File No. S360-24
Order No. GC28-6516-8

OS

IBM Systems Reference Library

IBM OS

COBOL (E, F) Language

Program Number (COBOL E) 360S-CO-503
(COBOL F) 360S-CB-524

COBOL (COmmon Business Oriented Language) is a programming language, similar to English, that is used for commercial data processing. It was developed by the Conference of Data Systems Languages (CODASYL).

This publication provides the programmer with rules for writing programs that are to be compiled by the COBOL E and COBOL F compilers under System/360 Operating System. Any violation of the rules for System/360 Operating System COBOL as defined in this publication is considered an error. The features implemented by the COBOL F compiler and not by COBOL E, and the IBM extensions to COBOL, are listed in an appendix.

IBM Technical Newsletter

File No. S360-24 (OS Release 20)

Re: Order No. GC28-6516-8

This Newsletter No. GN28-0427

Date: December 30, 1970

Previous Newsletter Nos. GN28-0266

IBM SYSTEM/360 OPERATING SYSTEM:
COBOL (E/F) LANGUAGE

This Technical Newsletter, a part of Release 20 of IBM System/360 Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent OS releases unless specifically altered.

Cover, 2	69-72
11,12	75,76
35,36,36.1	101,102
43,44	145,146
53,54	

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

SUMMARY OF AMENDMENTS

All changes are maintenance in nature, that is, to either clarify existing documentation or to correct typographical errors.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, N.Y. 10020

PRINTED IN U.S.A.

IBM Technical Newsletter

File Number	S360-24
Re: Form No.	GC28-6516-8
This Newsletter No.	GN28-0266
Date	June 1, 1970
Previous Newsletter Nos.	None

IBM SYSTEM/360 OPERATING SYSTEM
COBOL LANGUAGE

This Technical Newsletter, a part of Release 19 of the IBM System/360 Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent OS releases unless specifically altered. Pages to be inserted and/or removed are:

Cover,2	73,74
15,16	89,90
37,38	93,94
41,42	103-108
49,50	129-137
55,56	

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

Summary of Amendments

All changes in this Technical Newsletter are maintenance in nature. Areas affected are the VALUE clause, EXHIBIT CHANGED statement, PAGE-COUNTER and LINE-COUNTER, JUSTIFIED RIGHT clause, REWRITE statement, COPY clause, and the debugging packets.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, N.Y. 10020

PRINTED IN U.S.A.

PREFACE

This publication describes the IBM System/360 Operating System COBOL language as implemented for COBOL F and its subset, COBOL E. Its purpose is to serve as a reference manual for writing COBOL F and COBOL E programs.

The reader should have some knowledge of the COBOL language before using this publication. Useful COBOL information can be found in:

COBOL Program Fundamentals: Text, Form R29-0205

COBOL Program Fundamentals: Reference Handbook, Form R29-0206

Writing Programs in COBOL: Text, Form R29-0210

Writing Programs in COBOL: Reference Handbook, Form R29-0211

COBOL Programming Techniques: Text, Form R29-0215

Detailed information and examples helpful to the COBOL E and COBOL F programmer, including information about compiling, linkage editing, and executing COBOL E and COBOL F programs can be found in IBM System/360 Operating System: COBOL (E) Programmer's Guide, Form GC24-5029, and IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form GC28-6380.

A general knowledge of the IBM System/360 Operating System is desirable, although not mandatory. This knowledge can be acquired by reading IBM System/360 Operating System: Introduction, Form GC28-6534, and IBM System/360 Operating System: Concepts and Facilities, Form GC28-6535.

There are two Job Control Language manuals. IBM System/360 Operating System Job Control Language User's Guide, Form GC28-6703, is for those programmers with little or no knowledge of IBM's Job Control Language. It is to be used as a tutorial manual. IBM System/360 Operating System: Job Control Language Reference, Form GC28-6704, is for those programmers with a basic knowledge of JCL. It is designed to be used as a reference manual. In this manual, reference will only be made to the Job Control Language Reference, Form GC28-6704.

Most statements in this publication are common to both COBOL E and COBOL F. When these statements are processed by the respective compilers, the results produced are equivalent. Features implemented for COBOL F and not for COBOL E are designated throughout this publication by the symbol

When an entire chapter is implemented only for COBOL F (for example, the chapter entitled "Sort Feature"), the heading for the chapter is preceded by the above symbol.

When an entire paragraph is implemented only for COBOL F, the paragraph begins with the above symbol.

Certain features in this publication are IBM extensions to COBOL for System/360 Operating System; they are designated throughout this manual by the symbol

When an entire chapter describes an extension to COBOL (for example, the chapter entitled "COBOL Debugging Language"), the heading for the section is preceded by the above symbol.

When an entire paragraph describes an extension to COBOL, the paragraph begins with the above symbol.

ACKNOWLEDGMENT

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce 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.

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

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

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures 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 (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

BASIC FACTS 9
Character Set 9
Punctuation 9
Word Formation 10
Types of Names 10
Data-Names 10
External-Names 10
Procedure-Names 10
Paragraph-Names 10
Other Names 10
Qualification of Names 10
COBOL Program Sheet 11
Sequence Number: (Columns 1-6) 11
Continuation Indicator: (Column 7) . . 11
Source Program Statements: (Columns 8-72) 11
Program Identification Code: (Columns 73-80) 11
Margin Restrictions 12
Continuation of Non-Numeric Literals . 12
Format Notation 12

COBOL INPUT/OUTPUT PROCESSING CAPABILITIES 15
Data Organization 15
Standard Sequential Data Organization 15
Indexed Data Organization 15
Direct Data Organization 16
Relative Data Organization 16
Access Methods 16
File-Processing Techniques 16
Standard Sequential File-Processing Technique 16
Indexed File-Processing Techniques . 17
Direct File-Processing Techniques . 19
Relative File-Processing Techniques 20
Summary of File-Processing Specifications 21

IDENTIFICATION DIVISION 23

ENVIRONMENT DIVISION 25
Configuration Section 25
Special-Names Paragraph 25
Input-Output Section 26
File-Control Paragraph 26
SELECT Sentence 26
ASSIGN Clause 26
ACCESS Clause 27
ORGANIZATION Clause 27
RESERVE Clause 27
SYMBOLIC KEY Clause 27
ACTUAL KEY Clause 28
RECORD KEY Clause 28
TRACK-AREA Clause 29
FILE-LIMIT Clause 29
I-O-Control Paragraph 29
SAME Clause 29
RERUN Clause 29
APPLY Clause 30

