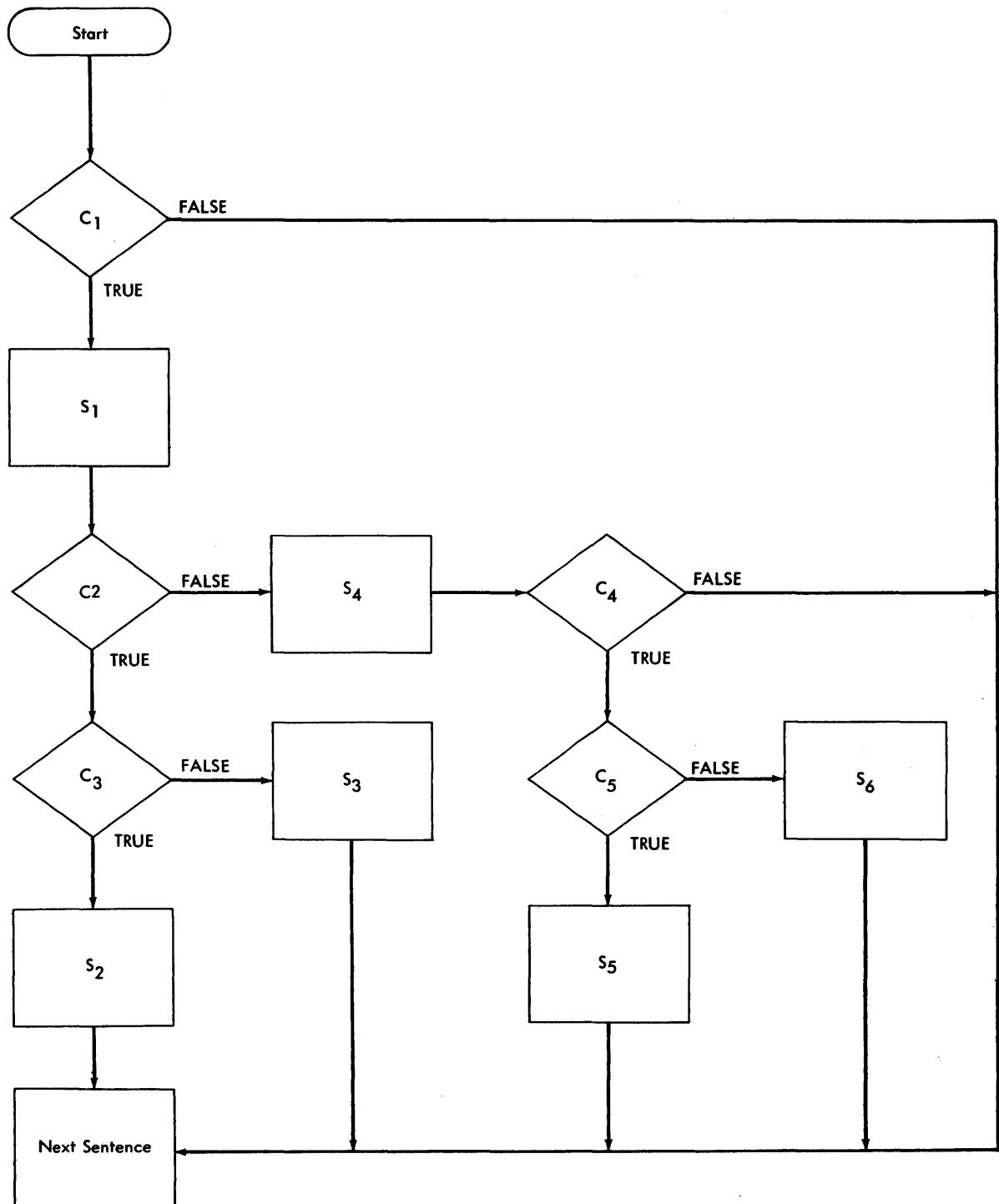


Figure 15. Logical Flow of Conditional Statement with Nested IF Statements

TEST-CONDITIONS: A test-condition is an expression that, taken as a whole, may be either true or false, depending on the circumstances existing when the expression is evaluated.

There are five types of simple test-conditions. When preceded by the word IF, each constitutes one of five types of tests: relation test, sign test, class test, condition name test, overflow test.

The word NOT may be used in any simple test-condition to make the relation specify the opposite of what it would express without the word NOT. For example, AGE NOT GREATER THAN 21 is the opposite of AGE GREATER THAN 21. NOT may also precede an entire condition, as in NOT (AGE GREATER THAN 21). AGE NOT GREATER THAN 21 and NOT (AGE GREATER THAN 21) are identical in meaning.

Relation Test: A relation test involves the comparison of two operands, either of which can be a data-name, a literal, or an arithmetic expression. The comparison of two literals is not permitted. A figurative constant may be used instead of either literal-1 or literal-2 in a relation test.

The format for a relation test is:

$$\left. \begin{array}{l} \text{data-name-1} \\ \text{arithmetic-expression-1} \\ \text{figurative-constant-1} \\ \text{literal-1} \end{array} \right\} \text{ IS [NOT] } \left\{ \begin{array}{l} > \\ < \\ = \\ \text{GREATER THAN} \\ \text{LESS THAN} \\ \text{EQUAL TO} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \text{data-name-2} \\ \text{arithmetic expression-2} \\ \text{figurative constant-2} \\ \text{literal-2} \end{array} \right\}$$

The symbol > is equivalent to the reserved words GREATER THAN. The symbol < is equivalent to the reserved words LESS THAN. The equal sign is equivalent to the reserved words EQUAL TO.

COMPARISON OF NUMERIC ITEMS: For numeric items, a relation test determines that the value of one of the items is less than, equal to, or greater than the other, regardless of the length. Numeric items are compared algebraically after alignment of decimal points. Zero is considered a unique value, regardless of length, sign, or implied decimal-point location of an item.

COMPARISON OF NON-NUMERIC ITEMS: For non-numeric items, a comparison results in the determination that one of the items is less than, equal to, or greater than the other, with respect to the binary collating sequence of characters in the IBM Extended BCD Interchange Code.

If the non-numeric items are of the same length, the comparison proceeds by comparing characters in corresponding character positions, starting from the high-order position and continuing until either a pair of unequal characters or the low-order position of the item is compared. The first pair of unequal characters encountered is compared for relative position in the collating sequence. The item containing the character that is positioned higher in the collating sequence is the greater item. The items are considered equal after the low-order position is compared.

If the non-numeric items are of unequal length, comparison proceeds as described for items of the same length. If this process exhausts the characters of the shorter item, the shorter item is less than the longer, unless the remainder of the longer item consists solely of spaces, in which case, the items are equal.

Figure 16 indicates the characteristics of the items being compared and the type of comparison made. A blank box in Figure 16 indicates that the test is not permitted.

		SECOND OPERAND									
		GR	AL	AN	ED	ID	BI	EF	IF	RP	FC
F I R S T O P E R A N D	Group Item (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN
	Alphabetic Item (AL)	NN	NN	NN							NN ¹
	Alphanumeric (non-report) Item (AN)	NN	NN	NN	NN ⁵					NN	NN
	External Decimal Item (ED)	NN		NN ⁵	NU	NU	NU	NU	NU		NN ³
	Internal Decimal Item (ID)	NN			NU	NU	NU	NU	NU		NU ²
	Binary Item (BI)	NN			NU	NU	NU	NU	NU		NU ²
	External Floating-point Item (EF)	NN			NU	NU	NU	NU	NU		NU ²
	Internal Floating-point Item (IF)	NN			NU	NU	NU	NU	NU		NU ²
	Report Item (RP)	NN		NN						NN	NN ⁴
	Figurative Constant (FC)	NN	NN ¹	NN	NN ³	NU ²	NU ²	NU ²	NU ²	NN ⁴	

Abbreviations for Types of Comparison

NN Comparison as described for non-numeric items.

NU Comparison as described for numeric items.

¹ Permitted with the figurative constants SPACE and ALL 'character' where character must be alphabetic.

² Permitted only if figurative constant is ZERO.

³ Permitted only if figurative constant is ZERO or ALL 'character' where character must be numeric.

⁴ Not permitted with figurative constant QUOTE.

⁵ External decimal field must consist of integers.

Figure 16. Permissible Comparisons

Sign Test: This type of condition tests whether or not the value of a numeric item is less than zero (NEGATIVE), greater than zero (POSITIVE), or is zero (ZERO). The value zero is considered neither positive nor negative.

The format for a sign test is:

$$\left\{ \begin{array}{l} \text{data name} \\ \text{arithmetic-expression} \end{array} \right\} \text{ IS [NOT] } \left\{ \begin{array}{l} \text{POSITIVE} \\ \text{ZERO} \\ \text{NEGATIVE} \end{array} \right\}$$

Class Test: When a class test is specified, determination is made as to whether or not an item consists solely of the following:

1. The characters 0 through 9 (NUMERIC)
2. The characters A through Z and space (ALPHABETIC)

The ALPHABETIC test may be performed on elementary alphabetic or alphanumeric items.

The NUMERIC test may be performed on elementary alphanumeric, internal decimal, or external decimal items.

The format for the class test is:

```
data-name IS [NOT] {
                     NUMERIC
                     ALPHABETIC
                   }
```

If the last character of an otherwise numeric field contains a digit with a sign over punch, the field is considered numeric. For a single character alphanumeric field containing a digit with a sign overpunch, the tests IF NUMERIC and IF ALPHABETIC will both be considered true while the NOT form of the tests will both be false.

Condition-name Test: The format for condition-name test is:

[NOT] condition-name

A condition-name test is one in which a conditional variable is tested to see whether or not its value is equal to the value specified for a condition-name associated with it. For example, in a program processing a payroll, the data item MARITAL-STATUS (the conditional variable) might be a code indicating whether an employee is married, divorced, or single. Assume that if MARITAL-STATUS has the value of 1, the employee is single; if it has the value of 2, he is married; and if it has the value of 3, he is divorced. To determine whether or not an employee is married, the programmer could test this condition by using a simple relational condition in a conditional statement such as IF MARITAL-STATUS = 2 SUBTRACT MARRIED-DEDUCTION FROM GROSS. Alternatively, he can associate a condition-name with each value that MARITAL-STATUS might assume. Thus, in the Data Division, the condition-names SINGLE, MARRIED, and DIVORCED might be associated with values 1, 2, and 3, respectively. For example:

```
02 MARITAL-STATUS PICTURE 9.
88 SINGLE VALUE IS 1.
88 MARRIED VALUE IS 2.
88 DIVORCED VALUE IS 3.
```

Then, as a shorthand form of the simple relational condition MARITAL-STATUS = 1, the programmer could write the single condition-name SINGLE. Therefore, the following two statements would produce identical results:

```
IF MARITAL-STATUS = 1 GO TO Z.
IF SINGLE GO TO Z.
```

The condition-name test, then, is an alternative way of expressing certain conditions which could be expressed by a simple relational condition.

Ext Overflow Test : This type of condition tests for form overflow of a printer to which a file named in an option 1 APPLY clause is assigned.

The format for the overflow test is:

[NOT] overflow-name

Overflow-name is true if a 'form-overflow' situation exists. Form-overflow exists when an end of page is sensed by an on-line printer. Overflow-name follows the rules for data-name formation.

COMPOUND CONDITIONS: Simple test-conditions can be combined with logical operators according to specified rules to form compound conditions. The logical operators are AND, OR, and NOT. Two or more simple conditions combined by AND and/or OR make up a compound condition.

The word OR is used to mean either or both. Thus, the expression A OR B is true if: A is true, B is true, or both A and B are true. The word AND is used to mean both. Thus, the expression A AND B is true only if both A and B are true. The word NOT is used in the manner described in the subsection "Test-Conditions." Thus, the expression NOT (A OR B) is true if A and B are false; and the expression NOT (A AND B) is true if A is false, B is false, or if both A and B are false.

The logical operators and truth values are shown in Figure 17, where A and B represent simple test-conditions.

Condition		Related Conditions				
A	B	not a	a and b	a or b	not (a and b)	not (a or b)
True	True	False	True	True	False	False
False	True	True	False	True	True	False
True	False	False	False	True	True	False
False	False	True	False	False	True	True

Figure 17. Truth Table

Parentheses may be used to specify the order in which conditions are evaluated. Parentheses must always be paired. Logical evaluation begins with the innermost pair of parentheses and proceeds to the outermost. If the order of evaluation is not specified by parentheses, the expression is evaluated in the following way:

1. AND and its surrounding conditions are evaluated first, starting at the left of the expression and proceeding to the right.
2. OR and its surrounding conditions are then evaluated, also working from left to right.

Thus, the expression: A IS GREATER THAN B OR A IS EQUAL TO C AND D IS POSITIVE would be evaluated as if it were parenthesized as follows:

(A IS GREATER THAN B) OR ((A IS EQUAL TO C) AND (D IS POSITIVE)).

The rules for formation of symbol pairs are shown in Figure 18. The letter C stands for conditional expression. P means that the combination is permissible. A dash means that the combination is not permissible.

		Second Symbol					
		C	OR	AND	NOT	()
F i r s t S y m b o l	C	-	P	P	-	-	P
	OR	P	-	-	P	P	-
	AND	P	-	-	P	P	-
	NOT	P	-	-	-	P	-
	(P	-	-	P	P	-
)	-	P	P	-	-	P

Figure 18. Formation of Symbol Pairs

COMPILER-DIRECTING DECLARATIVES

Declarative sections are identified by compiler-directing statements that specify the circumstances under which a procedure is to be executed in the object program. Declaratives consist of a section-name, followed by the word SECTION and a period, and a USE sentence followed by procedural statements. Declarative sections must be grouped together at the beginning of the Procedure Division, preceded by the key word DECLARATIVES in Margin A, and followed by the key words END DECLARATIVES, where END must also appear in Margin A. DECLARATIVES and END DECLARATIVES must be followed by a period.

The general form for declaratives is:

```
PROCEDURE DIVISION.
DECLARATIVES.
  {section-name SECTION. USE-sentence.
  {paragraph-name. sentence... .} ...} ...
END DECLARATIVES.
```

The occurrence of another section or the words END DECLARATIVES terminates a previous USE section. If there are two or more logical paths within a declarative procedure, these paths must lead to a common path within the section containing them. For options 1 and 2, an ALTER, PERFORM, or GO TO statement within a declarative section must not refer to paragraph-names or section-names outside that declarative section, except that a GO TO statement in an Option 1 or Option 2 USE section may refer to the reserved word MORE-LABELS. For option 3, a GO TO statement within a declarative section can refer to paragraph-names or section names outside that declarative section.

A declarative section may not be referred to by any PERFORM or GO TO statement outside the declarative. Within a given declarative section, there may be no reference to a point outside the declarative.

USE Statement

The USE statement identifies the type of declarative.

There are three options of the USE statement. Each is associated with the following types of procedures.

1. Label-writing procedures
2. Label-checking procedures
3. Error-checking procedures

The formats of the USE statements are:

Ext Option 1

```
USE FOR CREATING [ BEGINNING ] LABELS ON OUTPUT file-name... .  
                  [ ENDING   ]
```

Ext Option 2

```
USE FOR CHECKING [ BEGINNING ] LABELS ON INPUT file-name... .  
                 [ ENDING   ]
```

For options 1 and 2. Records associated with the file being opened or closed cannot be referenced within the declarative section. Conversely, the LABEL records can be referenced only while the declarative is being executed.

Options 1 and 2 are used to provide user label processing procedures. CHECKING refers to an input file; CREATING refers to an output file. In this context, "input" means all files opened as INPUT.

The file can be either an input or output file but not both.

The word BEGINNING refers to user header labels; the word ENDING refers to user trailer labels. Absence of either word indicates that the USE section will process both headers and trailers.

ENDING is not supported for direct-access files, as these do not have trailer labels.

The exit from an option 1 or option 2 USE section is inserted by the compiler following the last statement in the section. All logical program paths within the section must lead to this point. One exception to this rule is allowed: a special exit may be specified by the statement GO TO MORE-LABELS. When an exit is made from a label processing USE section by means of this statement, IOCS is directed to do one of the following:

1. Read an additional user header label or user trailer label and then re-enter the USE section for further checking of labels. In this case, IOCS will only re-enter the USE section if there exists another user label to check. Hence, there need not exist a program path that flows through the last statement in the section. The point of return to the USE section, after exit by means of a GO TO MORE-LABELS statement, is the beginning of the section.
2. Write the current user header label or user trailer label and then re-enter the USE section for further creating of labels. A label is written each time an exit from the USE section takes place. The label is created in an IOCS area.

If no GO TO MORE-LABELS statement is executed, then the USE section is not re-entered to check or create any immediately succeeding user labels.

The user label is contained in an IOCS area. If label processing is desired, the label must be described as a data item in the Linkage Section of the Data Division and must be listed as a data-name in the LABEL RECORDS clause in the File Description entry for the file.

In a declarative section containing an option 1 USE statement, there must be a path of program flow through the last statement of the section, so that writing of user labels can be terminated.

**** Option 3**

USE AFTER STANDARD ERROR PROCEDURE ON file-name.

Option 3 is used to provide user input/output error-processing procedures in addition to the procedures supplied by IOCS for tape or disk.

Within the section, the file associated with the USE sentence cannot be referred to by an OPEN, READ, WRITE, or REWRITE statement. Only a CLOSE statement can be given for the file.

An exit from this type of declarative section can be effected by executing the last statement in the section (normal return), or by means of a GO TO statement. Figure 19 summarizes the facilities and limitations associated with each file-processing technique when an error occurs.

File- Processing Technique		No Error- Processing Declarative Section Written	Type of I/O Statement	Error- Processing Declarative Section Written	
ACCESS	ORGANIZATION			Normal Return	GO TO Exit
SEQUENTIAL (or not specified)	Standard sequential tape	End of job	READ	Continue Processing of file permitted	User limited to CLOSE for file- name
	Standard sequential disk	Diagnostic error message is printed, job is terminated	READ		
	INDEXED DIRECT		WRITE REWRITE		Not applicable
RANDOM	INDEXED DIRECT		READ WRITE REWRITE		

Figure 19. Error-Processing Summary

CONTINUED PROCESSING OF FILE

Refer to Figure 19 normal return. Continued processing of a file is permitted under the following conditions.

1. An error processing procedure exists in the declarative section.

2. Detection of a standard error results in an automatic transfer to the error processing procedure, which enables the user to examine the error condition before continuing to process.
3. At the conclusion of processing an error, it is the programmer's responsibility to update the parameters normally returned by IOCS to the programmer (such as the ACTUAL KEY, in the case of sequential retrieval of a direct file).

COBOL VERBS

The COBOL verbs are the basis of the Procedure Division of a source program.

The organization of the remainder of this section is based on the classifications used in the following list:

Input/Output Verbs

	OPEN
	READ
	WRITE
Ext	REWRITE
	CLOSE
	ACCEPT
	DISPLAY

Data Manipulation Verbs

	MOVE
	EXAMINE
Ext	TRANSFORM

Arithmetic Verbs

ADD
SUBTRACT
MULTIPLY
DIVIDE
COMPUTE

Procedure-Branching Verbs

STOP
GO TO
ALTER
PERFORM

Compiler-Directing Verbs

ENTER
EXIT
NOTE

INPUT/OUTPUT STATEMENTS

OPEN Statement

The OPEN statement initiates the processing of files. When applicable, execution of an OPEN statement initiates label checking for input and output files, and label creation for output files. At this time, appropriate label-handling procedures specified by a USE declarative are executed.

The format of an OPEN statement is:

```

OPEN {
  [INPUT {file-name [WITH NO REWIND] [REVERSED]}...]
  [OUTPUT {file-name [WITH NO REWIND]}...]
  [I-O {file-name}...]
  [OUTPUT {file-name [WITH NO REWIND]}...]
  [INPUT {file-name [WITH NO REWIND] [REVERSED]}...]
  [I-O {file-name}...]
  I-O {file-name}... [OUTPUT {file-name [WITH NO REWIND]}...]
  [INPUT {file-name [WITH NO REWIND] [REVERSED]}...]
}

```

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	020	PROCEDURE DIVISION.
001	021	START. OPEN INPUT FILEB OUTPUT FILEA.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

The OPEN statement must be executed prior to any other input/output statement for any file. The OPEN statement, by itself, does not make an input record available for processing; a READ statement must be executed to obtain the first data-record. For an output file, an OPEN statement makes available an area for development of the first output record. A second OPEN statement for a given file cannot be executed prior to the execution of a CLOSE statement for that file.

The I-O option permits the opening of a direct-access file for both input and output operations.

An OPEN statement for an I-O file performs the same label checking functions as for an input file.

The NO REWIND option should only be written for files assigned to UTILITY device-numbers for which rewinding is possible, e.g., 2400. This option suppresses the rewinding normally associated with opening a file.

The REVERSED option can only be applied to files assigned to specific devices for which the reverse-read feature is available. The REVERSED option may only be used for a file containing type F records.

An example of the OPEN statement is:

```
OPEN OUTPUT X-FILE, INPUT Y-FILE REVERSED, Z-FILE.
```

Note that Z-FILE is not opened REVERSED.

READ Statement

The functions of the READ statement are:

1. For sequential file processing, to make available the next logical record from an input file and to allow performance of specified imperative statements when end-of-file is detected.
2. For nonsequential file processing; to make available a specific record from a direct-access file and to allow execution of statements if the contents of the associated symbolic and/or actual key is found to be invalid.

The format of the READ statement is:

READ file-name RECORD [INTO data-name]

$\left. \begin{array}{l} \text{AT END} \\ \text{INVALID KEY} \end{array} \right\}$ imperative statement...

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001 020		PROCEDURE DIVISION.
		.
001 022		START2. READ FILEB AT END GO TO LABA.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

When a READ statement is executed, the next logical record in the named file becomes accessible in the input area defined by the associated Record Description entry. The file-name must be defined by a File Description entry in the Data Division.

The record remains available in the input area until the next READ statement (or a CLOSE statement) for that file is executed. No reference can be made by any statement in the Procedure Division to information that is not actually present in the current record. Thus, it is not permissible to refer to the nth occurrence of data that appears fewer than n times. If such a reference is made, no assumption should be made about results in the object program.

If more than one logical record is described for the file, implicit redefinition of the area exists. It is the programmer's responsibility to identify which record is present in the area at any given time.

The INTO data-name option is equivalent to a READ statement and a MOVE statement. The data-name specified must be the name of a Working-Storage or a previously opened output record. When this option is used, the current record becomes available in the input area, as well as in

the area specified by data-name. Data will be moved into the data-name area in accordance with the COBOL rules for moving an item into the first record specified for that FD.

The AT END option is required for files for which access is sequential. The AT END portion of the READ statement is executed when an end-of-file condition is detected.

Once the AT END portion of a READ statement has been executed for a file, any subsequent attempt to read from that file or to refer to logical records in that file constitutes an error, unless subsequent CLOSE and OPEN statements have been executed.

If the INVALID KEY option is specified, the statements following INVALID KEY are executed when the contents of actual key and/or symbolic key are invalid.

If ACCESS RANDOM is specified for the file, the symbolic key and/or the actual key of the file must be set to the desired values prior to the execution of the READ statement.

Each time an end-of-volume condition occurs on a file, the READ statement causes the following operations to take place:

1. The volume trailer label checking procedure of IOCS is executed. The user trailer label checking procedures specified in a USE Option 2 sentence are executed, if such labels exist.
2. A volume switch occurs.
3. The volume header label checking procedure subroutine of IOCS is executed. The user header label checking procedures specified in a USE Option 2 sentence declarative are executed, if such labels exist.
4. The next logical record in the file is made available for processing.

If the end-of-volume is also the logical end of file, only the operations specified in item 1 are done and then the statements following AT END are executed.

WRITE Statement

The function of the WRITE statement is to release a logical record for a file specified as OUTPUT or I-O in an OPEN statement.

The formats for the WRITE statement are:

Option 1

```
WRITE record-name [FROM data-name-1]  
      [INVALID KEY imperative statement...]
```


Option 2

WRITE record-name [FROM data-name-1]

[AFTER ADVANCING { data-name-2 } integer] LINES]

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	020	PROCEDURE DIVISION. . .
002	005	WRITE RECORD-1 FROM RECORD-2. GO TO START2.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

An OPEN statement must be executed prior to executing the first WRITE statement for a file. After the WRITE statement is executed, the logical record named by record-name is no longer available.

Data-name-1 must not be the name of an item in the file containing record-name. When FROM data name is written, it is equivalent to the statement MOVE data-name-1 TO record-name followed by the statement WRITE record-name. Moving takes place according to the COBOL rules.

After execution of a WRITE statement with the FROM option, the information in record-name is no longer available, but the information in data-name-1 is available.

When the end-of-volume condition occurs, the WRITE statement causes the following operations to take place in addition to the WRITE.

1. The volume trailer label writing procedure of IOCS is executed. The user trailer label creating procedure, if specified in a USE Option 1 declarative, is executed.
2. A volume switch occurs.
3. The volume header label writing procedure of IOCS is executed. The user header label writing procedure, if specified in a USE option 1 declarative, is executed.
4. The next logical record area in the output file is made available.

If ACCESS RANDOM is specified, the record and/or actual key must be set to the desired values prior to the execution of the WRITE statement.

The INVALID KEY option allows performance of specified imperative statements if, for random access files, the contents of the associated actual key and/or record key is found to be invalid.

The AFTER ADVANCING option is used for output destined to be printed or punched. When this option is used, the first character in each

logical record for the file must be reserved for the control character. It is the user's responsibility to see that the appropriate channels are punched in the carriage control tape. If a WRITE statement with an ADVANCING option is written for a record in a file, every WRITE statement for records in the same file must contain an ADVANCING option. When the AFTER ADVANCING option is used, integer must be unsigned and have the value 0, 1, 2, or 3. The value 0 designates a carriage-control 'eject' (i.e., skip to next page). The value 1 designates single spacing; the value 2, double spacing; and the value 3, triple spacing. If "0" is used, COBOL will skip to a "1" punch in the carriage tape.

Data-name-2 must be an alphanumeric item of length one (i.e., must have PICTURE X). The following chart shows the values that data-name-2 may assume and their interpretations.

<u>Value</u>	<u>Interpretation</u>
b (blank)	single spacing
0	double spacing
-	triple spacing
+	suppress spacing
1 through 9	skip to channel 1 through 9, respectively
A,B,C	skip to channel 10, 11, 12, respectively
V,W	pocket select 1 or 2, respectively on the IBM 1442 or 2520, and P1 or P2 on the IBM 2540

Ext REWRITE Statement

The function of the REWRITE statement is to replace a logical record on a direct-access device with a specified record, if the contents of the associated actual key and/or symbolic key is found to be valid.

The format of the REWRITE statement is:

```
REWRITE record-name [FROM data-name]
  [INVALID KEY imperative-statement...]
```

The READ statement for a file must be executed before a REWRITE statement for a file can be executed. A REWRITE statement can only be written for files opened as I-O.

When the FROM option is used, data-name must not be the name of an item in a file containing record-name. This form of the REWRITE statement is equivalent to the statement MOVE data-name TO record-name followed by the statement REWRITE record-name. Moving takes place according to COBOL rules for moving.

For direct access files, the INVALID KEY procedure is executed when the contents of the actual key and/or the symbolic key are invalid for the file.

If ACCESS RANDOM is specified for the file, the actual key and/or the symbolic key must be set to the desired values prior to the execution of the REWRITE statement.

CLOSE Statement

The CLOSE statement is used to terminate the processing of one or more units or files. The format of the CLOSE statement is:

SYSPUNCH and UPON CONSOLE are omitted, the sum of the sizes of the operands may not exceed the maximum logical record length for the system logical printing device (SYSLSST). The number of data names displayed per print line on SYSLSST is 20 if the total number of characters in these words does not exceed 120. If the system logical printing device used in a display statement is also used for file, results are unpredictable.

Any spaces desired between displayed multiple operands must be explicitly specified.

When SYSPUNCH is written, an 80 character output record is produced, with positions 73 through 80 of the record containing the identification of the originating program (program-ID). If the message size exceeds 72 characters, it is truncated; if less than 72, the remaining positions are filled with spaces.

Data-names described as USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-3 are converted automatically to external format as follows:

1. Internal decimal and binary items are converted to external decimal. Only negative values cause a low-order sign overpunch to be developed.
2. Internal floating point items are converted to external floating point. No other data items require conversion.

For example, if two binary items have values -32 and 32, then they will be displayed as 3K and 32, respectively.

ACCEPT Statement

The function of the ACCEPT statement is to obtain data from the system logical input device (SYSIPT) , or from the console.

The format of the ACCEPT statement is:

ACCEPT data-name [FROM CONSOLE]

Data-name may be either a fixed-length group item or an elementary alphabetic, alphanumeric, external decimal or external floating-point item. One logical record is read and the appropriate number of characters are transferred from left to right into the area reserved for data-name. No editing or error-checking of the incoming data is done.

If the input/output device specified by an ACCEPT statement is the same one as designated for a READ statement, the results may be unpredictable.

When FROM CONSOLE is specified data-name may not exceed 72 character positions in length.

When an ACCEPT statement with the FROM CONSOLE option is executed, the following action is taken:

1. A system-generated message 'AWAITING REPLY' is automatically displayed.
2. Execution is suspended. When a console input message is identified by the Control Program, execution of the ACCEPT statement is resumed and the message is transferred to the specified data-name.

When the FROM CONSOLE option is not written, one logical record is read from the system logical input device (SYSIPT).

If the system logical input device used in an accept statement is also used for a file, the results are unpredictable.

Figure 20 states restrictions of input-output statements. Y means that the statement may appear; R indicates it may appear with restrictions; N indicates that it must not.

Statement	Appearing In:			
	Label Checking Declarative	Label Creating Declarative	Main Body of Procedure Division	Debug Packet
OPEN CLOSE	R*	R*	Y	Y
READ WRITE REWRITE	R*	R*	Y	Y
DISPLAY	Y	Y	Y	Y
ACCEPT FROM CONSOLE	Y	Y	Y	Y
ACCEPT (from SYSIPT)	Y	Y	Y	Y

*Only permitted for files other than the one for which entry into the declarative was made.

Figure 20. Restrictions for Input/Output Statements

DATA MANIPULATION STATEMENTS

MOVE Statement

The MOVE statement is used to move data from one area of main storage to another and to perform conversions and/or editing on the data that is moved. The MOVE statement has the following format:

MOVE { data-name-1
literal } TO data-name-2 ...

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
003	015	PROCEDURE DIVISION. . . .
003	019	MODIFY. MOVE MODIFICATION TO B.

Refer to Appendix E, Figure 35, for the relationship between the example above, and the sample program given therein.

The data represented by data-name-1 or the specified literal is moved to the area designated by data-name-2. The same information is also moved to any additional receiving areas mentioned in the statement.

When a group item is involved in a simple move, the data is moved as a group, and without regard to descriptions of items subordinate to the group (i.e. without editing, data conversion etc.).

The following considerations pertain to moving items:

1. Numeric (external decimal, internal decimal, binary, external floating, internal floating, numeric literals, and ZERO) to numeric or report:
 - a. The items are aligned by decimal points, with insertion of zeros or truncation on either end, as required.
 - b. When the USAGE of the source field and receiving field differs, conversion to the USAGE of the receiving field takes place.
 - c. The items may have special editing performed on them with suppression of zeros, insertion of a dollar sign, commas, etc., and decimal point alignment, as specified by the receiving area.
2. All other permissible combinations:
 - a. The characters are placed in the receiving area from left to right, unless the receiving field is specified as JUSTIFIED RIGHT.
 - b. If the receiving field is not completely filled by the data being moved, the remaining positions are filled with spaces.
 - c. If the source field is longer than the receiving field, the move is terminated as soon as the receiving field is filled.

Figure 21 contains several examples illustrating MOVE.

Source Field		Receiving Field		
PICTURE	Value	PICTURE	Value before MOVE	Value after MOVE
99V99	1234	99V99	9876	1234
99V99	1234	99V9	987	123
9V9	12	99V999	98765	01200
XXX	A2B	XXXXX	Y9X8W	A2Bbb
9V99	123	99.99	87.65	01.23
AAAAAA	REPORT	AAA	JKL	REP

Figure 21. Examples of Data Movement

Note that, in the fourth example, the information in any excess positions of a non-numeric receiving area is replaced by spaces at the right.

Figure 22 illustrates all permissible moves for the various data classifications. "Y", means "yes", the move is permitted;" N means "No", the move is not permitted."

Source Field	Receiving Field								
	GR	AL	AN	ED	ID	BI	EF	IF	RP
Group (GR)	Y	Y	Y	N	N	N	N	N	N
Alphabetic (AL)	Y	Y	Y	N	N	N	N	N	N
Alphanumeric (AN)	Y	Y	Y	N	N	N	N	N	N
External Decimal (ED)	Y	N	Y ¹	Y	Y	Y	Y	Y	Y
Internal Decimal (ID)	Y	N	Y ¹	Y	Y	Y	Y	Y	Y
Binary (BI)	Y	N	Y ¹	Y	Y	Y	Y	Y	Y
External Floating-Point (EF)	Y	N	N	Y	Y	Y	Y	Y	Y
Internal Floating-Point (IF)	Y	N	N	Y	Y	Y	Y	Y	Y
Report (RP)	Y	N	Y	N	N	N	N	N	N
ZEROS	Y	N	Y	Y	Y	Y	Y	Y	Y
SPACES	Y	Y	Y	N	N	N	N	N	N
All 'character', HIGH-VALUES, LOW-VALUES, QUOTES	Y	N	Y	N	N	N	N	N	N

¹ For integers only.