DATA DIVISION 33
Organization 33
Data Description 34
Level Indicators 34
Data-Names 34
Qualification of Data-Names 35
Literals 35
Non-Numeric Literals 35
Numeric Literals 35
Floating-Point Literals 35
Figurative Constants 36
Condition-Names 37
Types of Data Items 37
Group Items 37
Elementary Items 37
Alphabetic Item 38
Alphanumeric Item 38
Report Item 38
Fixed-Point Items 38
Floating-Point Items 39
File Section 40
Record Formats 40
Data Division Entry Formats 41
File Section Entries 41
File Description 41
Sort Description 41
File Section Notes 41
Clauses 42
Record Description Entry 45
Group Item Format 45
Formats for Elementary Items 45
REDEFINES Clause 47
USAGE Clause 49
PICTURE Clause 49
Alpha-form Option 50
An-form Option 50
Numeric-form Option 50
Report-form Option 50
Fp-Form Option 52
Additional Notes on the PICTURE Clause 53
BLANK Clause 53
VALUE Clause 53
OCCURS Clause 54
Subscripting 55
Subscripting A Qualified Data-Name . 55
JUSTIFIED RIGHT Clause 56
Working-Storage Section 57
Linkage Section 57

PROCEDURE DIVISION 59
Syntax 59
Statements 59
Compiler-Directing Statement 59
Imperative Statement 59
Conditional Statement 59

Sentences 60
Paragraphs 60
Sections 60
IF Statement 60
Evaluation of Conditional Statements . 60
Nested IF Statements 61
Test-Conditions 62
Relation Test 62
Sign Test 64
Class Test 64
Condition-Name Test 66
Overflow Test 66
Compound Conditions 66
Implied Subjects and Operators 67
Arithmetic Expressions 67
Compiler-Directing Declarative Sections 68
USE Sentence 68
COBOL Verbs 69
Input/Output Statements 69
OPEN Statement 69
READ Statement 71
WRITE Statement 72
REWRITE Statement 74
CLOSE Statement 75
DISPLAY Statement 77
ACCEPT Statement 78
Data Manipulation Statements 79
MOVE Statement 79
EXAMINE Statement 81
TRANSFORM Statement 83
Arithmetic Statements 85
GIVING Option 85
ROUNDED Option 85
SIZE ERROR Option 86
COMPUTE Statement 86
ADD Statement 87
SUBTRACT Statement 88
MULTIPLY Statement 88
DIVIDE Statement 88
Procedure Branching Statements 89
GO TO Statement 89
ALTER Statement 89
PERFORM Statement 90
STOP Statement 93
Compiler-Directing Statements 93
ENTER Statement 93
EXIT Statement 94
NOTE Statement 95

REPORT WRITER FEATURE 97
Introduction 97
Data Division Considerations 98
File Section 98
REPORT Clause 98
Report Section 98
CODE Clause 99
CONTROL Clause 99
PAGE LIMIT Clause 100
Report Group Description Entry . . .101
TYPE Clause 101
LINE Clause 102
NEXT GROUP Clause 103
COLUMN Clause 103
GROUP INDICATE Clause 103
SOURCE Clause 104
SUM Clause 104
PAGE-COUNTER and LINE-COUNTER 104
Procedure Division Considerations . . .105
INITIATE Statement 105
GENERATE Statement 105
TERMINATE Statement 105
USE BEFORE REPORTING Sentence . . .106
Sample Report Writer Program and Output 107
Key Relating Report to Report Writer Source Program 117

SORT FEATURE 119
Elements of the Sort Feature 119
Sort Work Files 119
Environment Division Statements . . .119
SELECT Entry 119
Data Division Statements 120
Sort Description Entry 120
Record Description Entry 120
File Description Entry 121
Procedure Division Statements 121
SORT Statement 121
The Input Procedure 123
RELEASE Statement 123
The Output Procedure 124
RETURN Statement 124
Control of Input/Output Procedures .124
Control Flow 125
Examples of a SORT Statement 125
Sample Program Using the Sort Feature 127

SOURCE PROGRAM LIBRARY FACILITY 129
COPY Clause 129
INCLUDE Statement 130
Extended Source Program Library Facility 130

STERLING CURRENCY FEATURE AND INTERNATIONAL CONSIDERATIONS 131
Sterling Currency Feature 131
Sterling Non-Report 131
Sterling Sign Representation 132
Sterling Report 132
Procedure Division Considerations . .134
International Considerations 134

COBOL DEBUGGING LANGUAGE 135
TRACE 135
EXHIBIT 135
ON (Count-Conditional Statement) . .136
Compile-Time Debugging Packet 136

APPENDIX A: SYSTEM/360 OPERATING SYSTEM COBOL WORD LIST 139

APPENDIX B: SLACK BYTES 141
Intra-Record Slack Bytes 141
Inter-Record Slack Bytes 142
Summary of Data Division Requirements .142

APPENDIX C: INTERMEDIATE RESULTS 145
Intermediate Results--E Compiler . . .145

Intermediate Results--F Compiler . . .145
Compiler Treatment of Intermediate Results 146

PPENDIX D: EXAMPLES OF COBOL PROGRAMS .149

APPENDIX E: COBOL F ONLY FEATURES AND EXTENSIONS 151
Cobol F Only Features 151
Cobol Extensions 151

INDEX 153

ILLUSTRATIONS

FIGURES

Figure 1. Subdivisions of a Weekly Time-Card Record 34
Figure 2. Example of Data Levels . . . 35
Figure 3. Condition-Name Example . . . 37
Figure 4. Internal Representation of Numeric Items 39
Figure 5. Relation Between Labels and Device Assignment 42
Figure 6. Storage Layout for Subscripting Example 56
Figure 7. Evaluation of IF or ON Conditional Statement 61
Figure 8. Evaluation of Conditional Statement other than IF or ON. 61
Figure 9. Conditional Statements with Nested IF Statements 62
Figure 10. Logical Flow of Conditional Statement with Nested IF Statements 63
Figure 11. Data Movement Effected by MOVE CORRESPONDING Statement 81
Figure 12. Logical Flow of Option 4 PERFORM statement Varying One Data-Name 92
Figure 13. Logical Flow of Option 4 PERFORM Statement Varying Two Data-Names 92
Figure 14. Logical Flow of Option 4 PERFORM Statement Varying Two Data-Names 92
Figure 15. Sample Portions of a Report Produced by Report Writer Feature 96
Figure 16. COBOL Program with Report Writer Feature 108
Figure 17. Report Produced by Report Writer Feature 112
Figure 18. Flow of Data Through a Sorting Operation 126
Figure 19. COBOL Program Using Sort Feature 127
Figure 20. Format of Sterling Report PICTURE Clause 133
Figure 21. Example of a Calling Program 149
Figure 22. Example of a Called Program 150

TABLES

Table 1. System/360 Operating System COBOL File-Processing Techniques 16
Table 2. Permissible Data Organization Clauses and Statements . . 22
Table 3. Values for Calculating Record Length 49
Table 4. Editing Applications of the PICTURE Clause 53
Table 5. Permissible Comparisons . . . 65
Table 6. Valid Forms of Class Test . . 66
Table 7. Truth Table 67
Table 8. Formation of Symbol Pairs . . 67
Table 9. Error-Processing Summary . . 70
Table 10. Volume Positioning for Intermediate Reels 76
Table 11. Restrictions on Use of Input/Output Statements 79
Table 12. Examples of Data Movement . . 80
Table 13. Permissible Moves 82
Table 14. Examples of Data Examination 83
Table 15. Combinations of the FROM and TO Options of the TRANSFORM Statement 84
Table 16. Examples of Data Transformation 85
Table 17. Rounding or Truncation of Calculations 86
Table 18. Restrictions for Procedure-Branching Statements 93
Table 19. Collating Sequence for Specific Data Items 121
Table 20. Sterling Currency Editing Applications 134
Table 21. Calculating Intermediate Results Using the E Compiler. 146
Table 22. Compiler Action on Intermediate Result 147

This chapter contains those facts that are basic to writing IBM System/360 COBOL programs. It includes a listing of the characters that are recognized by the COBOL E and COBOL F compilers. Also included are such special topics as punctuation, types of names, qualification of names, and the rules for writing COBOL source programs on a program sheet. The final part of this chapter contains an explanation of the system of notation used throughout the entire publication.