Figure 22. Permissible Moves

EXAMINE Statement

The EXAMINE statement is used to replace certain occurrences of a given character and/or to count the number of such occurrences in a data item.

The EXAMINE statement has the following two formats:

Option 1

EXAMINE data-name TALLYING { ALL
LEADING
UNTIL FIRST } 'character-1'

[REPLACING BY 'character-2']

Option 2

EXAMINE data-name REPLACING { ALL
LEADING
UNTIL FIRST
FIRST } 'character-1'

BY 'character-2'

Data-name in each option must refer to a data item whose USAGE is DISPLAY.

Character-1 and character-2 must be single-character non-numeric literals (i.e., enclosed in quotation marks) and members of the set of allowable characters for the data item. For example, a '2' cannot replace an 'A' in an alphabetic item, but may do so in an alphanumeric item.

The use of figurative constants instead of character-1 or character-2 is permitted.

When Option 1 is used, a count is made at object time of the number of occurrences of the specified character in data-name, and this count replaces the value of the special binary data item TALLY, whose length is five decimal digits. TALLY may also be used as a data-name in other procedural statements.

The count at object time depends on which of the following three TALLYING options is employed:

1. If ALL is specified, all occurrences of character-1 in the data item are counted.
2. If LEADING is specified, the count represents the number of occurrences of character-1 prior to encountering a character other than character-1. Examination proceeds from left to right.
3. If UNTIL FIRST is specified, the count represents the number of characters other than character-1 encountered prior to the first occurrence of character-1. Examination proceeds from left to right.

When the REPLACING option is used (either in option 1 or option 2), the replacement of characters depends on which of the following four REPLACING options is employed:

1. If ALL is specified, character-2 is substituted for each occurrence of character-1.

2. If LEADING is specified, the substitution of character-2 for character-1 terminates when a character other than character-1 is encountered, or when the righthand boundary of the data item is reached. Examination proceeds from left to right.
3. If UNTIL FIRST is specified, the substitution of character-2 terminates as soon as the first character-1 is encountered, or when the righthand boundary is reached. Examination proceeds from left to right.
4. If FIRST is specified, only the first occurrence of character-1 is replaced by character-2. Examination proceeds from left to right.

Sample EXAMINE statements showing the effect of each statement on the associated data item and the TALLY are shown in Figure 23.

EXAMINE Statement	ITEM-1 Before	Data After	Resulting Value of TALLY
EXAMINE ITEM-1 TALLYING ALL '0'	101010	101010	3
EXAMINE ITEM-1 TALLYING ALL '1' REPLACING BY '0'	101010	000000	3
EXAMINE ITEM-1 REPLACING LEADING '*' BY SPACE	**7000	7000	unchanged
EXAMINE ITEM-1 REPLACING FIRST '*' BY '\$'	**1.94	\$*1.94	unchanged

Figure 23. Examples of Data Examination

Ext TRANSFORM Statement

The TRANSFORM statement is used to alter characters according to a transformation rule. For example, it may be used to change the characters in an item to a different collating sequence.

The format of the TRANSFORM statement is:

TRANSFORM data-name-3 CHARACTERS

FROM { figurative-constant-1
non-numeric-literal-1
data-name-1 } TO { figurative-constant-2
non-numeric-literal-2
data-name-2 }

Data-name-3 must be an elementary alphabetic, alphanumeric, or report item, or a group item.

The combination of the FROM and TO options determines what the transformation rule is. These combinations are:

TRANSFORMATION RULE

FROM figurative-constant-1
TO figurative-constant-2
All characters in data-name-3 equal to the single character figurative-constant-1 are replaced by the single character figurative-constant-2.

FROM
figurative-constant-1
TO
non-numeric-literal-2

All characters in data-name-3 equal to the single character figurative-constant-1 are replaced by the single character non-numeric-literal-2.

FROM
figurative-constant-1
TO
data-name-2

All characters in data-name-3 equal to the single character figurative-constant-1 are replaced by the single character in data-name-2.

FROM
non-numeric-literal-1
TO
figurative-constant-2

All characters in data-name-3 that are equal to any character in non-numeric-literal-1 are replaced by the single character figurative-constant-2.

FROM
non-numeric-literal-1
TO
non-numeric-literal-2

Non-numeric-literal-1 and non-numeric-literal-2 must be equal in length or non-numeric-literal-2 must be a single character. If equal in length, any character in data-name-3 equal to a character in non-numeric-literal-1 is replaced by the character in the corresponding position of non-numeric-literal-2.

If the length of non-numeric-literal-2 is one, all characters in data-name-3 that are equal to any character appearing in non-numeric-literal-1 are replaced by the single character given in non-numeric-literal-2.

FROM
non-numeric-literal-1
TO
data-name-2

Non-numeric-literal-1 and data-name-2 must be equal in length or data-name-2 must be a single-character item.

If equal in length, any character in data-name-3 equal to a character in non-numeric-literal-1 is replaced by the character in the corresponding position of data-name-2.

If the length of data-name-2 is one, all characters in data-name-3 that are equal to any character appearing in non-numeric-literal-1 are replaced by the single character given in data-name-2.

FROM
data-name-1
TO
figurative-constant-2

All characters in data-name-3 that are equal to any character in data-name-1 are replaced by the single character figurative-constant-2.

FROM
data-name-1
TO
non-numeric-literal-2

Data-name-1 and non-numeric-literal-2 must be of equal length or non-numeric-literal-2 must be one character.

If equal in length, any character in data-name-3 equal to a character in data-name-1 is replaced by the character in the corresponding position of non-numeric-literal-2.

If the length of non-numeric-literal-2 is one, all characters in data-name-3 that are equal to any character appearing in data-name-1 are

replaced by the single character given in non-numeric-literal-2.

FROM Any character in data-name-3 equal to a character in data-name-1 is replaced by the character in the corresponding position of data-name-2.
data-name-1
TO These items can be one or more characters, but must be equal in length.
data-name-2

The following rules pertain to the operands of the FROM and TO options:

1. The non-numeric-literals require enclosing quotation marks, as specified in the section, "Literals."
2. Data-name-1 and data-name-2 must be elementary alphabetic, or alphanumeric items, or fixed length group items less than 257 characters in length.
3. A character may not be repeated in non-numeric-literal-1 or in the area defined by data-name-1. If a character is repeated the results will be unpredictable.
4. The allowable figurative-constants are: ZERO, ZEROS, ZEROES, SPACE, SPACES, QUOTE, QUOTES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, and LOW-VALUES.

When either data-name-1 or data-name-2 appear as a determinant of the transformation rule, the user can change the transformation rule during object time.

Figure 24 contains examples of data-name-3 results, using the figurative-constant-1 to figurative-constant-2, non-numeric-literal-1 to non-numeric-literal-2, and data-name-1 to data-name-2 combinations, respectively. (The small b represents a blank.)

Data-name-3 Before	FROM	TO	Data-name-3 After
1b7bbABC	SPACE	QUOTE	1'7' 'ABC
1b7bbABC	'17CB'	'QRST'	QbRbbATS
1b7bbABC	b17ABC	CBA71b	BCACC71b
1234WXY89	98YXW4321	ABCDEFGHI	IHFEDCBA

Figure 24. Examples of Data Transformation

ARITHMETIC STATEMENTS AND OPTIONS

The following rules apply to the arithmetic statements:

1. All data-names used in arithmetic statements must represent elementary numeric data items that are defined in the Data Division of the program, except when they are the operands of the GIVING option.
Operands of the GIVING option can be either elementary numeric or report.
2. The maximum size of any data-name or literal is 18 decimal digits.
3. Intermediate result fields generated for the evaluation of fixed-point arithmetic expressions assure the accuracy of the result field, except where high order truncation is necessary.
4. Decimal point alignment is supplied automatically throughout computations.

The **ROUNDED** and **SIZE ERROR** options apply to all the arithmetic statements. The **GIVING** option applies to all arithmetic statements but **COMPUTE**.

OPTIONS:

GIVING Option: If the **GIVING** option is written, the value of the data-name that follows the word **GIVING** will be made equal to the calculated result of the arithmetic operation. The data-name that follows **GIVING** is not used in the computation and may contain editing symbols.

If the **GIVING** option is not written, the operand following the words **TO**, **FROM**, **BY**, and **INTO** in the **ADD**, **SUBTRACT**, **MULTIPLY**, and **DIVIDE** statements, respectively, must be a data-name. This data-name is used in the computation and is made equal to the result.

ROUNDED Option: If, after decimal-point alignment, the number of places in the calculated result are greater than the number of places associated with the data-name whose value is to be set equal to the calculated result, truncation occurs unless the **ROUNDED** option has been specified.

When the **ROUNDED** option is specified, the least significant digit of the resultant data-name has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

Rounding of a computed negative result is performed by rounding the absolute value of the computed result and then making the final result negative (unless the final result is zero).

Figure 25 illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result.

Item to Receive Calculated Result			
Calculated Result	PICTURE	Value After Rounding	Value After Truncating
12.36	99V9	12.4	12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	99V	66	65
.0055	V999	.006	.005

Figure 25. Rounding or Truncation of Calculations

SIZE ERROR Option: Whenever the number of integral places in the calculated result exceeds the number of integral places specified for the resultant data-name, a size error condition arises.

If the **SIZE ERROR** option has been specified and a size error condition arises, the value of the resultant data-name is not altered and the series of imperative statements specified for the condition is executed.

If the **SIZE ERROR** option has not been specified and a size error condition arises, no assumption should be made about the final result; but the program flow is not interrupted.

It should be noted that the **SIZE ERROR** option applies only to final calculated results. When a size error occurs in the handling of

intermediate results, no assumption should be made about the final result.

An arithmetic statement, if written with a SIZE ERROR option, is not an imperative statement. Rather, it is a conditional statement and is prohibited in contexts where only imperative statements are allowed.

Refer to Appendix D for a discussion on significant positions retained in arithmetics.

ADD Statement

The ADD statement adds two or more numeric values and substitutes the resulting sum for the current value of an item. The ADD statement has the following format:

$$\underline{\text{ADD}} \left\{ \begin{array}{l} \text{numeric-literal} \\ \text{floating-point-literal} \\ \text{data-name-1} \end{array} \right\} \dots \left\{ \begin{array}{l} \text{TO} \\ \text{GIVING} \end{array} \right\} \text{data-name-n}$$

[ROUNDED] [ON SIZE ERROR imperative-statement...]

When the TO option is used, the values of all the data-names (including data-name-n) and literals in the statement are added, and the resulting sum replaces the value of data-name-n. At least two data-names and/or numeric literals must follow the word ADD when the GIVING option is written.

SUBTRACT Statement

The SUBTRACT statement subtracts one or a sum of two or more numeric data items from a specified item and sets the value of a data item equal to the difference.

The SUBTRACT statement has the following format:

$$\underline{\text{SUBTRACT}} \left\{ \begin{array}{l} \text{data-name-1} \\ \text{numeric-literal-1} \\ \text{floating-point-literal-1} \end{array} \right\} \dots$$
$$\underline{\text{FROM}} \left\{ \begin{array}{l} \text{data-name-m} \text{ [GIVING data-name-n]} \\ \text{numeric-literal-m} \text{ GIVING data-name-n} \\ \text{floating-point-literal-m} \text{ GIVING data-name-n} \end{array} \right\}$$

[ROUNDED] [ON SIZE ERROR imperative statement...]

The effect of the SUBTRACT statement is to add the values of all the operands that precede FROM and then to subtract the sum from the value of the item following FROM. A literal can follow FROM only when the GIVING option is specified.

MULTIPLY Statement

The MULTIPLY statement multiplies two numeric data items and sets the value of data-name-2 (unless data-name-3 is specified) equal to the product.

The format of the MULTIPLY statement is:

MULTIPLY { data-name-1
numeric-literal-1
floating-point-literal-1 }

BY { data-name-2 [GIVING data-name-3]
numeric-literal-2 GIVING data-name-3
floating-point-literal-2 GIVING data-name-3 }

[ROUNDED] [ON SIZE ERROR imperative statement...]

DIVIDE Statement

The DIVIDE statement divides one numeric data item into another and sets the value of data-name-2 (unless data-name-3 is specified) equal to the quotient.

The format of a DIVIDE statement is:

DIVIDE { data-name-1
numeric-literal-1
floating-point-literal-1 }

INTO { data-name-2 [GIVING data-name-3]
numeric-literal-2 GIVING data-name-3
floating-point-literal-2 GIVING data-name-3 }

[ROUNDED] [ON SIZE ERROR imperative statement...]

Division by zero results in a SIZE ERROR condition.

COMPUTE Statement

The COMPUTE statement assigns to a data item the value of a numeric data item, literal, or arithmetic expression. The format of a COMPUTE statement is:

COMPUTE data-name-1 [ROUNDED] = { data-name-2
numeric-literal
floating-point-literal
arithmetic-expression }

[ON SIZE ERROR imperative statement...]

The data-name, specified to the left of the equal sign, must be an elementary report, binary, internal decimal, external decimal, internal floating-point, or external floating-point item.

The ON SIZE ERROR option applies only to the final result and not to any of the intermediate results.

Example: COMPUTE ANNUAL-PREMIUM = AGE * RATE * YEAR + BASE.

Arithmetic Expressions

An arithmetic expression consists of arithmetic operators, data-names, and/or literals representing items on which arithmetic may be performed.

The following five arithmetic operators may be used in arithmetic expressions:

Operator Operation

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Parentheses may be used to indicate the hierarchy of operations on elements in an arithmetic expression.

When the hierarchy of operations in an expression is not completely specified by parentheses, the order of operations is assumed to be exponentiation, then multiplication and division, and finally addition and subtraction. Thus, the expression $A + B / C + D ** E * F - G$ is taken to mean $A + (B / C) + ((D ** E) * F) - G$.

When the order of a sequence of consecutive operations on the same hierarchical level (i.e., consecutive multiplications and divisions or consecutive additions and subtractions) is not completely specified by parentheses, the order of operation is assumed to be from left to right. Thus, certain expressions ordinarily considered ambiguous are permitted in COBOL. For example, $A / B * C$ and $A / B / C$ are taken to mean $(A / B) * C$ and $(A / B) / C$, respectively. The expression $A * B / C * D$ is taken to mean $((A * B) / C) * D$. The expression $A ** B ** C$ is taken to mean $(A ** B) ** C$.

Exponentiation of a negative value is allowed only if the exponent is a literal or data-name having an integral value.

Exponentiation is performed in floating-point when an exponent is a fractional literal or is a data-name whose PICTURE describes a fractional number.

Plus and minus are allowable unary operators (having only one operand). The unary sign must be the first character of an arithmetic expression or must be immediately preceded by a left parenthesis. Two operators may not be adjacent to each other.

PROCEDURE BRANCHING STATEMENTS

In the GO TO, ALTER, and PERFORM statements, procedure-name signifies paragraph-name or section-name.

STOP Statement

The STOP statement is used to terminate or delay execution of the object program. The format of this statement is:

STOP { RUN
{ literal } }

The STOP RUN statement terminates execution of the object program and returns control to the operating system.

The STOP literal statement causes the specified literal to be displayed on the console and causes the object program to pause. The program may be resumed only by operator intervention. End of block must be keyed on the console to resume execution. The size of the literal is restricted to 72 characters.

GO TO Statement

The GO TO statement transfers control from one portion of the program to another. The GO TO statement has the following formats:

Option 1

GO TO [procedure-name]

Option 1 of the GO TO statement provides a means of transferring the path of flow of a program to a designated paragraph or section.

When Option 1 (unconditional GO TO) is used and a procedure-name is not specified, the GO TO statement must be preceded by a paragraph-name, must be the only statement in the paragraph, and must be modified by an ALTER statement prior to the first execution of the GO TO statement. The paragraph-name assigned to the GO TO statement is referred to by the ALTER statement in order to modify the sequence of the program. If procedure-name is omitted and the GO TO statement has not been preset by an ALTER statement prior to the first execution of the GO TO statement, execution of the program is terminated.

Option 2

GO TO procedure-name-1 [procedure-name-2...] DEPENDING ON data-name

In Option 2, data-name must be an elementary integral numeric item whose length does not exceed four digits and whose usage is either DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3.

Option 2 specifies alternative branch points; control is transferred to the point specified by the value of data-name. Control goes to the 1st, 2nd, ..., nth procedure-name as the value of data-name is 1, 2, ..., n. If data-name has a value outside the range 1 to n, no transfer takes place, and control passes to the next statement after the GO TO statement.

ALTER Statement

The ALTER statement is used to modify an unconditional GO TO statement elsewhere in the Procedure Division, thus changing the sequence in which program steps are to be executed.

The format of the ALTER statement is:

ALTER {procedure-name-1 TO PROCEED TO procedure-name-2}...

Procedure-name-1 designates a paragraph containing a single sentence consisting only of an Option 1 GO TO statement. The effect of an ALTER statement is to replace the procedure-name specified in Option 1 of the GO TO statement with procedure-name-2 of the ALTER statement, where the paragraph-name containing the GO TO statement is procedure-name-1 in the ALTER statement.