CHARACTER SET

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

Digits 0 through 9
Letters A through Z
Special characters:
+ Plus sign
- Minus sign or hyphen
Blank or space
* 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 relational tests:

> Greater than
< Less than
= Equal to

All of the preceding characters are contained in the COBOL character set. In addition, the programmer may use as characters in non-numeric literals any characters (except the quotation mark) included in the IBM Extended Binary Coded Decimal Interchange Code (EBCDIC); 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. Except in the case of a unary operator, an arithmetic operator or an equal sign must always be preceded by a space and followed by another space. A unary operator is a plus (+) or a minus (-) sign that is prefixed to a data-name, an arithmetic expression, or a literal. A unary operator may be preceded by a left parenthesis.

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.

8. A comma or a semicolon may be used to separate a series of clauses.

9. A semicolon, a comma, or the word THEN may be used to separate a series of statements.

WORD FORMATION

A word consists of not more than 30 characters chosen from the following set of 37 characters: the letters A through Z, the digits 0 through 9, and the hyphen (-), which can appear anywhere in the word except as the first or last character.

A word is ended by a space, or by proper punctuation. A word may contain more than one embedded hyphen; consecutive embedded hyphens are also 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

Several types of names are used in writing a COBOL program. Each must conform to specific requirements.

DATA-NAMES

A data-name must contain at least one alphabetic character and must be formed according to the rules for word formation. It is used to identify a data item in the Data Division.

EXTERNAL-NAMES

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

PROCEDURE-NAMES

Procedure-names follow the rules for word formation.

Procedure-names may be composed solely of numeric characters. When so written, procedure-names are equivalent only if they are composed of the same number of digits having an equal numeric value. For example, 00123 and 123, when used as procedure-names, are not equivalent.

PARAGRAPH-NAMES

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

OTHER NAMES

The following are formed according to the rules for data-names:

- FILE-NAMES
- SORT-FILE-DESCRIPTION-NAMES
- REPORT-NAMES
- CONDITION-NAMES
- OVERFLOW-NAMES

QUALIFICATION OF NAMES

Every name used in a COBOL source program must be unique within the source program in either of two ways:

1. Because no other name has the identical spelling.

2. 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 this method of ensuring uniqueness is used. 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 for all levels higher than the level number of the data name being qualified. 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. For COBOL E, the total number of characters in a series of qualifiers together with the qualified name may not exceed 300.
8. No matter what qualification is available, a procedure-name must not be the same as any data-name.
9. A file-name is the highest level qualifier that can be used to qualify a data-name.
10. A section-name is the highest level qualifier (and the only qualifier) that can be used to qualify a procedure-name.

COBOL PROGRAM SHEET

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

The COBOL program sheet, despite its necessary restrictions, is 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 must not be used. The sequence number has no effect on the source program and need not be written, unless the user wishes to refer to a card with an INSERT or DELETE card (see "Extended Source Program Library Facility"). 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.

┌-------┐
|F ONLY|
└-------┘ Sequence checking may be suppressed via an EXEC statement parameter. (See the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form GC28-6380.)

CONTINUATION INDICATOR: (COLUMN 7)

A hyphen placed in this column indicates the continuation of non-numeric literals (see "Continuation of Non-numeric Literals").

SOURCE PROGRAM STATEMENTS: (COLUMNS 8-72)

These columns are used for writing the COBOL source program. Conceptually, one blank is assumed to be appended after column 72 on every line of a program sheet, except where a non-numeric literal spans more than one line. Hence, if the last character of a word is in column 72, a blank is assumed to be appended to it, thus terminating the word.

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).

The names of divisions must begin in Margin A. The division-name must 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, followed by a space, the word SECTION, and then a period. This entry must appear on a line by itself, except in the DECLARATIVES portion of the Procedure Division, or when an INCLUDE statement follows it. A paragraph-name must also begin in Margin A, and must be followed immediately by a period and a space. Statements may start on the same line in Margin B. Succeeding lines of the paragraph must be written in Margin B.

The level indicators (FD, SD, and RD) of the File, Sort, and Report Description entries in the Data Division, must begin in 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 rest of the entry (data-names and/or clauses) must not begin before column 12.

CONTINUATION OF NON-NUMERIC LITERALS

When a non-numeric literal occupies more than one line of a coding sheet, the following rules apply:

1. A hyphen is placed in column 7 of the continuation line.
2. A quotation mark is placed in Margin B preceding the continuation of the literal.
3. All spaces at the end of the continued line and any spaces following the quotation mark in the continuation line and preceding the final quotation mark of the literal are considered part of the literal.

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. Although it is not part of COBOL, this notation is useful in describing COBOL.

1. All words printed entirely in capital letters are reserved words. These are words that have preassigned meanings in COBOL. In all formats, words in capital letters represent an actual occurrence of those words.
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. These words 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. Words printed in lower-case letters in formats represent information to be supplied by the programmer. All lower-case words that appear in a format are defined in the accompanying text.
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.

System/360 COBOL supports various data organizations, record formats, and access methods. The facilities available to the COBOL programmer are specified in this chapter.

In this publication, the term file can be considered equivalent to the term data set used in other IBM System/360 Operating System publications.

A file is described to the operating system by a data control block. The sources of information for the construction of this data control block are the Environment Division and FD entry of the COBOL program, the Data Set Label, and the DD statement. A file is considered to be created when it is opened as an output file.

The DD statement is associated with the System/360 Operating System control program. A number of file characteristics and file-processing operations are controlled by entries in the DD statement. In some cases, the DD statement is used only when the COBOL user does not specify his particular choice among such characteristics and operations.

Certain characteristics of files (for example, the NTM parameter in files with indexed organization) cannot be expressed in the COBOL language, and may be specified on the DD card. The following functions can only be expressed by the DD card:

1. If it is desired that a file be catalogued, this may be specified by means of the DD card.
2. The amount of space allocated for a direct-access output file must be specified on the DD card.

Additional functions of the DD statement are cited in the descriptions of statements and clauses throughout this publication.

Further information concerning the DD statement, the Data Set Label, and data sets is found in the following publications:

IBM System/360 Operating System: Supervisor and Data Management Services, Form GC28-6646.

IBM System/360 Operating System: Job Control Language Reference, Form GC28-6704.

DATA ORGANIZATION

System/360 COBOL provides four types of data organization: standard sequential, indexed, relative, and direct.

The number and type of control fields used to locate logical records in a file differ, depending on which type of data organization is used. Consequently, each type of data organization is incompatible with the others. For example, a file created with standard sequential organization cannot also be read as a file with indexed organization. Organization of an input file is the same as the organization of the file when it was created.

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 were created (or in sequentially reversed order if the REVERSED option of the OPEN statement is written). This type of data organization is normally used for tape or unit-record files, but it is device-independent. Standard sequential files may be assigned to utility, direct-access, or unit-record devices.

Standard sequential data organization is assumed when the ORGANIZATION clause is not written in the Environment Division.

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 operating system and created with the file. The indexes are based on record keys provided by the COBOL programmer. Indexed files must be assigned to direct-access devices.

Indexed data organization is specified by writing the clause ORGANIZATION IS INDEXED in the Environment Division.

Direct Data Organization

Direct data organization is characterized by use of the relative track addressing scheme. When this addressing scheme is used, the position of each logical record in a file is determined by keys supplied by the programmer. The actual key specifies the track (relative to the first track for a file) on which space to place a record is first sought, or at which the search for a record is to begin. A symbolic key is used to identify a record on a track.

It should be noted that the physical tracks allocated to a file are not necessarily contiguous. Records are positioned on each track in the order in which they are written.

Files with direct data organization must be assigned to direct-access devices.

Direct data organization is specified by writing the clause ORGANIZATION IS DIRECT in the Environment Division.

Relative Data Organization

Relative data organization is characterized by use of the relative record addressing scheme. When this addressing scheme is used, the position of the logical records in a file is determined relative to the first record of the file starting with the initial value of zero. Symbolic keys are used to identify the records. The records are positioned on each track in the order in which they are written.

Files with relative data organization must be assigned to direct-access devices.

Relative data organization is specified by writing the clause ORGANIZATION IS RELATIVE in the Environment Division.

ACCESS METHODS

Two access methods are provided by System/360 COBOL: sequential access and random access.

Sequential access 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.

Random access is the method of reading and writing records of a file in a programmer-specified manner; the control of successive references to the files is expressed by specifically defined keys supplied by the user.

FILE-PROCESSING TECHNIQUES

Each combination of data organization and access method specified in the COBOL language is defined as a file-processing technique. Of eight possible combinations, only seven are permitted. System/360 Operating System implements each file-processing technique with a specific Data Management access method.

The Data Management techniques are henceforth referred to as QSAM, QISAM, BISAM BSAM, and BDAM. The manner in which these techniques relate to COBOL programming is discussed in the following text.

Table 1 summarizes the relation between COBOL access and data organization, Data Management techniques, and file-processing techniques.

Table 1. System/360 Operating System COBOL File-Processing Techniques.

	Access	
Organization	Sequential	Random
Standard Sequential	QSAM	
Indexed	QISAM	BISAM
Direct	BSAM(1)	BDAM(1)
Relative	BSAM(2)	BDAM(2)

(1) With relative-track addressing scheme
(2) With relative-record addressing scheme

Standard Sequential File-Processing Technique

Only one file-processing technique is associated with standard sequential organization, the Queued Sequential Access Method, QSAM.

QSAM:

QSAM supports three record formats: format U records; format F records, blocked

or unblocked; and format V records, blocked or unblocked. Record formats are explained in the chapter "Data Division."

QSAM is assumed when the ORGANIZATION clause is omitted and the clause ACCESS IS RANDOM is not written. This file-processing technique is device-independent and can be used in programs executed with input/output devices assigned to direct-access, unit-record, or utility units, except when certain options are contained in the program. These options are: the I-O and REVERSED options of the OPEN statement, and the FORM-OVERFLOW option of the APPLY clause. When any of these options is written, the program is not device-independent and is valid only for those devices to which these options are applicable.

Use of the RESERVE clause in a SELECT statement permits definition of more than one buffer area, allowing overlap of input/output operations with the processing of data.

When QSAM is used, the READ, WRITE, and REWRITE statements provide the following functions for the following types of file:

Output Files: Output records are blocked as required, and data is written sequentially on an output device.

Input Files: One logical record at a time is made available, in the order in which the records were written. QSAM can be used to process input files created using QSAM, or input files created using BSAM, whose organization is standard sequential.

I-O Files: For direct-access devices, a logical record can be updated in place; that is, the record is read, updated, and rewritten from the same area. Alteration of record length, insertion of new records, or deletion of existing records is not permitted.

Specification of checkpoints and user error routines is permitted when using this access method.

The ACTUAL KEY, SYMBOLIC KEY, and RECORD KEY clauses cannot be associated with files using QSAM.

Indexed File-Processing Techniques

Two file-processing techniques are possible with files whose organization is indexed:

1. The Queued Indexed Sequential Access Method, QISAM.
2. The Basic Indexed Sequential Access Method, BISAM.

Both of these techniques support records in format F, blocked and unblocked.

When a file is created using an indexed file-processing technique, the RECORD KEY clause must be written in the File-Control paragraph of the Environment Division. When indexed file-processing techniques are used, the record is identified to the system by means of a record key. The record key for each record must be unique. Record keys must be ordered by the user when creating the file in ascending order from one record to the next. The record key for a data record is the contents of the field named in the RECORD KEY clause. The record key field should not include the first byte of the data record in the following cases:

1. If the records of the file are unblocked, since the control program requires special handling for the file when records are unblocked.
2. If the file contains records that are to be deleted.
3. If any record key in the file contains the figurative constant HIGH-VALUE in its first character position.

The use of indexed file-processing techniques is restricted to direct-access devices.

QISAM:

QISAM is the file-processing technique specified when the clause ORGANIZATION IS INDEXED is written, and the ACCESS clause is omitted or the clause ACCESS IS SEQUENTIAL is written. Although conceptually similar to QSAM, QISAM differs from it in two ways:

1. In QISAM, records must be presented in ascending order according to a user-specified key (the record key).
2. In QISAM, access to records is by means of an indexed structure by which the operating system associates each record key with a physical address.

When QISAM is used for reading or rewriting the records of a file, the record key is always read and rewritten in the same relative position it was assigned when the file was created. For this reason, the field named in the RECORD KEY clause must appear in the data record description in

the same relative position as it had when the file was created.

Writing of the RESERVE clause in a SELECT statement permits definition of more than one buffer area, allowing overlap of processing operations and input/output operations.

When QISAM is used, the READ, WRITE, and REWRITE statements provide the following functions for the following types of files:

Output Files: Indexed sequential files are created. Output records are blocked as required, and data is written sequentially. Room for insertion of new records at a future time may be reserved by constructing a dummy record whose first character is the figurative constant HIGH-VALUE. When HIGH-VALUE is moved into the first character position in a record, it marks the record for deletion. The contents of the record keys for such records must be in the proper sequence.

Input Files: Make available one logical record at a time, in the order in which the records were written. Files must have been created using QISAM. Records containing the figurative constant HIGH-VALUE as a first character are not made available.