PERFORM Statement

The PERFORM statement specifies a transfer of control from one portion of a program to another, in order to execute some procedure a specified number of times, or until a condition is satisfied. It directs that control is to be returned to the statement immediately following the point from which the transfer was made.

The PERFORM statement has the following four formats:

Option 1

PERFORM procedure-name-1 [THRU procedure-name-2]

Option 1 is the simple PERFORM statement. A procedure referred to by this type of PERFORM statement is executed once, and then control passes to the next statement after the PERFORM statement. All statements in the paragraphs or sections named by procedure-name-1 through procedure-name-2 constitute the range of the PERFORM statement.

Option 2

PERFORM procedure-name-1 [THRU procedure-name-2]

{integer
data-name TIMES}

Option 2 is the TIMES option. When the TIMES option is used, the procedure is performed the number of times specified by data-name or integer. Control is then transferred to the statement following the PERFORM statement. Data-name must have an integral value and data-name or integer must have a positive non-floating point value, less than 32,768. If the value of the data-name is negative, zero, or greater than 32,767, control is passed immediately to the statement following the PERFORM statement.

Option 3

PERFORM procedure-name-1 [THRU procedure-name-2]

UNTIL test-condition

Option 3 is the UNTIL option. Test-condition may be simple or compound. The procedures specified by the procedure-names are performed until the condition specified by the UNTIL option is true. At this time, control is transferred to the statement following the PERFORM statement. If the condition specified by the UNTIL option is true at the time the PERFORM statement is encountered, the specified procedure is not executed.

Option 4

```
PERFORM procedure-name-1 [THRU procedure-name-2]  
VARYING data-name-1 FROM{numeric-literal-2}  
data-name-2  
BY{numeric-literal-3}  
data-name-3 UNTIL test-condition-1  
[AFTER data-name-4 FROM{numeric-literal-4}  
data-name-5  
BY{numeric-literal-6}  
data-name-6 UNTIL test-condition-2 ]  
[AFTER data-name-7 FROM{numeric-literal-8}  
data-name-8  
BY{numeric-literal-9}  
data-name-9 UNTIL test-condition-3 ]
```

Option 4 is the VARYING option. Test-condition may be simple or compound.

This option is used to augment the value of one or more data-names in an orderly fashion during the execution of a PERFORM statement. When one data-name is varied, data-name-1 is set equal to its starting value (FROM) when commencing the PERFORM statement. Then, test-condition-1 is evaluated: if it is true, control passes to the next statement following the PERFORM statement; if false, procedure-name-1 through procedure-name-2 is executed once. The value of the increment (BY) is added to data-name-1, and the condition (UNTIL) is evaluated again. The cycle continues until test-condition-1 is true, at which point control is passed to the statement following the PERFORM statement.

All data-names and literals used must represent non-floating point numeric values; they may be positive, negative, or zero.

Data-name-1, data-name-4, and data-name-7 must not be alternate names for the same data items. For all options, the first statement of procedure-name-1 is the point to which sequence control is transferred by the PERFORM statement.

When two data-names are varied, the value of data-name-4 goes through a complete cycle (FROM, BY, UNTIL) each time that data-name-1 is augmented with its BY value. For three data-names, the value of data-name-7 goes through a complete cycle (FROM, BY, UNTIL) each time that data-name-4 is augmented with its BY value, which in turn goes through a complete cycle each time data-name-1 is varied.

Regardless of the number of data-names being varied, as soon as test-condition-1 is found to be true, control is transferred to the next statement after the PERFORM statement.

The return of control is from a point determined as follows:

1. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, the return is made after the last statement of the procedure-name-1 paragraph.
2. If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is made after the last statement of the last paragraph of the procedure-name-1 section.
3. If procedure-name-2 is specified and is a paragraph-name, the return is made after the last statement of the procedure-name-2

paragraph.

4. If procedure-name-2 is specified and is a section-name, the return is made after the last statement of the last paragraph of the procedure-name-2 section.

GO TO statements and other PERFORM statements are permitted between procedure-name-1 and the last statement of procedure-name-2. Furthermore, the time sequence of execution of exits established by PERFORM statements must be in the inverse order in which they were established.

*** For Tape Operating System only, the exact range of a PERFORM statement must not be activated again while the range is currently active. An active PERFORM statement, whose execution point begins within the range of another PERFORM, must not contain the exit point of the other active PERFORM, except when the exit points are common. If the logic of a procedure requires a conditional exit prior to the final sentence, the EXIT sentence must be used. In this case, procedure-name-2 must be the name of the paragraph that consists solely of the EXIT sentence.

A procedure referred to by one PERFORM statement can be referred to by other PERFORM statements. Moreover, a procedure referred to by one or more PERFORM statements can also be executed by "dropping through," that is, by entering the procedure through the normal passage of control from one statement to the next, in sequence. Accordingly, if procedure-name-1 were the next statement following the PERFORM statement, the procedure would be executed once more than specified by the PERFORM statement because, after execution of the PERFORM statement, control would pass to procedure-name-1 in the normal continuation of the sequence.

Figures 26, 27, and 28 illustrate the logical flow of Option 4 PERFORM statements, varying one, two, and three data-names, respectively.

Figure 29 states restrictions on the appearance of procedure-branching statements. Y means that the statement may appear; N indicates that it must not; text indicates the outcome if the statement does appear.

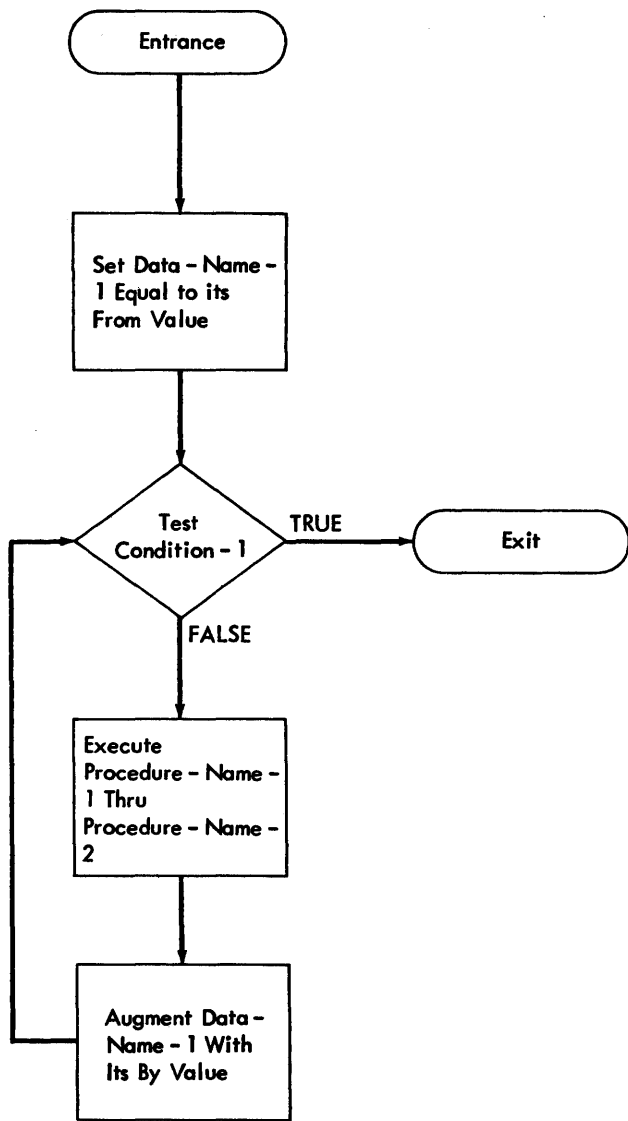


Figure 26. Logical Flow of Option 4 PERFORM Statement Varying One Data-name

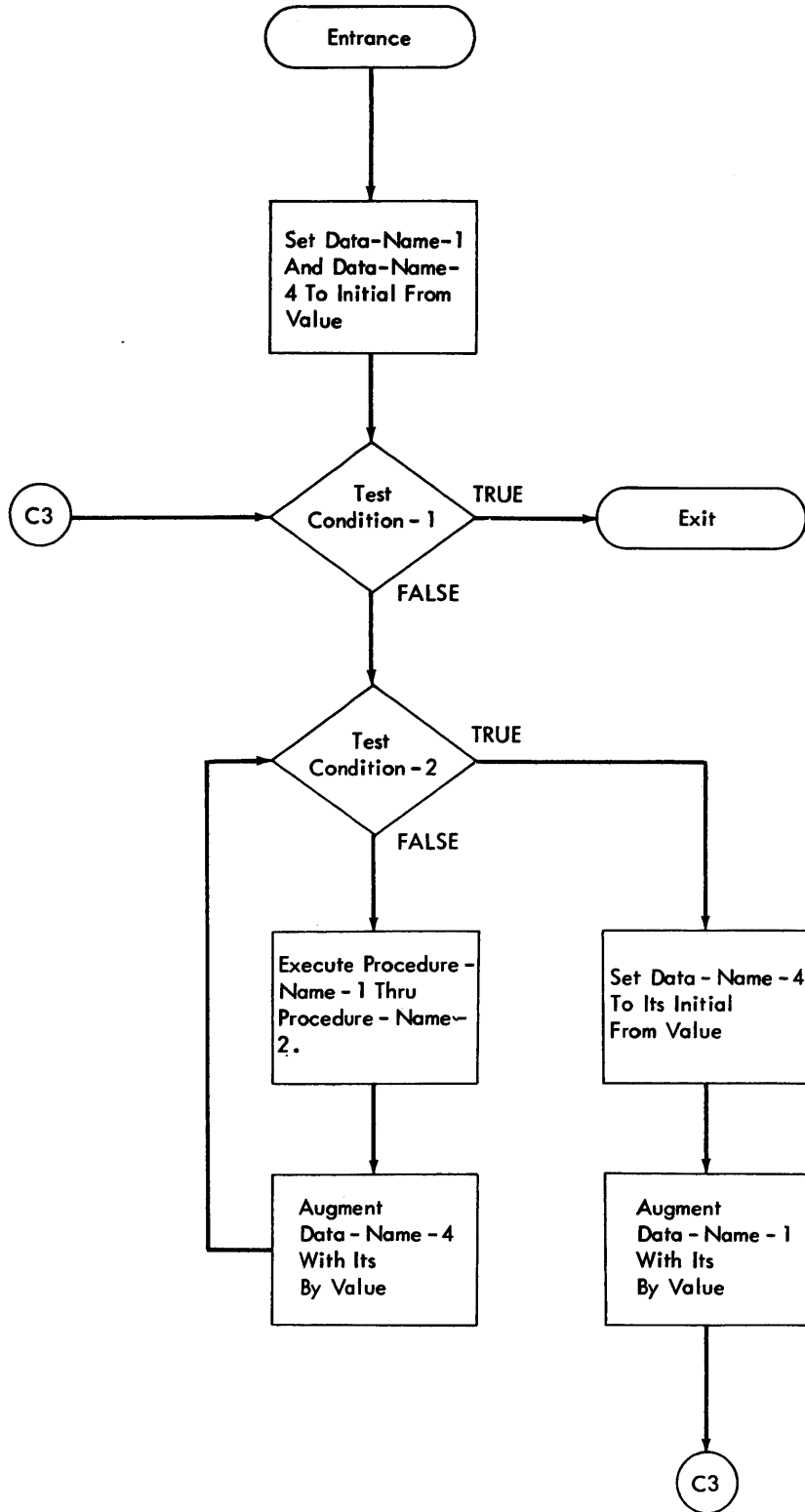


Figure 27. Logical Flow of Option 4 PERFORM Statement Varying Two Data-names

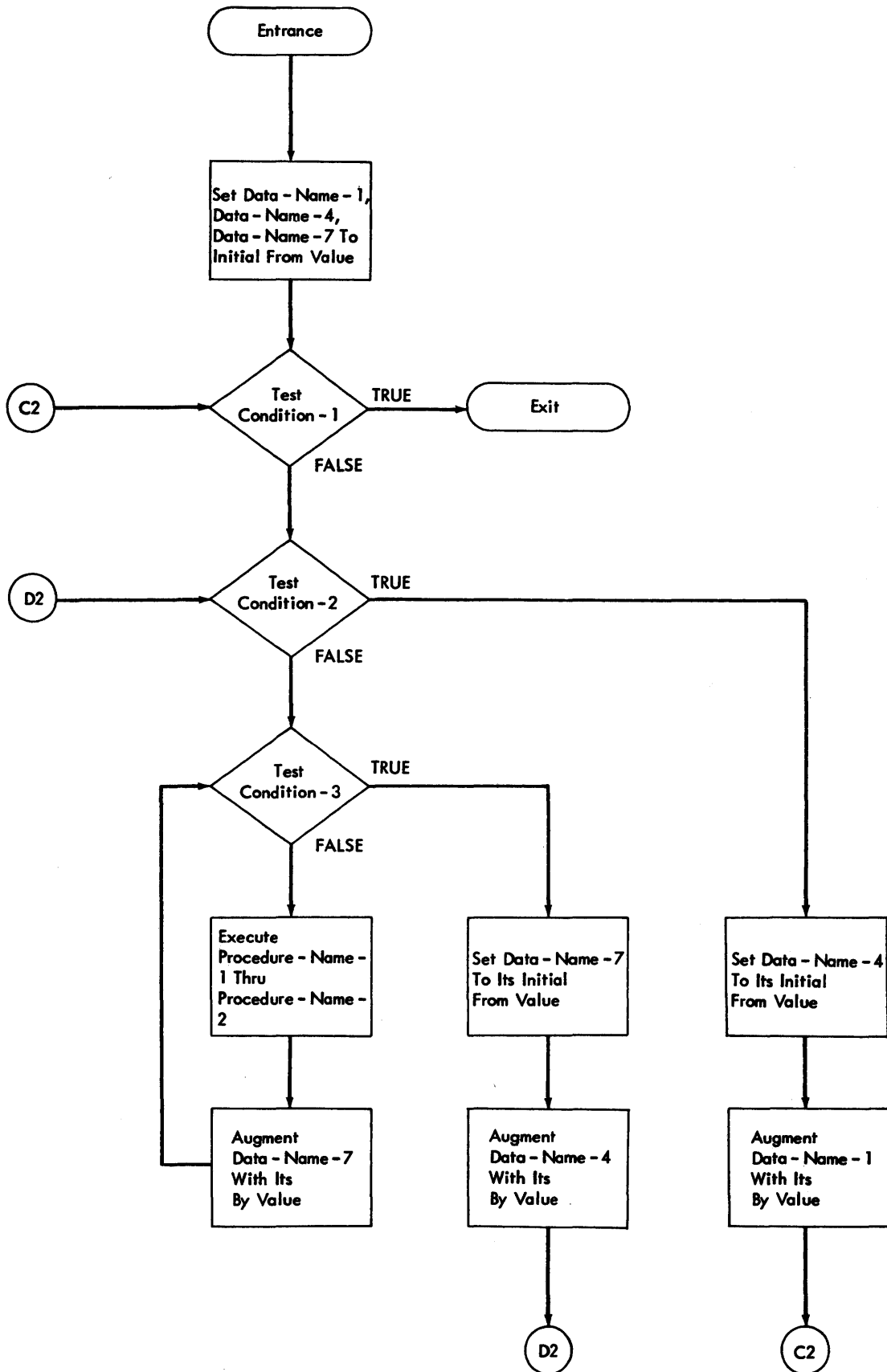


Figure 28. Logical Flow of Option 4 PERFORM Statement Varying Three Data-names

Statement	Appearing In:			
	Label Checking Declarative	Label Creating Declarative	Main Body of Procedure Division	Debug Packet
GO TO PERFORM ALTER	Y*	Y*	Y**	Y***
STOP RUN	N	N	end of execution	abnormal end of execution
STOP Literal	Y	Y	Y	Y

- * Operands of these statements must be procedure-names appearing in the declarative containing the statement.
- ** Operands of these statements must be procedure-names appearing in the main body of the Procedure Division.
- *** Operands of these statements may be procedure-names appearing either in the main body or in any debug packet.

Figure 29. Restrictions for Procedure-Branching Statements

COMPILER-DIRECTING STATEMENTS

Compiler-directing statements must be separate sentences.

ENTER Statement

The ENTER statement, used in conjunction with CALL or ENTRY statements, permits communication between a COBOL object program and COBOL subprograms or other language subprograms.

The ENTER statement has the following two formats:

Option 1 (Used in calling program)

ENTER LINKAGE.
CALL entry-name [USING argument...].
ENTER COBOL.

Option 2 (Used in a COBOL subprogram)

ENTER LINKAGE.
ENTRY entry-name [USING data-name...].
ENTER COBOL.

subprogram statements

ENTER LINKAGE.
RETURN.
ENTER COBOL.

An example of the use of this format is:

COBOL PROGRAM SHEET

SEQUENCE	A	B
1	6 7 8	12
001	020	PROCEDURE DIVISION. . .
002	001	ENTER LINKAGE
002	002	CALL 'SUBPRGM' USING RECORD-2.
002	003	ENTER COBOL.