I-O Files: Updating-in-place or deletion of a logical record is permitted. A logical record is updated-in-place by reading, updating, and rewriting it from the same area. Alteration of record length or insertion of new records is not permitted. A logical record is marked for deletion when the figurative constant HIGH-VALUE is in the first character position of the record. Records in the file that contain this delete code are not made available.

Specification of user error routines by use of an error-processing declarative section in the Procedure Division is permitted when using QISAM.

BISAM:

BISAM is specified when the clauses ORGANIZATION IS INDEXED and ACCESS IS RANDOM are written. A file may be opened as an input file or an I-O file using this technique. Only files created by use of QISAM can be referred to by use of BISAM.

Both symbolic keys and record keys are required when using BISAM. The symbolic key is the field named in the SYMBOLIC KEY clause of the File-Control paragraph.

When a record is read or rewritten using BISAM, the contents of the field named in the SYMBOLIC KEY clause are used to locate the record in the file with matching contents in the RECORD KEY field.

Since a record key is used to identify a record to the system, the record keys associated with the logical records of the file may be thought of as a table of arguments. The symbolic key may be considered to be a search argument that is compared with the entries in the table.

When a new record is added to an existing file, the contents of the field named in the SYMBOLIC KEY clause are used to locate the two records in the file between which the new record is to be inserted. The records sought are those whose respective RECORD KEY field contents are less than or greater than the value in the SYMBOLIC KEY field.

The READ, WRITE, and REWRITE statements with BISAM provide the following functions with the following types of files:

Input Files: One logical record at a time is made available, for files created in QISAM. The records are retrieved randomly on the basis of the symbolic key value. Records containing the figurative constant HIGH-VALUE as a first character are made available in a BISAM READ unless they have been displaced from their prime track during a previous update.

I-O Files: Records can be updated or deleted in the same manner as specified for QISAM files with two exceptions: When a BISAM file is opened as I-O and a REWRITE statement for the file-name is issued anywhere in the program, a REWRITE statement must be given after every READ statement, and before any other input/output statement, for the same file-name; also, for COBOL E only, when a BISAM file is opened as I-O anywhere in the program, a REWRITE statement must be given after every READ statement and before any other I-O statement on this file. Therefore, when this READ/REWRITE combination is required, the file cannot be opened as input within the same program. Use of the WRITE statement causes a direct insertion of the logical record. When a record is added, both the record key and the symbolic key must have the same value; this value must be unique in the file.

Specification of user error routines is permitted using this access method.

New records may be added to a BISAM I-O file. However, each new record is inserted so that the system maintains the ascending order according to the record key. Records that are displaced because of the insertion of the new record will be placed in an overflow area, if one exists. If the dis-

placed record has been marked for deletion, it will not be placed into the overflow area. If a file is later processed sequentially (using QISAM), the order in which the records are processed is not affected by their being in an overflow area, since the records are made available according to ascending record keys. The time required to retrieve a record in the overflow area is greater than that required to retrieve other records. Therefore, when many overflow records are developed, the programmer should organize the file by creating a new version, using the existing file as input.

Direct File-Processing Techniques

Two techniques are used with direct organization:

1. The Basic Sequential Access Method, BSAM, with relative track addressing.
2. The Basic Direct Access Method, BDAM, with relative track addressing.

BSAM is used to create files with direct organization; BDAM is used to refer to files with direct organization. A file that is created with direct organization must be referred to only with direct organization. (A BSAM file with standard sequential organization cannot be created using COBOL.)

In COBOL, both BSAM and BDAM are applicable only to direct-access devices. (Although within Data Management BSAM may be used to create files with standard sequential organization when it is used with unit-record or magnetic-tape equipment, COBOL uses only QSAM for this function.)

The direct file-processing techniques support records in three formats: format U, format F unblocked, and format V unblocked.

When direct organization is specified for a file, the contents of the field named in the ACTUAL KEY clause must be the relative position of the track with respect to the first track assigned to the file, and the contents of the field named in the SYMBOLIC KEY clause must be the key that makes the record unique on the track. The same symbolic key may be specified for more than one record, provided that the records are not on the same track. If two or more such records are on the same track, only the first record is accessed. Both the actual key and the symbolic key must be defined outside the file with which they are associated.

When direct organization is used with random access in reading or rewriting the records of a file, the actual key is used to indicate the track on which the search is begun for the record whose key matches that specified in the SYMBOLIC KEY field.

When adding a new record to an already existing file, the actual key indicates the first track on which space will be sought for the record, and the symbolic key is used to distinguish the added record from all other records on the track on which space is found. If the search for space is to extend beyond a single track, the symbolic key must be distinct from the keys of all the records on all the tracks on which space is sought.

When the file is referred to by sequential access, the advance from one logical record to the next is implicit. The key associated with the logical record is placed in the field named in the SYMBOLIC KEY clause. Again, this field must be defined outside the file with which it is associated.

BSAM: (WITH RELATIVE TRACK ADDRESSING)

BSAM with relative track addressing is assumed when ORGANIZATION IS DIRECT is written, and either the ACCESS clause is omitted or ACCESS IS SEQUENTIAL is written.

When the READ and WRITE statements are used in conjunction with BSAM, the following functions are provided for the types of files listed below:

Output Files: Data is written sequentially; a user-specified value (symbolic key) is associated with each of the logical records written. Only unblocked records in formats F, U, and V are supported. When the user wishes to switch tracks, he must add a number equal to the number of tracks to be advanced to the actual key. COBOL will add dummy (format F) or capacity (format V or U) records to complete the previous track or tracks. An actual key value of zero corresponds to the first track assigned to the file. If the initial value is not zero, COBOL will complete the intervening tracks with dummy or capacity records and write the first record on the track indicated by the actual key. When no more space is available on the specified track, the compiler generates coding to advance to the next track by adding a one to the contents of the field named in the ACTUAL KEY clause. Data Management will automatically replace the dummy or capacity records when additions are made to the file. At the time that the file is closed, dummy or capacity records are added to the current track and all following tracks as determined by the FILE-LIMIT clause.

Files created with BSAM may not be referred to by use of QSAM.

Input Files: One logical record is made available for processing in the order in which records were written for files created in BSAM. Dummy records, if present, are also made available. When a file is created using direct organization, it must be read using direct organization. The key associated with the logical record is placed in the symbolic key field by the compiler.

Specification of user error routines is permitted when using BSAM by writing an error-processing declarative section in the Procedure Division.

BDAM: (WITH RELATIVE TRACK ADDRESSING)

BDAM with the relative track addressing scheme is assumed when both ORGANIZATION IS DIRECT and ACCESS IS RANDOM are written.

Searching for the requested record can be limited to a specific number of tracks as specified by the APPLY RESTRICTED SEARCH clause. If the clause is omitted, the entire file is searched forward, starting with the track specified in the actual key.

The READ, WRITE, and REWRITE statements used in conjunction with BDAM provide the following functions:

Input Files: One logical record at a time is made available for files created in BSAM. The records are retrieved randomly on the basis of the symbolic and actual keys.

I-O Files: Records can be updated by reading, updating, and rewriting from the same area. Use of a WRITE statement causes the record to be added on the track specified. When adding a record, the symbolic key must be unique for the track(s) specified.

Specification of user error routines by writing an error-processing declarative section is permitted using this access method.

Relative File-Processing Techniques

Two techniques are used with relative organization:

1. The Basic Sequential Access Method, BSAM, with relative record addressing.
2. The Basic Direct Access Method, BDAM, with relative record addressing.

BSAM is used to create files with relative organization; BDAM is used to refer to files with relative organization. A file that is created with relative organization must be referred to only with relative organization.

As explained above under "Direct File-Processing Techniques," in COBOL, both BSAM and BDAM are applicable only to direct-access devices.

When a file with relative organization is created, the compiler assumes that when the logical records of the file are referred to by random access, the contents of the field specified in the SYMBOLIC KEY clause indicate the position of the record relative to the beginning of the file, starting with an initial value of zero. The symbolic key is defined outside the file with which it is associated. When the file is referred to by sequential access, no symbolic key is used, since the advance from one logical record to the next is implicit.

The relative file-processing techniques support records in only one format, format F unblocked.

BSAM: (WITH RELATIVE RECORD ADDRESSING)

BSAM with the relative record addressing scheme is assumed when ORGANIZATION IS RELATIVE is written, and the ACCESS clause is omitted or ACCESS IS SEQUENTIAL is written.

When READ and WRITE statements are used in conjunction with BSAM, the following functions are provided for the types of files listed below:

Output Files: Data is written sequentially, associating a relative record number with each of the logical records written. Only unblocked records in format F are supported in this mode. Since Data Management requires that all tracks be full, the compiler adds dummy records to complete the last track of the file when it is closed. Dummy records are identified by the presence of the figurative constant HIGH-VALUE in the first position of the record. When the file is used as an input file, the user must be able to recognize these dummy records. These records are assigned relative record numbers and can never be deleted, only replaced. In order to allow for file expansion, the user can write dummy records in a similar manner by means of the figurative constant HIGH-VALUE.

Input Files: One logical record at a time is made available, in the order in which records were written for files created in BSAM. Dummy records created by the compil-

er to complete the last track are also made available. A file created using relative organization must be read using relative organization.

Specification of user error routines is permitted when using BSAM, by writing an error-processing declarative section in the Procedure Division.

BDAM: (WITH RELATIVE RECORD ADDRESSING)

BDAM with the relative record addressing scheme is assumed when ORGANIZATION IS RELATIVE and ACCESS IS RANDOM are written.

The READ and REWRITE statements used in conjunction with BDAM provide the following functions:

Input files: One logical record at a time is made available for files created in BSAM. The records are retrieved randomly on the basis of the symbolic key, which contains the position of the record relative to the beginning of the file, starting with an initial value of zero.

I-O Files: Records can be updated by reading, updating, and rewriting a record from the same area. Records may not be added to the file except by replacement of dummy records specified by the user or the compiler.

Error routines may be specified by the user when this file processing method is used.

It should be noted that the foregoing discussion applies specifically to files referred to or created in a COBOL program. It is possible in lower level languages to create files with different addressing schemes, that is, files that are created using relative track and block identification, or actual device address. In general, such files may not be used by COBOL object programs.

Summary of File-Processing Specifications

Table 2 summarizes the clause and statement specifications allowed for each of the file-processing techniques. In addition, 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.

• Table 2. Permissible Data Organization Clauses and Statements

File-Processing Technique		Data Management	Addressing	Permissible	Type of OPEN	Type of I-O	Required	Other Required	Optional Clauses
ORGANIZATION	ACCESS	Technique	Scheme	Record Formats	Statement	Statement	Key Clauses	Clauses	
Standard Sequential (or not specified)	SEQUENTIAL (or not specified)	QSAM		F,U,V	INPUT [REVERSED]	READ AT END	Not permitted	ASSIGN LABEL RECORDS DATA RECORDS CLOSE	BLOCK CONTAINS RECORD CONTAINS RERUN SAME AREA RESERVE APPLY (option2, 3) REPORTS ARE* USE (option 1*, 2) CLOSE {REEL / UNIT}
					OUTPUT	WRITE [AFTER ADVANCING]			
					I-O	READ AT END REWRITE		ASSIGN TO DIRECT-ACCESS LABEL RECORDS DATA RECORDS CLOSE	
INDEXED	SEQUENTIAL (or not specified)	QISAM		F	INPUT	READ AT END	RECORD KEY	ASSIGN TO DIRECT-ACCESS LABEL RECORDS ARE STANDARD DATA RECORDS CLOSE	BLOCK CONTAINS RECORD CONTAINS SAME AREA RESERVE APPLY (option 3) USE(option 2)
					OUTPUT	WRITE [INVALID KEY]**			
					I-O	READ AT END REWRITE			
	RANDOM	BISAM		F	INPUT	READ [INVALID KEY]**	RECORD KEY SYMBOLIC KEY		BLOCK CONTAINS RECORD CONTAINS TRACK-AREA SAME AREA APPLY(option 3) USE(option 2)
					I-O	READ WRITE REWRITE [INVALID KEY]**			
DIRECT	SEQUENTIAL (or not specified)	BSAM	Relative Track	F,U,V	INPUT	READ AT END	SYMBOLIC KEY		RECORD CONTAINS RERUN USE(option 2) FILE-LIMIT (with output files) CLOSE UNIT
					OUTPUT	WRITE	SYMBOLIC KEY ACTUAL KEY		
	RANDOM	BDAM	Relative Track	F,U,V	INPUT	READ [INVALID KEY]**	SYMBOLIC KEY ACTUAL KEY		RECORD CONTAINS SAME AREA APPLY(option 1) USE(option 2)
					I-O	READ WRITE REWRITE [INVALID KEY]**			
RELATIVE	SEQUENTIAL (or not specified)	BSAM	Relative Record	F	INPUT	READ AT END	Not permitted		RECORD CONTAINS RERUN SAME AREA USE(option 2) CLOSE UNIT
					OUTPUT	WRITE			
	RANDOM	BDAM	Relative Record	F	INPUT	READ [INVALID KEY]	SYMBOLIC KEY		RECORD CONTAINS SAME AREA USE(option 2)
					I-O	READ REWRITE [INVALID KEY]			