Refer to Appendix E, Figure 34, for the relationship between the example above, and the sample program given therein.

Entry-name is an external name and must follow the rules for external name formation.

Option 1 is used to effect transfer of control to a subprogram. Entry-name represents the name of the subprogram's entry point.

In the USING option, an argument may be one of the following:

1. A data-name when calling a COBOL subprogram
2. A data-name, file-name, or a procedure-name when calling a subprogram written in a language other than COBOL.

Option 2 is used to establish an entry point in a COBOL subprogram. Control is transferred to the entry point by a CALL statement in another program. Entry-name defines the entry point where parameters are saved for eventual return and address parameters are obtained.

Entry-name must not be the same as the Program-ID.

Each data-name in the USING portion of the ENTRY statement must be defined in the Linkage Section of the Data Division, and must have level number 01 or 77.

Computer base addresses of data items named in the USING list of an ENTRY statement are obtained from the USING list of the associated CALL statement. Names in the two USING lists (that of the CALL in the main program, and that of the ENTRY in the subprogram) are paired in one-to-one correspondence.

There is no necessary relationship between the actual names used for such paired names, but the data descriptions must be equivalent. When a group data item is named in the USING list of an ENTRY statement, names subordinate to it in the subprogram's Linkage Section may be employed in subsequent subprogram procedural statements, when elementary items in the group are utilized.

RETURN enables restoration of the necessary registers saved at an entry point. The return from a subprogram is always to the first instruction following the last instruction in the calling sequence of the main program.

There must be no path of program flow to an ENTRY statement within the program containing the ENTRY statement. Hence, the statement should not have a paragraph-name.

EXIT Statement

The EXIT statement may be used when it is necessary to provide an end point for a procedure that is to be executed by means of a PERFORM statement or for a procedure that is a declarative.

The format for the EXIT statement is:

paragraph-name. EXIT.

EXIT must appear in the source program as a one-word paragraph preceded by a paragraph-name.

When the PERFORM statement is used, an EXIT paragraph-name may be the procedure-name given as the object of the THRU option. In this case, a statement in the range of a PERFORM being executed may transfer to an EXIT paragraph, bypassing the remainder of the statements in the PERFORM range. In all other cases, EXIT paragraphs have no function and control passes sequentially through them to the first sentence of the next paragraph.

NOTE Statement

The NOTE statement permits the programmer to write explanatory comments, in the Procedure Division of a source program, which will be produced on the listing but serve no other purpose. The format of the NOTE statement is:

NOTE comment...

NOTE, when used, must begin a sentence. Following the word NOTE, any combination of the characters from the COBOL character set may appear. If NOTE is the first word of a paragraph, any remaining sentences within the paragraph are also considered notes. Proper format rules for paragraph structure must be observed.

Prewritten source program entries can be included in a COBOL program at compile time. Thus, an installation can utilize standard file descriptions, record descriptions, or procedures without having to repeat programming them. These entries and procedures are contained in a user-created library. They are included in a source program by means of a COPY clause or an INCLUDE statement.

COPY CLAUSE

The COPY clause permits the user to include prewritten Data Division entries or Environment Division clauses in his source program. The COPY clause is written in one of the following forms:

(option 1
(Within the Input-Output Section)

{FILE-CONTROL.} COPY library-name.
{I-O CONTROL.}

Option 2
(Within the File-Control Paragraph)
SELECT file-name COPY library-name.

Option 3
(within a file area description entry or within the Working Storage or Linkage section)
01 data-name COPY library-name.

Option 4
(Within the Working Storage or Linkage Section)
77 data-name COPY library-name.

Option 5
(Within the file section)
FD file-name COPY library-name.

Library-name is contained in the user's library and identifies the entries to be copied. It is an external-name and must follow the rules for external-name formation.

The words preceding COPY library-name must follow the rules for COBOL margination. On a given source program card containing the completion of a COPY clause, there must be no information beyond the clause-terminating period. The material introduced into the source program by the COPY statement will follow the COPY statement on the listing, beginning on the next line.

No COPY clause may be contained in the information copied from the library.

When options 1, 2, or 5 are written, the words COPY library-name are replaced by the information identified by library-name. This information comprises the sentences or clauses needed to complete the paragraph, sentence, or entry containing the COPY clause.

When options 3 and 4 are written, the entire entry is replaced by the information identified by library-name except that the data-name specified, and replaces the corresponding data-name in the library and any references to the corresponding data-name in the information copied. This information comprises a 01 or 77 level entry and any immediately subsequent entries with level numbers higher than 01 or 77.

The data-name replacement is for the compilation, but is not shown on the listing.

A COPY clause may be preceded by other information on a source program card, and may be written on more than one card; however on a given card, containing the completion of a COPY clause, there must be no information beyond the clause-terminating period. The material introduced into the source program by the COPY statement will follow the COPY statement on the listing, beginning on the next line.

INCLUDE STATEMENT

The INCLUDE statement permits the user to include prewritten procedures in the Procedure Division of his source program. The INCLUDE statement has the following formats:

Option 1 (For insertion of a paragraph):
paragraph-name. INCLUDE library-name.

Option 2 (For insertion of a section):
section-name SECTION. INCLUDE library-name.

Library-name is contained in the user's library and identifies the entries to be copied. It is an external name and must follow the rules for external name formation.

The words preceding INCLUDE library-name must follow the rules for COBOL margination. On a given source program card, containing the completion of an INCLUDE statement, there must be no information beyond the clause-terminating period. The material from the library will follow the INCLUDE statement on the listing.

When the INCLUDE statement is written, the words INCLUDE library-name are replaced by the information identified by library-name. This information comprises the paragraphs or sentences needed to complete the section or paragraph containing the INCLUDE statement.

The library entries for paragraphs and sections must not contain INCLUDE statements.

Refer to IBM DOS and TOS/360 System Control and System Service Programs for a description of library facilities.

(text deleted)

STERLING CURRENCY FEATURE

Disk and Tape Operating Systems COBOL provide facilities for handling sterling currency items by means of an expansion of the PICTURE clause. Additional options and formats, necessitated by the non-decimal nature of sterling, and by the conventions by which sterling amounts are represented in punched cards, are also available.

Sterling amounts are normally expressed in pounds, shillings and pence, in that order. There are twenty shillings in a pound, and twelve pence in a shilling. Though sterling amounts are sometimes expressed in shillings and pence only (in which case the number of shillings may exceed 99), within machine systems shillings will always be expressed as a two-digit field. Pence, when in the form of integers, likewise will be expressed as a two-digit field. However, provision must be made for pence to be expressed as decimal fractions as well, as in the form 17s. 10.237d.

The IBM method for representing sterling amounts in punched cards uses two columns for shillings and one for pence. Tenpence (10d.) is represented by an '11' punch and elevenpence (11d.) by a '12' punch. The British Standards Institution (B.S.I.) representation uses single columns for both shillings and pence. B.S.I. pence representation for tenpence and elevenpence is the reverse of that of IBM: an '11' punch is used for 11d. and a '12' punch for 10d. B.S.I. representation for shillings consists of a '12' punch for 10 shillings and double punches A to I for eleven to nineteen shillings.

The indicated representations may be used separately or in combination, resulting in four possible conventions.

1. IBM pence and IBM shillings
2. IBM pence and B.S.I. shillings
3. B.S.I. pence and B.S.I. shillings
4. B.S.I. pence and IBM shillings

Any of these conventions may be associated with any number of digits, or no digits, in the pound field; and any number of decimals, or no decimals, of pence. In addition, sign representation may be present as an overpunch in one of several allowable positions in the amount, or may be separately entered from another field.

Two formats are provided by Disk and Tape Operating Systems COBOL in the PICTURE clause for the representation of sterling amounts, Sterling Non-Report format and Sterling Report format. In the formats that follow, n stands for a positive nonzero integer. When such an integer is used, it must be parenthesized. The characters 6 7 8 9 D * , £ / B Z V . : s d CR - are the PICTURE characters used to describe sterling items.

Throughout the following text, parentheses are used to denote multiple successive occurrences of a picture character. With the exception of the pound sign , there is no difference between the parenthesized form and multiple successive occurrences of a picture character.

STERLING NON-REPORT

The format for Sterling non-report is:

USAGE IS DISPLAY-ST

PICTURE IS 9[(n)]D[8]8D $\left\{ \begin{array}{l} 6[6] \\ 7[7] \end{array} \right\} [[v]9(n)]$

The representation for pounds is 9(n)D where:

1. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.
2. The character D indicates the position of an assumed pound separator.
3. An entry must always appear in the pound field for a picture to be valid.

The representation for shillings is [8]8D where:

1. The characters [8] 8 indicate the position of the shilling field, and the convention by which shillings are represented in punched cards. 88 indicates IBM shilling representation occupying a two column field. 8 indicates B.S.I. single column shilling representation. An entry must always appear in the shilling field for a picture to be valid.
2. The character D indicates the position of an assumed shilling separator.

The representation for pence is 6[6] [[V]9(n)]
7[7]

1. The character 6 indicates IBM single column pence representation wherein 10d. is represented by an '11' punch and 11d. by a '12' punch. The characters 66 indicate two column representation of pence, usually from some external medium other than punched cards.
2. The character 7 indicates B.S.I. single column pence representation wherein 10d. is represented by a '12' punch and 11d. by an '11' punch. The characters 77 indicate two column representation of pence. Consequently, 66 and 77 serve the same purpose and are interchangeable. An entry must always appear in the pence field for a picture to be valid.
3. The character V indicates the position of an assumed decimal point in the pence field. Its properties and use are identical with that of V in dollar amounts. Decimal positions in the pence field may extend to n positions.

STERLING SIGN REPRESENTATION

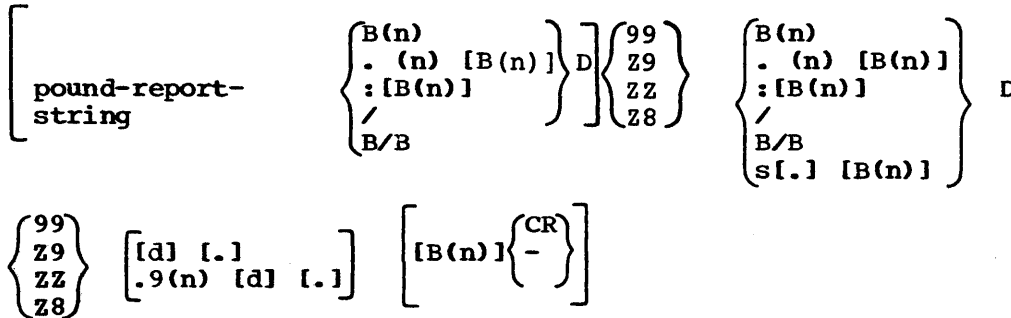
Signs for sterling amounts may be entered as overpunches in one of several allowable positions of the amount. A sign is indicated by an embedded S in the non-report picture immediately to the left of the position containing the overpunch. Allowable overpunch positions are the high order and low order positions of the pound field, the high order shilling digit in two column shilling representation, the low order pence digit in two column pence representation, or the least significant decimal position of pence. Examples of such a picture are:

STERLING REPORT

The format for Sterling report is:

USAGE IS DISPLAY-ST

PICTURE IS



The picture for STERLING REPORT is composed of a sequence of characters representing the fields for, respectively, (i)pounds (ii) pound separators (iii) shillings (iv) shilling separators (v) pence integers (vi) pence decimal fractions and pence terminators. The USAGE IS DISPLAY-ST clause is necessary to enable the compiler to distinguish between sterling and decimal data in cases where their formats may be the same.

1. a. Pound-report-string for the representation of pounds is similar to the report-form option for decimal fields. The editing characters that may be combined to describe a pound report item are: 9 Z * , B £ -. With the exception of the pound sign (£) the editing characters have the same meaning in pound-report-string as the report form for decimal fields.

With one exception, the pound sign may be equated to the dollar sign in terms of function and manner of use. Specifically, the pound sign may be used as a floating string character and, as with the dollar sign, may be floated through Bs and/or commas, should they be embedded in the floating string. Examples of such strings are:

£,£££,£99

££B££9

- b. The exception to equating the pound and dollar signs consists of an option available in pound-report-strings. With this option the floating pound sign may be suppressed when the value of the pound field is zero. Such suppression is the pound-report-string equivalent of the BLANK WHEN ZERO clause. The floating pound string may be described in either of two formats:

<u>Format</u>	<u>Values of pounds field</u>	<u>Output</u>
(1) £ . . . £	1 0	b£1 bb
(2) £ (n)	1 0	b£1 bbb

The use of parenthesis to indicate multiplicity of pound signs in a picture, as in format 2, indicates the pound sign suppression option is desired.

c. The single character £ indicates the position containing a fixed pound sign.

2. The representation for pound separators is:

$$\left. \begin{array}{l} B(n) \\ .(n) [B(n)] \\ : [B(n)] \\ / \\ B/B \end{array} \right\} D$$

- The characters B(n) specify n character positions in which blanks will be inserted. B or multiples of B may not be used alone as pound separators.
- The characters :[B(n)] specify a colon which may be followed by n character positions containing blanks.
- The characters .(n) [B(n)] specify n character positions containing periods which may be followed by n character positions containing blanks.
- The character / may stand alone as a separator, or it may be preceded by one blank and followed by another.
- The character D must be used. Its use is internal only and enables the compiler to properly align pound and shilling fields.
- In cases where the pound sign suppression option has been specified, and the pound field is equal to zero, suppression will be extended to include the pound separator. Stated generally, under the pound sign suppression option the suppression of all digits to the left of a separator will result in the suppression of the separator as well. Consequently, an amount equal to zero will result in output consisting only of spaces.

3. The representation for shillings is:

$$\left. \begin{array}{l} 99 \\ ZZ \\ Z8 \\ Z9 \end{array} \right\}$$

The character 8 is identical to Z when all digits to the left, including those in the pound field, have been suppressed, and is identical to 9 if they have not been.

4. The representation for shilling separators is:

$$\left. \begin{array}{l} B(n) \\ .(n) [B(n)] \\ : [B(n)] \\ / \\ B/B \\ s[.] [B(n)] \end{array} \right\} D$$

- Those characters which are used also as pound separators are discussed under that heading.

- b. The characters s or s. may stand alone as separators, followed immediately by the high order pence digit, or they may be followed by n spaces, when written in the formats sB(n) and s.B(n).
 - c. The character D must be used. Its use is internal only and enables the compiler to properly align pounds and shillings.
5. The representation for pence integers is:

$$\left. \begin{array}{l} 99 \\ Z9 \\ ZZ \\ Z8 \end{array} \right\}$$

- a. The character 8 is identical to Z when all digits to the left, including those in the pound and shilling fields, have been suppressed, and is identical to 9 when they have not been. The remaining characters have been explained under other headings. If there are no positions of pence decimals, the pence terminator follows immediately. If there are decimals, the terminators follow the low order decimal position.

6. The representation for pence decimal fractions and terminators is:

$$\left[\begin{array}{l} [d] [.] \\ .9 (n) [d] [.] \end{array} \right] \left[B(n) \left\{ \begin{array}{l} CR \\ - \end{array} \right\} \right]$$

- a. The upper set of characters d and . represents pence terminators which may be used if there is no pence decimal fraction. The characters 9(n) indicate the number of positions the pence decimal fraction will occupy, should there be one. The second set of characters d and. indicates that these pence terminators may also be used following a pence decimal fraction.
- b. The characters B(n) CR and - indicate terminators which may follow those discussed above. The characters CR or - may append to an amount to indicate a debit and may immediately follow a previous pence terminator or low order pence digit, or they may be preceded by n spaces. The + sign is not used in sterling amounts.

The user may use sterling non-report items as operands in connection with other numeric operands in MOVE, ADD, and SUBTRACT statements only.

Decimal items moved to sterling report and sterling non-report are considered as pence.

Some examples of the sterling report picture are:

```

ZZBZ(3)/DZ8/D99
99S.BD99d.B-
££, £(3), £(3).DZ8..DZ9.
*(7)D99D99.99.CR

```

INTERNATIONAL CONSIDERATIONS

1. Installations may interchange the function of the comma and decimal point characters in numeric literals and the PICTURE clause.
2. Installations may alter the compiler's character set for non-English language requirements, so that, for example, data-names may be composed of letters of a particular national alphabet.
3. The PICTURE of report items may terminate with the currency symbol in cases where the graphic \$ is supplanted by a particular national currency symbol.
4. Sentences may be substituted to allow translation (by modification) of output messages into any non-English language.

The following statements are provided for program debugging. They may appear anywhere in a DOS and TOS/360 COBOL program or in a compile-time debugging packet.

TRACE

The format of the TRACE statement is:

$\left. \begin{array}{l} \text{READY} \\ \text{RESET} \end{array} \right\} \text{TRACE}$

After a READY TRACE statement is executed, each time execution of a paragraph or section begins, a message is written of arrival at such a point. The message is written on the system logical printing device (SYSLST).