*F ONLY
**INVALID KEY required, E ONLY

The Identification Division appears as the first of four parts of a COBOL program. It 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...]
```

Program-name consists of single quotation marks enclosing no more than eight alphabetic and numeric characters, the first of which must be alphabetic. 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 Extended Binary Coded Decimal Interchange Code set.

If DATE-COMPILED is specified, any sentences in that paragraph will be replaced in the program listing by the date of compilation.

1. SYSIN--system logical input device
2. SYSOUT--system logical output device
3. SYSPUNCH--system logical punch device
4. CONSOLE--console

Mnemonic names associated with input/output devices are used in ACCEPT and DISPLAY statements in the Procedure Division (see the chapter entitled "Procedure Division"). Mnemonic names associated with a single EBCDIC character are used in the CODE clause in the Report Writer Feature (see the chapter entitled "Report Writer Feature").

If the clause DECIMAL-POINT IS COMMA is written, the functions of the comma and the period are exchanged in report and floating-point PICTURE character strings and in decimal and floating-point literals. When this clause is written, the user must represent the decimal point, when required in a numeric literal or in the PICTURE clause, by a comma (,). The period must be used for the function ordinarily served by the comma. The purpose of this option is to facilitate the use of COBOL in countries where this convention is used. The functions are not interchanged for sterling report pictures. These items are processed as if the clause had not been specified.

Note: The DECIMAL-POINT IS COMMA facility is provided in COBOL E by a parameter in the EXEC statement.

INPUT-OUTPUT SECTION

The format of the Input-Output Section is:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT file-name ASSIGN-clause
   [ACCESS-clause]
   [ORGANIZATION-clause][1]
   [RESERVE-clause]
   [SYMBOLIC KEY-clause]
   [ACTUAL KEY-clause]
   [RECORD KEY-clause][1]
   [TRACK-AREA-clause][1]
   [FILE-LIMIT-clause] ... .
[I-O-CONTROL.
   [SAME-clause.] ...
   [RERUN-clause.] ...
   [APPLY-clause.] ...]
```

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

The I-O-Control paragraph 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 referred to in the COBOL source program.

The name of each file must be unique within a program, and must have a File Description (FD) entry 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 ASSIGN clause is used to assign a file to a particular device.

The format of the ASSIGN clause is:

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

External-name specifies the name by which the file is known to the Control Program. It is the name that appears in the name field of the DD card (See the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form C28-6380 or IBM System/360 Operating System: COBOL (E) Programmer's Guide, Form C-5029.

---

[1]Extension

DIRECT-ACCESS, UTILITY, and UNIT-RECORD specify device classes. Each file must be assigned to a device class. Access to files assigned to UNIT-RECORD or UTILITY must be sequential; data contained on these files is organized in the standard sequential fashion. Files assigned to DIRECT-ACCESS may have standard sequential, indexed, direct, or relative organization.

Device-number is used to specify a particular device type within a device class. If device independence for a file is desired, the device class should be UTILITY, no device number may be specified, and no device-dependent statements may be associated with the file-name of the file. At object time, such a file may be assigned to any device class (including UNIT-RECORD).

The allowable device numbers are:

DIRECT-ACCESS--2301, 2302, 2303, 2311, 2314, 2321

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

Note: R indicates Reader: P indicates punch.

UTILITY--2400, 2301, 2302, 2311, 2314, 2321

[F ONLY] Note that a sort-file may be assigned to any utility device except device-number 2321.

ACCESS Clause

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

The format of the ACCESS clause is:

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 the ORGANIZATION clause must be specified. A full explanation of the types of access available to users of COBOL is provided in the chapter "COBOL Input/Output Processing Capabilities."

[EXT] ORGANIZATION Clause

The ORGANIZATION clause indicates the organization of the data associated with a particular file. The format of the ORGANIZATION clause is:

INDEXED specifies indexed data organization; DIRECT specifies direct data organization; and RELATIVE specifies relative data organization.

If the ORGANIZATION clause is omitted, a standard sequential file is assumed. A full explanation of the types of data organization available to users of COBOL is provided in the chapter "COBOL Input/Output Processing Capabilities."

This clause may only be written for files assigned to direct-access devices.

RESERVE Clause

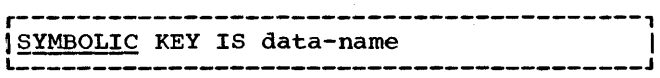
The format of the RESERVE clause is:

This clause specifies that the number of buffers represented by integer be reserved for a sequential file whose organization is standard sequential or indexed, in addition to the standard minimum of one required for a file. If NO is written, no additional buffers are reserved. If this clause is omitted, the number of buffers assigned at object time is taken from the DD card.

The value of integer must not exceed 254.

SYMBOLIC KEY Clause

The format of the SYMBOLIC KEY clause is:



This clause specifies data-name as the name of the item whose contents are used:

1. when reading or rewriting, to locate the record with the matching key.

2. when writing to create the key associated with the record.

Data-name must never be defined in any file or in the Linkage Section. When it is desired to have the logical effect of the symbolic key in a file or in the Linkage Section, a READ INTO or MOVE may be done to an item in the Working-Storage Section which is defined as the desired key.

If a record is added to an indexed sequential file, and the TRACK-AREA clause was not specified for the file, the contents of the SYMBOLIC KEY field are unpredictable after a WRITE statement is executed.

The SYMBOLIC KEY clause is required for sequential-access files whose organization is direct. The symbolic identity of the record is stored in data-name whenever a READ statement is executed for the file. Any changes the programmer may make to data-name do 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 record to be read or written must be placed in data-name before the READ or WRITE statement for that record is executed. The symbolic identity is transmitted to the Data Management portion of the operating system, and used to determine the physical location from which the record is to be read, or onto which it is to be written.

Data-name may specify any fixed-length item less than 256 bytes in length, except when used to specify a relative record number for a file with relative organization. In this case, data-name must be defined as an 8-integer binary item (that is, whose description contains the clauses PICTURE S9(8) and USAGE COMPUTATIONAL) whose maximum value does not exceed 16,777,215.

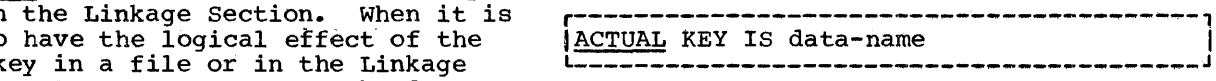
The SYMBOLIC KEY clause must precede all OCCURS DEPENDING ON clauses in the Working-Storage Section.

ACTUAL KEY Clause

This clause is used for files whose organization is direct. It is required except when access is sequential and the file is opened as input.

This clause specifies the name of a data item containing the relative track number in a file on which a record is to be found or placed. The relative track number for the first track assigned to the file is zero.

The format of the ACTUAL KEY clause is:



Data-name must never be defined in any file or in the Linkage Section. When it is desired to have the logical effect of the actual key in a file or in the Linkage Section, a READ INTO or MOVE may be done to an item in the Working-Storage Section which is defined as the desired key.

Data-name must be defined as a 5-integer binary data item (that is, whose description contains the clauses PICTURE S9(5) and USAGE COMPUTATIONAL) whose maximum value does not exceed 65,535.

EXT RECORD KEY Clause

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

The format of the RECORD KEY clause is:

Data-name must be defined to exclude the first byte of the record in the following cases:

1. Files with unblocked records

2. Files from which records are to be deleted

3. Files containing a record key with the figurative constant HIGH-VALUE in the first character position

In all other cases, the item specified by data-name may appear anywhere within the record.

When more than one record description is associated with a file, each description must contain a field for the record key. This field must be in the same relative position from the beginning of each record. It may be identified by different names in different record descriptions.

Data-name may be any fixed-length item less than 256 bytes in length.