The execution of a RESET TRACE statement terminates the functions of a previous READY TRACE statement.

EXHIBIT

The format of the EXHIBIT statement is:

$\text{EXHIBIT} \left\{ \begin{array}{l} \text{NAMED} \\ \text{CHANGED NAMED} \\ \text{CHANGED} \end{array} \right\} \left\{ \begin{array}{l} \text{data-name} \\ \text{non-numeric-literal} \end{array} \right\} \dots$

The execution of an EXHIBIT NAMED statement causes a formatted display of the data-names (or non-numeric literals) listed in the statement. The system logical printing device (SYSLST) is used. The format of the output for each data-name listed in the NAMED or CHANGED NAMED form of an EXHIBIT statement is:

blank
original data-name (including qualifiers, if written)
blank
equal sign
blank
value of data-name

Literals listed in the statement are preceded by a blank, when displayed.

The sum of the sizes of the operands of an EXHIBIT statement may not exceed the maximum logical record length for the system logical printing device (SYSLST).

The EXHIBIT NAMED option statement is exhibited with up to four data names and their data per print line. The sum of the size of the operands of each group of data names cannot exceed the maximum logical record length for the system logical printing device (SYSLST).

Each EXHIBIT statement must be the last statement in a sentence.

The CHANGED form of the EXHIBIT statement provides for a display of items when they change value, compared to the value at the previous time the EXHIBIT CHANGED statement was executed. The initial time such a statement is executed, all values are considered changed; they are displayed and saved for purposes of comparison.

Note that, if two distinct EXHIBIT CHANGED data-name statements appear in a program, changes in data-name are associated with the two separate statements. Depending on the path of program flow, the values of data-name saved for comparison may differ for the two statements.

Only one data-name may be listed in an EXHIBIT CHANGED statement.

The CHANGED NAMED form of the EXHIBIT statement causes a printout of each changed value for items listed in the statement. Only those values representing changes and their identifying names are printed. A fixed columnar format for the data to be displayed cannot be created with EXHIBIT CHANGED NAMED.

ON (COUNT-CONDITIONAL STATEMENT)

The format of the ON statement is:

ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]

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

[ELSE statement ...]
[OTHERWISE NEXT SENTENCE]
```

The ON statement is a conditional statement. It specifies when the statements it contains are to be executed. ELSE (OR OTHERWISE) NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

The count-condition (integer-1 AND EVERY integer-2 UNTIL integer-3) is evaluated as follows:

Each ON statement has a compiler-generated counter associated with it. The counter is initialized in the object program with a value of zero.

Each time the path of program flow reaches the ON statement, the counter is advanced by 1. Where K is any positive integer, if the value of the counter is equal to integer-1 + (K*integer-2), but is less than integer-3 if specified, the imperative statements (or NEXT SENTENCE) are executed. Otherwise, the statements after ELSE (or NEXT SENTENCE) are executed. If the ELSE option does not appear, the next sentence is executed.

If integer-2 is not given, it is assumed that integer-2 has a value of 1. If integer-3 is not given, no upper limit is assumed for it.

If neither integer-2 nor integer-3 is specified, the imperative statements are executed only once.

Examples:

ON 2 AND EVERY 2 UNTIL 10 DISPLAY A ELSE DISPLAY B.

On the second, fourth, sixth, and eighth times, A is displayed. B is displayed at all other times.

ON 3 DISPLAY A.

On the third time through the count-conditional statement, A is displayed. No action is taken at any other time.

COMPILE-TIME DEBUGGING PACKET

Debugging statements for a given paragraph or section in a program may be grouped together into a debugging packet. These statements will be compiled with the source language program, and will be executed at object time. Each packet refers to a specified paragraph-name or section-name in the Procedure Division. Compile-time debugging packets are grouped together and are placed immediately preceding the source program.

Each compile-time debug packet is headed by the control card *DEBUG. The general form of this card is

1 8
*DEBUG location

where the parameters are described as follows:

Location is the COBOL section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the packet is to be executed. Effectively, the statements in the packet are executed as if they were physically placed in the source program following the section-name or paragraph-name, but preceding the text associated with the name. The same location must not be used in more than one *DEBUG control card.

Location cannot be a paragraph name within any DEBUG packet.

A debug packet may consist of any procedural statements conforming to the requirements of Disk and Tape Operating Systems COBOL. A GO TO, PERFORM, or ALTER statement in a debug packet may refer to a procedure-name in any debug packet or in the main body of the Procedure Division.

In this appendix, definitions are provided for certain elements of the COBOL language. These definitions are presented in a uniform system of notation, explained in the following paragraphs. This notation is useful in describing the COBOL language, although it is not part of COBOL. All definitions presented in this notation are syntactical definitions. They define the structure, rather than the meaning, of the defined element.

1. All words printed entirely in capital letters are reserved words. These are words that have preassigned meanings in the COBOL language. Words in capital letters represent an actual occurrence of those words.

Example: ADD

When this is specified, the letters ADD are indicated.

2. All punctuation and special characters (except those symbols described in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted, according to the rules for punctuation specified in this publication.
3. Lower-case words that appear in a definition are themselves defined under the proper entry.

Example: integer

This specifies the appearance of an item of the class "integer."

4. | (the or sign) The or sign indicates a choice.

Example: +|-

This specifies a plus sign or a minus sign.

5. :=(the defined-as symbol) This symbol means "defined as."

Example: digit ::=1|2|3|4|5|6|7|8|9|0

6. ... (the ellipsis) The ellipsis indicates that the immediately preceding unit may occur one or more times in succession. A unit, in this and succeeding paragraphs, means either a single reserved word, a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If an item is enclosed in brackets or braces, the entire unit of which it is a part must be repeated if repetition is specified.

Example: unsigned-integer ::=digit. . .

This defines an unsigned integer as any number of sequential digits.

7. [] (brackets) Brackets are used to indicate that an item is optional. The programmer may use the item or not, depending on the requirements of his program.

Example: non-negative-integer ::= [+] digit. . .

This defines a non-negative integer as a series of digits optionally preceded by a plus sign.

8. When brackets surround items separated by the or sign, any one of the items enclosed, or none of them, may be chosen.

Example: `integral-number ::= [+|-] digit . . .`

This specifies that an integral number is a series of digits preceded by a minus sign, a plus sign, or neither.

9. { } (braces) When braces surround items separated by the or sign, one of the items shown must be chosen.

Example: `signed-integer ::= {+|-}digit . . .`

This defines a signed integer as a series of digits that must be preceded by either a plus sign or a minus sign.

10. Braces may also be used to indicate that the enclosed items are to be treated as a unit.

Example: `[letter] {digit letter} . . .`

This specifies a sequence of any length consisting of digits alternating with letters.

11. not (the not symbol) This symbol indicates that the unit following it may not occur.

Example: `digit not 0`

This specifies any member of the class digit except 0. Combined with the definition of digit used above, it specifies the equivalent of

`1|2|3|4|5|6|7|8|9`

12. min (the minimum symbol) This indicates the minimum number of times that a unit may occur. When it is used without an accompanying maximum symbol (defined below), the implied maximum of times the unit may occur is infinity.

Example: `min 3 digit`

This specifies a sequence of no less than three digits. There is no upper limit on the length of this sequence.

Example: `min 5{digit|letter}`

This specifies a sequence of at least five letters and digits, in any order.

13. max (the maximum symbol) This specifies the maximum number of times that the following unit may appear in succession. When max appears without an associated min, a minimum of zero occurrences is assumed.

Example: `max 18 digit`

This specifies that no digits, one digit, or a sequence of up to 18 digits may be indicated at this point by the programmer.

Example: `min 2 max 6{digit|letter}`

This specifies that digits and letters, intermixed in any succession, must occur in a sequence at least two long, or at most six long.

Example: min 7 max 7 digit

This specifies a sequence of exactly seven digits.

14. blank This symbol represents the occurrence of a space.

Example: {.|blank}

This specifies that either a period or a space must occur at this point.

15. Further restrictions and explanations will be found in the text.
16. Certain special restrictions and definitions that apply only to the Sterling Currency Feature are not included in this glossary. They are discussed in Section 8.

alphanumeric-literal::= 'Min 1 max 120{ letter|blank }'

alpha-form::= min 1 max 30 A[(integer)]

The sum of the value of integer plus the number of As must not exceed 30.

alphanumeric-literal::= 'min 1 max 120{character not ' }'

an-form::= min 1 max 30 X[(integer)]

The sum of the value of integer plus the number of Xs must not exceed 30.

argument::= data-name|file-name|procedure-name

arithmetic expression::= [() {{{numeric-literal|floating-point-literal|data-name|arithmetic-expression}arithmetic-operator}+|-}{numeric-literal|floating-point-literal|data-name|arithmetic-expression} []]

The use of the) is obligatory if the (has been used. Data-names used in arithmetic-expressions must refer to numeric data.

arithmetic-operator::= +|-|*|/|**

character::= COBOL-character|
{EBCDIC-character not COBOL-character}

clause::= Individual clauses are described in appropriate sections of the text.

COBOL-character::= letter |digit|+|-|/|*|=|\$|,|
()|.|;|<|>|blank

comment::= COBOL-character . . .{ . blank}

compound-condition::= [NOT]{test-condition{AND/OR}
test-condition}

compiler-directing-statement::=See "Compiler-Directing Statements"
in text.

condition::=	test-condition event-condition
condition-name::=	data-name
conditional-statement::=	See "Conditional Statements" in text.
data-name::=	{{{digit letter}max n{digit - letter}}letter[max m{digit - letter}{digit letter}}} not reserved-word
The sum of m plus n is 28.	
device-number::=	unit-record-device utility-device direct-access-device
digit::=	0 1 2 3 4 5 6 7 8 9
division-name::=	DATA PROCEDURE ENVIRONMENT IDENTIFICATION
EBCDIC-character	Any character in the IBM Extended Binary Coded Decimal Interchange Character set.
elementary item::=	alphabetic-item alphanumeric-item report-item external-decimal-item internal-decimal-item binary-item external-floating-point-item internal-floating-point-item
entry-name::=	external-name
event-condition::=	{[AT]END} {ON[EVERY]} {{ON}SIZE ERROR}
exponent::=	min 1 max 2 digit
external-name::=	'letter max 7{digit letter}'
figurative-constant::=	{HIGH-VALUE HIGH-VALUES} {LOW-VALUE LOW-VALUES} {QUOTE QUOTES} {SPACE SPACES} ALL'{character not ''}' {ZERO ZEROES ZEROS}
file-name::=	data-name
floating-point-literal::=	[+ -]mantissa E[+ -] exponent
fp-form::=	{+ -}max n9[integer-1][V .] max m9[(integer-2)]F{+ -}99
The sum of m plus n plus the values of integer-1 plus integer-2 must not exceed 16.	
imperative-statement::=	See "Imperative Statements" in text.
integer::=	[max n digit] digit not 0 [max m digit]
The sum of m plus n is 17.	

letter::= 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

level-indicator::= FD|level-number

level-number::= {{b|0|2|3|4}digit}|77|88

library-name::= external-name

literal::= numeric-literal|non-numeric-literal|
 floating-point-literal

logical operator::= AND|OR|NOT

mantissa::= max m digit . max n digit
 The sum of m plus n is 16.

mnemonic-name::= data-name

mode::= F|U|V

model-number::= The model number of an IBM 360/
 series computer.

name::= data-name|procedure-name|external-
 name

non-numeric-literal::= alphabetic-literal|alphanumeric-
 literal

numeric-form::= {min 1 9[P...][V]}|
 {[V][P...] min 1 9}|
 {9...[V]9...}

The form (integer) placed after a 9 or a P specifies the equivalent of
 the appearance of that character integer times. The sum of all 9s and
 Ps plus the value of all integers must not exceed 18.

numeric-literal::= [+|-]max m digit[.]digit max n
 digit
 The sum of m plus n is 17.

operand::= literal|figurative-constant|data-
 name|arithmetic-expression

overflow-name::= data-name

paragraph::= [paragraph-name.] sentence. . .

paragraph-name::= I-O CONTROL|FILE-CONTROL|SOURCE-
 COMPUTER|OBJECT-COMPUTER|procedure-
 name

procedure-name::= {{digit|letter} [max 28 {digit|letter|-}
 {digit|letter}}}not reserved-word

program-name::= external-name

qualifier::= {OF|IN} {data-name|section-name}

record-name::= data-name

relational-operator ::= {GREATER [THAN] | >} | {EQUAL TO | =} |
{LESS [THAN] | <}

report-form ::= {+ | - | \$ | [* | Z] | P | , | B | 0 | 9 | [V | .] |
[CR | DB]} . . .

The valid combinations of these characters are described under "Report-Form Option." The form (integer) placed after any of these characters except V. CR and DB specifies the equivalent of the appearance of that character integer times.

reserved-word ::= Any word in the System/360 COBOL Word List, Appendix B in this publication.

section ::= section-header paragraph . . .

section-header ::= section-name SECTION.

section-name ::= CONFIGURATION | INPUT-OUTPUT | FILE
| WORKING-STORAGE | LINKAGE | procedure-name

sentence ::= {statement... | USE-sentence |
SELECT-sentence}.

simple-condition ::= [NOT] {relation-test | sign-test
| class-test | condition-name-test | overflow-test}

statement ::= imperative-statement | conditional-statement
| compiler-directing-statement

These types of statements are defined in the appropriate portions of the text.

test-condition ::= simple-condition | compound-condition

word ::= name | reserved-word

APPENDIX B: DISK AND TAPE OPERATING SYSTEMS COBOL WORD LIST

The words listed below make up the complete Disk and Tape Operating Systems COBOL vocabulary of reserved words.

ACCEPT	DATA	IBM-360
ACCESS	DATE-COMPILED	IDENTIFICATION
ACTUAL	DATE-WRITTEN	IF
ADD	DE	IN
ADVANCING	DECLARATIVES	INCLUDE
AFTER	DEPENDING	INDEXED
ALL	DESCENDING	INDICATE
ALPHABETIC	DETAIL	INITIATE
ALTER	DIRECT	INPUT
ALTERNATE	DIRECT-ACCESS	INPUT-OUTPUT
AND	DISPLAY	INSTALLATION
APPLY	DISPLAY-ST	INTO
ARE	DIVIDE	INVALID
AREA	DIVISION	I-O
AREAS		I-O-CONTROL
ASCENDING		IS
ASSIGN	ELSE	
AT	END	
AT END	ENDING	JUSTIFIED
AUTHOR	ENTER	
	ENTRY	KEY
BEFORE	ENVIRONMENT	
BEGINNING	EQUAL	LABEL
BLANK	ERROR	LABELS
BLOCK	EVERY	LAST
BY	EXAMINE	LEADING
	EXHIBIT	LEFT
CALL	EXIT	LESS
CF		LIBRARY
CH	FD	LINE-COUNTER
CHANGED	FILE	LINE
CHARACTERS	FILES	LINES
CHECKING	FILE-CONTROL	LINKAGE
CLOCK-UNITS	FILE-ID	LOCK
CLOSE	FILLER	LOW-VALUE
COBOL	FINAL	LOW-VALUES
CODE	FIRST	
COLUMN	FOOTING	MORE-LABELS
COMPUTATIONAL	FOR	MOVE
COMPUTATIONAL-1	FORM-OVERFLOW	MULTIPLY
COMPUTATIONAL-2	FROM	
COMPUTATIONAL-3		NAMED
COMPUTE	GENERATE	NEGATIVE
CONFIGURATION	GIVING	NEXT
CONSOLE	GO	NO
CONSOLE	GO	NO
CONTAINS	GREATER	NOT
CONTROL	GROUP	NOTE
CONTROLS		NUMERIC
COPY	HEADING	
CORRESPONDING	HIGH-VALUE	OBJECT-COMPUTER
COUNT	HIGH-VALUES	OCCURS
CREATING		OF
CYCLES		

OH	SA
OMITTED	SAME
ON	SD
OPEN	SEARCH
OR	SECTION
ORGANIZATION	SECURITY
OTHERWISE	SELECT
OUTPUT	SENTENCE
OV	SEQUENTIAL
OVERFLOW	SIZE
	SORT
PAGE	SOURCE
PAGE-COUNTER	SOURCE-COMPUTER
PERFORM	SPACE
PF	SPACES
PH	STANDARD
PICTURE	STOP
PLUS	SUBTRACT
POSITIVE	SUM
PRINT-SWITCH	SYMBOLIC
PROCEDURE	
PROGRAM-ID	SYSPUNCH
PROCEED	
PROCESSING	TALLY
PROTECTION	TALLYING
	TERMINATE
QUOTE	THAN
QUOTES	THEN
	THRU
RANDOM	TIMES
RD	TO
READ	TRACE
	TRACK
READY	TRACKS
RECORD	TRANSFORM
RECORDS	TYPE
REDEFINES	
RELATIVE	
RELEASE	UNIT
REMARKS	UNIT-RECORD
REPLACING	UNITS
REPORT	UNTIL
REPORTING	UPON
REPORTS	USE
RERUN	USING
RESERVE	UTILITY
RESET	
RESTRICTED	VALUE
RETURN	VARYING
REVERSED	
REWIND	WHEN
REWRITE	
RF	WITH
RH	WITHOUT
RIGHT	WORKING-STORAGE
ROUNDED	WRITE
RUN	
	ZERO
	ZEROES
	ZEROS

APPENDIX C: INTRARECORD SLACK BYTES AND RECORD ALIGNMENT IN BLOCK FILES

In IBM System/360, storage is organized into bytes. Four bytes comprise a word of storage. Two bytes comprise a half-word; eight bytes comprise a double-word. Certain types of processing operations require that data be aligned on a certain type of boundary--half-word, full-word, or double-word. In order to insure correct alignment in such cases, it is sometimes necessary to insert bytes containing no meaningful data between data-items or between records. These are called slack bytes. In certain cases, they are inserted by the compiler; in other cases, it is the responsibility of the user to insert them.

INTRARECORD SLACK BYTES

For ease of programing and efficient object code, the COBOL compiler:

- Provides automatic field alignment within working-storage section, and unblocked files.
- Provides the facility of working with input and output records directly in a buffer.
- Provides efficient use of file space by packing succeeding blocked records without regard to alignment.

Whenever the USAGE is defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 within an 01 record description, slack bytes, if required, are automatically added to the record by the compiler in order to ensure proper synchronization of these computational data items. These slack bytes are added on the assumption that all 01 levels are aligned on a double word boundary. It is the user's responsibility to ensure that the data item is aligned on a double word boundary when:

- The argument of a CALL statement corresponds to a data name defined with an 01 level in the LINKAGE SECTION of the subprogram.
- The record names are associated with a file where logical records are blocked.

The user can ensure this double word boundary alignment by moving the data item to an 01 level defined in working storage, or by adding interrecord slack bytes to force proper alignment of succeeding records in the block. (See Record Alignment Within Block Files.)

To determine the intrarecord slack bytes required, the compiler:

- Sums up the size of all elementary data items preceding a COMPUTATIONAL, COMPUTATIONAL-1 or COMPUTATIONAL-2 field including any slack bytes previously added.
- Divides this sum by K where:
K = 8 for COMPUTATIONAL-2
K = 4 for COMPUTATIONAL-1, or COMPUTATIONAL (5 digits or greater)
K = 2 for COMPUTATIONAL (4 digits or smaller)
- If the remainder (R) = 0, no slack bytes are required.

- If R=0, no slack bytes are required. If R≠0, K-R slack bytes are added.

The slack bytes are added at a level number equal to that of the group + 1 at the end of the group with the OCCURS.

See Figure 31 and the discussion that follows for an illustration of the use of slack bytes with an OCCURS clause.

CODING OF AN OCCURS CLAUSE

01 A.

02 B PICTURE IS X(7).

02 C OCCURS 3 TIMES.

03 D .

04 E PICTURE IS X.

04 F USAGE COMPUTATIONAL-2.

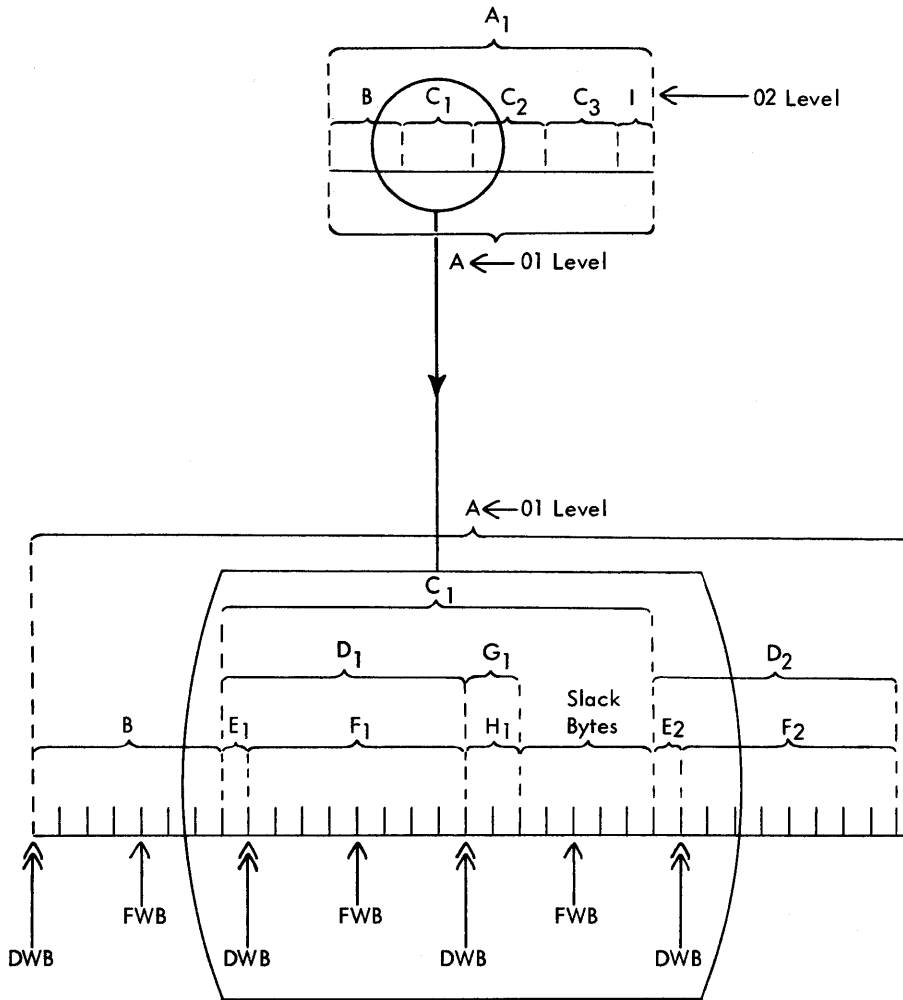
03 G.

04 H PICTURE IS XX.

Implied Filler 03 FILLER PICTURE IS X(5).

02 I PICTURE IS X.

Fields A, B, and C can be represented as shown in Figure 31.



DWB = Double Word Boundary
 FWB = Full Word Boundary

Figure 31. Use of Slack Bytes as a Filler When a Group Field is Defined

To compute the number of slack bytes, calculate the size of the group(C₁):

$$\frac{\text{BYTES}}{K} = 1 + 3,$$

$$\frac{\text{BYTES}}{K} = Q + R$$

$$\frac{11}{8} = 1 + 3,$$

$$K - R = \text{NUMBER OF SLACK BYTES}$$

8 - 3 = 5 Slack Bytes, therefore,

The filler is implied as:

03 FILLER PICTURE IS X(5).

RECORD ALIGNMENT WITHIN BLOCK FILES

Because the processing is done in a buffer, and not deblocked to a work area, the user must follow record alignment procedures within blocked files. When working with blocked files, slack bytes are not automat-

ically added as when working with data. However, diagnostics will inform the user of the number of slack bytes required for fixed-length record alignment.

For purposes of adding intrarecord slack bytes to assure proper alignment of COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 fields, all 01 levels are assumed to start on a double-word boundary.

Valid alignment of records can be accomplished by:

- Moving to an 01 in working-storage
- Adding the necessary slack bytes.

The compiler assures the user that all 01 levels in WORKING-STORAGE SECTIONS and all I/O buffers (not including any control bytes required by data management) will be aligned on a double-word boundary.

When processing records in the buffer or on a file where the logical records are blocked and contain COMPUTATIONAL, COMPUTATIONAL-1 and COMPUTATIONAL-2 fields, the user must add the necessary slack bytes to ensure proper alignment of the logical records within the buffer.

To determine if interrecord slack bytes are required:

- Sum up the size of the record (include all intrarecord slack bytes).
- Divide this sum by maximum K required in any one of the elementary items.
- If $R = 0$, no slack bytes are required.
If $R \neq 0$, $K - R$ slack bytes are required.

For alignment, the record should be expanded by the required number of slack bytes. Figure 32 (parts A and B) and the following text, illustrate invalidly aligned, and appropriately aligned, blocked files respectively.

Blocked V-type records containing COMPUTATIONAL-2 entries must be moved for proper alignment.

BLOCK FILES EXAMPLE CODING

FD A BLOCK CONTAINS 3 RECORDS LABEL RECORDS ARE OMITTED, DATA RECORDS IS B.

01 B.

02 C COMPUTATIONAL-1.

02 D PICTURE X.

Note the invalid alignment for a COMPUTATIONAL-1 field in Figure 32, Part A.

To rectify this condition, (invalid alignment), move field B to an 01 in WORKING-STORAGE where automatic alignment will be provided, or compute and add the required filler as shown in the following.

BLOCK FILE EXAMPLE CODING SHOWING THE FILLER FOR ALIGNMENT (REPEATED HERE FOR CLARITY).

FD A BLOCK CONTAINS 3 RECORDS LABEL RECORDS ARE OMITTED, DATA RECORDS IS B.

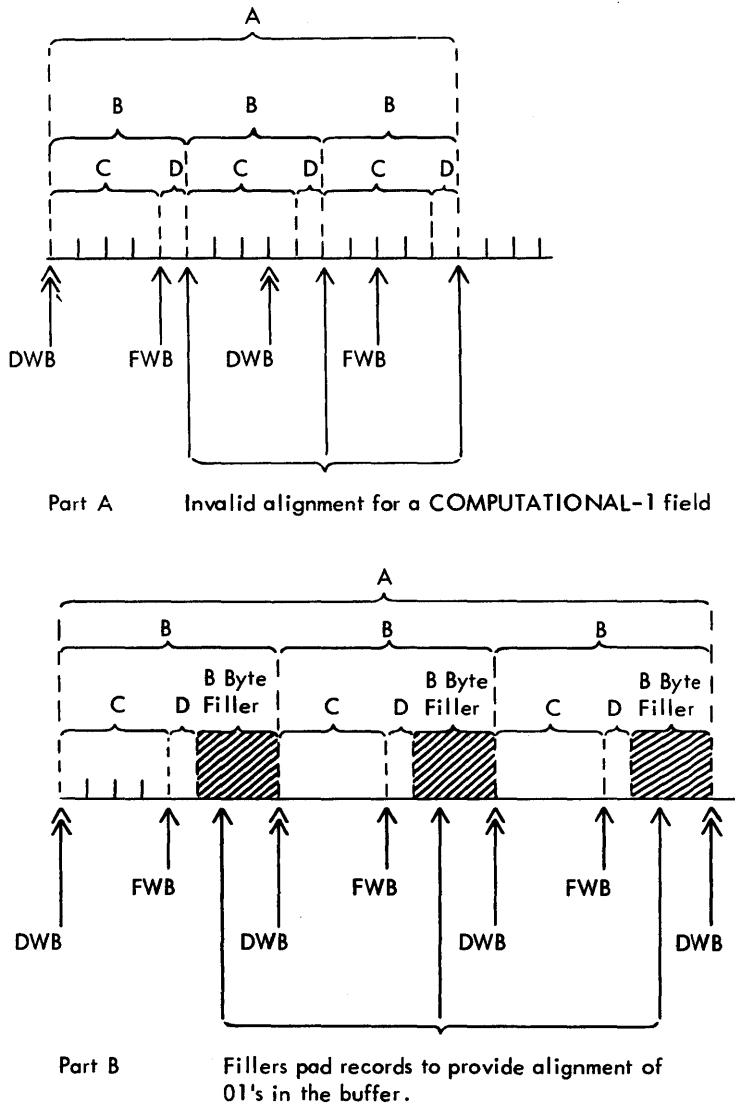
01 B.

02 C COMPUTATIONAL-1.

02 D PICTURE X.

02 FILLER PICTURE X(3).

Note how fillers pad records to provide alignment of all 01's in the buffer, Figure 32, Part B.



DWB = Double Word Boundary FWB = Full Word Boundary

Figure 32. Invalidly Aligned and Appropriately Aligned File A Buffer

SOME RULES TO REMEMBER

Linkage Section

In Linkage Section all 01's are assumed to be on a double-word boundary. It is the user's responsibility to ensure proper alignment between an argument in CALL, and the corresponding data name in an ENTRY statement.

In File Section

INPUT FILES: It is the user's responsibility to ensure that the logical records contain the necessary intrarecord slack bytes. If the file is blocked, and processing is done in the buffer, he must have added the necessary interrecord slack bytes when the file was created.

OUTPUT FILES: The compiler adds the necessary intrarecord slack bytes. The user defines the necessary interrecord slack bytes as required by input.

APPENDIX D: INTERMEDIATE RESULTS IN ARITHMETIC OPERATIONS

In the case of an arithmetic statement containing only a single pair of operands, no intermediate results are generated. Intermediate results are possible in the following cases:

1. In an ADD or SUBTRACT statement containing multiple operands immediately following the verb
2. In a COMPUTE statement specifying a series of arithmetic operations
3. In arithmetic expressions contained in IF or PERFORM statements

In concept the compiler treats a statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G is replaced by
MULTIPLY B      BY C      GIVING ir1
ADD A          TO ir1     GIVING ir2
DIVIDE E       INTO D     GIVING ir3
SUBTRACT ir3   FROM ir2   GIVING ir4
** F          BY G       GIVING ir5
ADD ir4       TO ir5     GIVING Y
```

This appendix contains a discussion of the compiler algorithms for determining the number of integer and decimal places reserved for intermediate results.

The following abbreviations will be used in this discussion and in Figure 33.

i--number of integer places carried for an intermediate result

d--number of decimal places carried for an intermediate result

d1,d2--number of decimal places defined for op1 or op2, respectively.

df--number of decimal places in final result field

ir--intermediate result field obtained from the execution of a generated arithmetic statement or operation. ir1, ir2, etc., represent successive intermediate results. These intermediate results are generated either in registers or in storage locations. Successive intermediate results may have the same location.

fr--number of integer and decimal places in final result field.

INTERMEDIATE RESULTS

The number of integer and decimal places contained in an ir is calculated as shown in Figure 33.

Operation	Statement Type	Decimal Places	Integer Places
+ or - (internal decimal) ¹	Arithmetic	d1 or d2, whichever is greater	i1 + 1 or i2 + 1, whichever is greater
+ or - (binary) ¹		d1 or d2, whichever is greater	i1 + 1 or i2 + 1 whichever is greater
*		d1 + d2	i1 + i2
/ if (i2+max(df+1,d2)+d1)≤30		df+1 or d2, whichever is greater	i2+d1
/ if (i2+max(df+1,d2)+d1)>30		d2-d1	i2+d1
**		df	fr - df
+ or -	IF or PERFORM	d1 or d2, whichever is greater	30 - d
*		d1+d2	30-d
/		d2	30-d
**		12	18

¹ The user should assume that i will increase by one in all + or - operations if either field is binary or packed.

Figure 33. Calculating Intermediate Results

COMPILER TREATMENT OF INTERMEDIATE RESULTS

The following indicates the action of the compiler when handling intermediate results.

If Value of $i + d$ is	The Action Taken is
<30	i integer and d
$=30$	decimal places are carried for ir . (If operation is $/$ or $**$, $i + d$ never exceeds 30)
>30	$30 - df$ integer and df decimal places are carried

Note: If **ROUNDED** is specified, the value of df is $df + 1$.

IBM**COBOL PROGRAM SHEET**Form No. X28-1464
Printed in U.S.A.

System IBM BOS/360		Punching Instructions				Sheet 1 of 2	
Program EXAMPLE OF A SUBPROGRAM		Graphic			Card Form#	*	
Programmer	Date	Punch				Identification 73 80	

SEQUENCE	COBOL	A	B															
(PAGE)	(SERIAL)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
003	001																	
	002																	
	003																	
	004																	
	005																	
	006																	
	007																	
	008																	
	009																	
	010																	
	011																	
	012																	
	013																	
	014																	
	015																	
	016																	
	017																	
	018																	
	019																	
Y	020																	
003	021																	

* A standard card form, IBM electro C61897, is available for punching source statements from this form.

Figure 35. Example of a Subprogram (Part 1 of 2)

INDEX

- A (PICTURE character) 51
- ACCEPT 84
- ACCESS Clause 26
 - SEQUENTIAL 26
 - RANDOM 26
- Accessing a Direct File Randomly 18
- Accessing an Indexed File Randomly 19
- Accessing an Indexed or Direct File Sequentially 19
- Access Methods 18
- Actual Decimal Point (PICTURE) 52
- ACTUAL KEY Clause 18, 27
- ADD 94
- Addition Arithmetic Operator 96
- AFTER 99
- AFTER ADVANCING LINES WRITE 81
- Alignment of Data Fields 41
- ALL
 - Character 37
 - EXAMINE 89
- Alpha-form (PICTURE) 51
- ALPHABETIC
 - Class Test 71
- Alphabetic Item 37
 - Format 47
 - PICTURE 51
- Alphanumeric Item 38
 - Format 47
 - PICTURE 51
- ALTER (See Go To) 97
- Alternate Area(s) 27
- An-form (PICTURE) 51
- AND (Logical Operator) 73
- APPLY Clause 29
 - RESTRICTED-SEARCH 30
 - WRITE-ONLY 30
- Arithmetic Expressions 96
 - COMPUTE 95
- Arithmetic Operators 96
- Arithmetic Statements and Options 92
- Arithmetic Verbs 77
- Argument 104
- Arrays 59
- ASSIGN Clause 25
- Assumed Decimal Point (V PICTURE Character) 51
- Asterisk (PICTURE Character) 52
- AT END
 - READ 79, 65
- Author 23

- B (PICTURE Character) 53
- Basic Facts 9
- BEGINNING 75
- Binary Item 39
 - Format 49
 - Subscripting 59
- BLANK Clause 56, 46
- BLANK (B PICTURE Character) 53
- BLANK WHEN ZERO 53, 56
- BLOCK CONTAINS 43, 44
- Braces in Formats 16
- Brackets in Formats 16

- Branching (See GO TO and PERFORM) BY (PERFORM)
 - EXAMINE 89
 - PERFORM 99
- BY
 - Multiply 95
- CALL 104
- Carriage Control 82
- CHANGED (EXHIBIT) 115
- Character Meaning (PICTURE)
 - 9 51
 - V 51
 - P 52
 - S 52
 - . 52
 - Z 52
 - * 52
 - CR 53
 - DB 53
 - , 53
 - O 53
 - B 53
 - \$ 53
 - + 54
 - 54
- Character Set 10
- Check Protection (PICTURE) 52
- Checking Labels 75
- Class Test 71
- CLOSE 20, 82, 85
- COBOL Character Set 10
- COBOL Debugging Language 115
- COBOL Processing Capabilities 17
- COBOL Program Sheet 13
- COBOL VERBS 77
- COMMA (PICTURE Character) 53
 - Punctuation 11
- Comments (NOTE) 106
- Comparison
 - Non-Numeric Items 70
 - Numeric Items 70
- Compiler-Directing Declaratives 74
- Compiler-Directing Statements 64, 104
- Compiler-Directing Verbs 77
- Compile-Time Debugging Packet 117
- Compound Conditions 73
- COMPUTATIONAL (DISPLAY) 50
- COMPUTATIONAL-1 50
- COMPUTATIONAL-2 50
- COMPUTATIONAL-3 50
- COMPUTE 95
- Concepts of Data Description 32
- Conditions
 - Compound 73
 - Event Conditions 65
 - Test Conditions 69
- Condition-Names 13, 33
- Condition-Name Test 72
- Condition-Name Values 57
- Conditionals 65
- Conditionals Statement Evaluation 64

Configuration Section Format	23	Entry Point	104
CONSOLE		Environment Division	23
ACCEPT	84	EQUAL TO	70
DISPLAY	83	Evaluation of Conditional Statement	66
Concepts of Data Division	32	Event-Conditions	65
Constants (See Literals)		EXAMINE	89
Continuation Indicator	14	EXHIBIT	115
Continuation of Non-Numeric Literals	15	EXIT Statement	106
Control Characters	81	Exponent	36
Control Field	42	Exponentiation Arithmetic Operator	96
COPY clause	107	Expressions	64
Count-Conditional Statement	117	External Decimal Item	38
Creating Labels	75	Format	48
Creation of Direct Files	20	External Floating-Point Item	40
Creation of Indexed Files	20	Format	49
Credit Symbol (CR PICTURE Character)	53	PICTURE	55
		External-Name	12, 26
Data Description Concepts	32	COPY (Library-Name)	107
Data Division	31	ENTER	104
Organization	31	INCLUDE (Library-Name)	108
Data Division Entry	31		
Data Division Sections		FD	43
File Section	41	FD (COPY)	107
Linkage Section	61	F Format Record	42
Working-Storage Section	61	Figurative Constants	36
Data Items	32	File and Record Handling	41
Data Manipulation Statements	85	FILE-CONTROL	25, 107
Data Manipulation Verbs	77	File-Control Paragraph	24, 25
Data-Names	12, 34	ACCESS Clause	24
Qualification	13	ACTUAL KEY Clause	24
Data Organization	17	ASSIGN Clause	24
DATA RECORDS	43, 45	ORGANIZATION Clause	24
Date-Compiled	22	RECORD KEY Clause	24
Date-Written	22	RESERVE Clause	24
Debit Symbol (DB PICTURE Character)	53	SELECT Sentence	24
Decimal Point (. PICTURE Character)	52	SYMBOLIC KEY Clause	24
Device Number	26	File-Name	13
DIRECT-ACCESS	25, 26	File Section	31, 41
Direct Data Organization	17	FD Entry	43
DIRECT Organization	26	File Section Entries	43
DISPLAY	83	FIRST	89
USAGE	50	Fixed Length Record	42
DISPLAY Statement	83	Fixed Point Items	38
DISPLAY-ST	110	Binary	39
DIVIDE	95	External Decimal	38
Division Arithmetic Operator	96	Internal Decimal	39
Division-Names (Margins)	14	Floating-Point Item	40
Dollar Sign (PICTURE Character)	54	Floating-Point Literal	36
		Floating String (PICTURE)	53
E (Floating-Point Literal)	36	Format Notation	15
Editing		Format F	42
MOVE	86	Format U	42
PICTURE	50	Format V	42
Elementary Items	37	FORM-OVERFLOW	
Format	47	APPLY	29
Ellipsis in Formats	16	Fp-Form (PICTURE)	55
ELSE		FROM	
IF	65	ACCEPT	84
ON	116	SUBTRACT	94
End of Volume		TRANSFORM	90
READ	80	WRITE	80
WRITE	81		
ENDING	75	Giving	93
ENTER Statement	104	GIVING	
ENTER COBOL	104	ADD	94
ENTER LINKAGE	104	SUBTRACT	94
ENTRY	104	MULTIPLY	95
Entry-Name	104	DIVIDE	95

GO TO	97, 104	
GREATER THAN	70	
Group Item	37	
Format	47	
Maximum Length	37	
Group Move	86	
Header Labels		
LABEL RECORDS	43	
USE	75	
HIGH-VALUES	36	
Hyphenated Words	15	
Hyphens	15	
Identification Division	22	
IF	65	
Evaluation	66	
Nested	67	
Test-Condition	69	
Imperative Statement	64	
IN	13	
INCLUDE Statement	108	
Indexed Data Organization	17	
INDEXED Organization	26	
INPUT	75	
Input/Output Processing	17	
Input/Output Section	24	
Input/Output Statements	77	
Input/Output Verbs	77	
I-O CONTROL	107	
I-O Control Paragraph	24, 28	
APPLY Clause	24, 29	
RERUN Clause	24, 29	
SAME Clause	24, 28	
IOCS	83, 75, 18	
Installation	22	
Internal Decimal Item	39	
Format	48	
Internal Floating-Point Item	40	
Format	49	
International Considerations	14	
INTO		
DIVIDE	95	
READ	79	
INVALID KEY		
READ	79, 65	
REWRITE	82, 65	
WRITE	80, 65	
Item Information Clauses	50	
JUSTIFIED RIGHT Clause	60	
Keys	18	
ACTUAL	27	
RECORD	28	
SYMBOLIC	27	
Key Words	15	
LABEL RECORDS	43, 44	
Labels		
Checking	75	
Creating	75	
LEADING	89	
LESS THAN	70	
Level Numbers	32	
Record Description	46	
Levels of Data Items	32	
Library Facility	107	
Library-Name	107	
Linkage Section	61, 32	
Literals		
Floating Point	36	
Non-Numeric	35	
Numeric	35	
LOCK	83	
Logical Operators	73	
Long-Precision	40	
COMPUTATIONAL-2	50	
Looping (PERFORM)	98	
LOW-VALUES	37	
Lower-Case Words	15	
Machine Requirements	9	
System Requirements	10	
Object Program Requirements	10	
Mantissa	36	
Margin Restrictions	14	
Minus Sign		
Arithmetic Operator	96	
PICTURE Character	54	
Mode	45	
MORE-LABELS	76	
MOVE	85	
Multiplication Arithmetic Operator	96	
Multiple Record File	42	
MULTIPLY	94	
Name Qualification	13	
NAMED (EXHIBIT)	117	
Name Types		
Data-Names	12, 34	
External-Names	12	
Procedure-Names	12	
Paragraph-Names	13	
Other-Names	13	
NEGATIVE	71	
Nested IF Statements	67	
NEXT SENTENCE		
IF	65	
ON	116	
Nine (9-PICTURE Character)	51	
Non-Numeric		
Comparison	70	
Literals	35	
Move	87	
NO (RESERVE)	27	
No Labels (Records)	44	
Non-Standard Label Records	44	
Non-Numeric literal	35	
NO REWIND	83	
NOT		
Conditions	72	
Logical Operators	73	
Notation Format	15	
NOTE Statement	106	
Numeric		
Class Test	71	
Comparison	70	
Literals	35	
Move	86	
PICTURE Character	51	
Numeric-Form (PICTURE)	51	
OBJECT-COMPUTER	23	
OCCURS Clause	58, 46	
OF	13	

OMITTED Labels (Records) 44
 ON 116
 Count-Condition 65, 116
 ON SIZE ERROR 93
 OPEN 78, 85
 OPEN Statement 77
 Operational Sign 52
 Operators (See Logical Operators,
 Arithmetic Operators, Relational
 Operators)
 Optional Words 15
 OPTIONS 93
 OR 73
 ORGANIZATION Clause
 INDEXED 26
 DIRECT 26
 Organization of Data Division 31
 Other Names 13
 OTHERWISE
 IF 65
 ON 116
 OUTPUT 75
 Overflow-Names 13
 Overflow Test 72

 P (PICTURE Character) 52
 Packed Decimal 39
 COMPUTATIONAL-3 50
 Paragraph-Name 12, 63
 Margins 13
 Parenthesis
 Arithmetic Expressions 96
 Compound Conditions 73
 Pence Decimal Fractions (Sterling) 114
 Pence (Sterling) 114
 Pence Terminators (Sterling) 114
 PERFORM Statement 98, 104
 Period 11
 PICTURE Clause 50, 46
 Considerations 55
 Sterling Non-Report 110
 Sterling Report 111
 Plus Sign
 Arithmetic Operator 93
 PICTURE Character 54
 POSITIVE 71
 Pound-Report-String 111
 Pound Separators (Sterling) 112
 Pound (Sterling) 111
 Prewritten Source Program 107
 Procedure Branching Statements 96
 Procedure Branching Verbs 77
 Procedure Division 63
 Procedure-Name 12, 63
 PROGRAM-ID 22
 Program Identification Code 14
 Program-name 22
 Program Sheet 13
 Punctuation 11, 15

 Qualification of Names 13
 Qualifiers 13
 QUOTE 37
 Quotient 95

 RANDOM-ACCESS 18, 26
 READ Statement 65, 79, 85
 READY 115

 RECORD CONTAINS 43, 44
 Record Description Entry
 Format 46
 Record size 37
 RECORDING MODE 43, 45
 RECORD KEY Clause 18, 19, 28
 Record length 37
 Record-Name 13
 Record Types 42
 REDEFINES Clause 57, 46
 REEL
 CLOSE 83
 RERUN 29
 Relational Operands 70
 Relational Operators 70
 Relation Test 70
 Remarks 22
 REPLACING 89
 Report-Form (PICTURE) 52
 Report Item 38
 Format 48
 PICTURE 51
 RERUN Clause 29
 RESERVE Clause 27
 Reserved Words 15
 RESET (TRACE) 115
 RESTRICTED SEARCH
 APPLY 30
 RETURN (ENTER) 104
 REWRITE Statement 82, 85, 65
 ROUNDED 93
 ADD 94
 COMPUTE 95
 DIVIDE 95
 MULTIPLY 95
 SUBTRACT 95
 Rules for Notation 15
 RUN (STOP) 97

 S (PICTURE Character) 52
 SAME Clause 28
 Sections 63
 Name Qualification 13
 Security 22
 SELECT (Copy) 107
 SELECT Sentence 25
 Sentences 64
 Separator 11
 Sequence Number 14
 SEQUENTIAL-ACCESS 18, 26
 Series of Imperative Statements 65
 Shilling Separator (Sterling) 112
 Shillings (Sterling) 112
 Short-Precision 50, 40
 COMPUTATIONAL-1 50
 Sign Test 71
 Simple Condition 69
 Single Record File 42
 SIZE ERROR 93, 65
 ADD 94
 COMPUTE 95
 DIVIDE 95
 MULTIPLY 95
 SUBTRACT 94
 SOURCE-COMPUTER 23
 Source Program Library Facility 107
 Source Program Statements 14
 SPACE 36

Space (S PICTURE Character)	53
Special Characters	10
Square Brackets in Formats	16
Standard and User Labels	44
Standard Error	76
STANDARD Label Records	44
Standard Sequential Data Organization	17
Statements	64
Arithmetic	92
Compiler-Directing	64
Conditional	64, 65
Data Manipulation	85
Imperative	64
Procedure Branching	96
Sterling Currency Feature	109
Sterling Non-Report	110
Sterling Report	111
Sterling Sign Representation	110
STOP Statement	97, 104
STOP literal	104
Structure of COBOL Source Program	13
Subscripting	59
SUBTRACT	94
Subtraction Arithmetic Operator	96
SYMBOLIC KEY Clause	18, 19, 27
Syntax	63
SYSPUNCH	83
Tables	59
TALLYING	89
Test-Conditions	65, 69
THEN (IF)	65
THRU (Perform)	98
TIMES (Perform)	98
TO	
ADD	94
ALTER	98
GO	97
MOVE	86
TRANSFORM	90
TRACE	115
Trailer Labels (USE)	75
Transfer (GO TO)	97
TRANSFORM	90
Truncation	93
Types of Data Items	37
Types of Names	12
Types of Statements	64
Unary Sign	96
UNIT	
CLOSE	83
RERUN	29
UNIT-RECORD	25, 26
Unspecified Format Record	42
UNTIL (Perform)	98
UNTIL FIRST (EXAMINE)	89
UPON	
CONSOLE	83
DISPLAY	83
SYSPUNCH	83
USAGE Clause	50, 46
DISPLAY-ST	111
User Labels	
LABEL RECORDS	44
USE	75
USE Statement	74
USING	
CALL	104
ENTRY	104
UTILITY	25, 26
V (PICTURE Character)	51
VALUE Clause	57, 46
LINKAGE SECTION	61
WORKING-STORAGE SECTION	61
Variable Format Record	42
Variable Record Length	42
VARYING (Perform)	98
Word Formation	11
Word List	124
Working Storage Section	61, 32
WRITE	65, 80, 85
WRITE-ONLY	
APPLY	30
X (PICTURE Character)	51
Z (PICTURE Character)	52
ZERO	
Figurative Constant	36
Sign Test	71
ZERO (O PICTURE Character)	53
Zero Suppression (Z PICTURE Character)	52
Zoned Format	38

READER'S COMMENT FORM

IBM System/360
Disk and Tape Operating Systems
COBOL Language Specifications

C24-3433-3

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. All comments will be handled on a non-confidential basis. Copies of this and other IBM publications can be obtained through IBM Branch Offices.

- | | Yes | No |
|--|--------------------------|---|
| ● Does this publication meet your needs? | <input type="checkbox"/> | <input type="checkbox"/> |
| ● Did you find the material: | | |
| Easy to read and understand? | <input type="checkbox"/> | <input type="checkbox"/> |
| Organized for convenient use? | <input type="checkbox"/> | <input type="checkbox"/> |
| Complete? | <input type="checkbox"/> | <input type="checkbox"/> |
| Well illustrated? | <input type="checkbox"/> | <input type="checkbox"/> |
| Written for your technical level? | <input type="checkbox"/> | <input type="checkbox"/> |
| ● What is your occupation? _____ | | |
| ● How do you use this publication? | | |
| As an introduction to the subject? <input type="checkbox"/> | | As an instructor in a class? <input type="checkbox"/> |
| For advanced knowledge of the subject? <input type="checkbox"/> | | As a student in a class? <input type="checkbox"/> |
| For information about operating procedures? <input type="checkbox"/> | | As a reference manual? <input type="checkbox"/> |
| Other _____ | | |
| ● Please give specific page and line references with your comments when appropriate. | | |

COMMENTS:

Fold

Fold

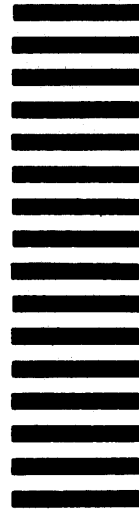
FIRST CLASS
PERMIT NO. 170
ENDICOTT, N. Y.

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

POSTAGE WILL BE PAID BY . . .

IBM Corporation
P. O. Box 6
Endicott, N. Y. 13760

Attention: Programming Publications, Dept. 157



Cut Along Line

Fold

Fold



International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Additional Comments:

IBM

**International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]**

**IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]**