[EXT] TRACK-AREA Clause

When records are to be added to random-access files with indexed organization, this clause specifies the area required. Efficiency in adding records to such files is considerably improved when the clause is specified.

The format of the TRACK-AREA clause is:

```
TRACK-AREA IS {data-name  } CHARACTERS
              {integer[1] }
```

The area must be no less than the size of an entire track plus one logical record.

When the data-name option is written, data-name must specify an item described with an 01 or 77 level number in the Working-Storage Section.

[F ONLY] When the integer option is specified, an area of integer bytes is obtained by the operating system when the file is opened. It is released to the system when the file is closed.

If a record is added to an indexed sequential file, and the TRACK-AREA clause was not specified for the file, the contents of the SYMBOLIC KEY field are unpredictable after a WRITE statement is executed.

The area defined by the TRACK-AREA clause must be a multiple of 8 and must not exceed 32,760 bytes.

FILE-LIMIT Clause

This clause specifies the number of tracks to be initialized for the creation of files with direct organization. This clause does not cause track allocation, which is the function of a DD card parameter (see the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form C28-6380).

The format of the FILE-LIMIT clause is:

```
FILE-LIMIT IS integer TRACKS
```

If the relative number of the last track used by the file when it is closed is less than that specified in the FILE-LIMIT clause, the tracks are initialized up to the last track specified by the clause. If the relative number of the last track is equal to or greater than that specified, or if the clause is omitted, the tracks are initialized up to the last track written. The remaining allocated tracks are not initialized. The number of tracks initialized is thus at least integer +1.

---

[1]Implement for COBOL F only

I-O-CONTROL PARAGRAPH

SAME Clause

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

The format of the SAME clause is:

```
SAME AREA FOR file-name-1
  file-name-2 ... .
```

No more than one of the files named in this clause may be open (by means of an OPEN statement) at one time.

More than one SAME clause may appear in a COBOL program, but no file-name may appear in more than one SAME clause.

A sequential file whose organization is standard sequential or indexed, named in this clause, must have a RESERVE clause associated with it.

This clause cannot refer to files whose organization is direct and whose access is sequential.

RERUN Clause

This clause specifies that checkpoint records are to be written on the unit specified by external-name. A checkpoint record is a recording of the status of the computer at a specific point in the execution of the object program. It contains all information necessary to restart the program from that point.

The format of the RERUN clause is:

```
RERUN ON 'external-name'
      EVERY integer RECORDS OF file-name.
```

External-name is the name by which the file is known to the operating system's control program, that is, the name that appears in the name field of the DD statement. External-name should be unique for RERUN clauses. (It should not be used in an ASSIGN clause.) The purpose of this stipulation is to prevent checkpoint records from being entered among other data sets. However, several RERUN clauses may specify either the same external-name or different ones.

Integer indicates the number of READ or WRITE statements to be executed on file-name between checkpoints. It must not exceed $2^{24}-1$, or 16,777,215. Checkpoint records are written immediately preceding execution of the first READ, WRITE, or REWRITE statement after the first OPEN, and thereafter. (Subsequent OPEN's and CLOSE's of file-name will not affect the count of integer.)

File-name must be unique for each RERUN clause, i.e., it may have only one RERUN clause associated with it.

Additional information concerning the use of the Checkpoint/Restart feature is contained in the publications IBM System/360 Operating System: COBOL (E) Programmer's Guide, Form C24-5029, and IBM System/360 Operationg System: COBOL (F) Programmer's Guide, Form C28-6380.

APPLY Clause

There are three options of the APPLY clause, each with its own function.

Option 1

Option 1 of the APPLY clause relates to files with direct organization. It is used to limit the number of tracks involved in searching for a record or in seeking space for an output record. The format of Option 1 of the APPLY clause is:

```
APPLY RESTRICTED SEARCH OF integer
   TRACKS ON file-name... .
```

When a record is read or rewritten, a search is made for that record whose key matches the symbolic key. The search starts with the relative track contained in the actual key, and continues until either the record is found or the number of tracks specified by integer has been exhausted. If the record is not found, an invalid key condition exists. When a record is being added to a file, the search for space starts on the relative track contained in the actual key, and continues until either space is found or until the number of tracks specified by integer has been exhausted.

When Option 1 of the APPLY clause is not written, searching starts with the relative track specified by the actual key, and continues until the condition is satisfied or until the entire file has been searched.

Option 2

Option 2 of the APPLY clause is used to specify overflow-name, a condition-name that may be used in a test for a form-overflow condition in a printer to which the file named by file-name is assigned. The condition is true if a form-overflow condition exists. Overflow-name follows the rules for data-name formation. The format of Option 2 of the APPLY clause is:

```
APPLY overflow-name TO FORM-OVERFLOW
   ON file-name.
```

A form-overflow condition exists when an end-of-page is sensed on printer channel 12 by an on-line printer.

An overflow-name test may be written in conjunction with a WRITE statement with an ADVANCING option, in order to control spacing of printed records.

Only one overflow-name may be applied to a file and only one buffer reserved (see the RESERVE clause in this section.)

The overflow-name test is discussed in the chapter entitled "Procedure Division."

Option 3

Option 3 of the APPLY clause is used to make optimal use of buffer and device space allocated for a file whose record format is V. The format of Option 3 of the APPLY clause is:

```
APPLY WRITE-ONLY ON file-name... .
```

Normally, a buffer is truncated when the maximum size record no longer fits. Use of this option will cause a buffer to be truncated only when the next record does not fit in the unused portion.

For example, assume that variable length records of from 200 to 500 characters are to be written out with a blocking factor of 3. The buffer size would be 1,500 bytes plus control bytes -- large enough to hold three records of maximum length. However, seven records, each 200 bytes in length, could also be accommodated. But unless APPLY WRITE-ONLY is specified for the file, the buffer would be truncated after the sixth record because the remaining space (300 bytes) is not large enough to hold a maximum size (500-byte) record. By using Option 3, the size of each output record would be inspected and the buffer filled with as many complete records as it could accommodate, regardless of their individual sizes.

The files named in this clause must have standard sequential or indexed organization.

For an output file, the data records associated with each of the file-names may be referred to only in a WRITE statement containing the FROM option. None of the subfields of these records may be referred to; in any file for which APPLY WRITE-ONLY was specified, the subfields of records can be referred to only if the file is opened for input or I-O.

If the APPLY WRITE-ONLY clause was originally specified for the creation of an indexed file (QISAM file processing technique), then APPLY WRITE-ONLY must be specified if any records are later added by use of the WRITE verb (BISAM file processing technique).

Sample coding for the ENVIRONMENT DIVISION would be as follows:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
   SOURCE-COMPUTER.  IBM-360 G50.
   OBJECT-COMPUTER.  IBM-360 G50.
INPUT-OUTPUT SECTION.
   FILE-CONTROL.
    SELECT TRANSACT ASSIGN TO 'DDNAME1'
     UTILITY.
    SELECT ACCOUNTS ASSIGN TO 'DDNAME2'
     DIRECT-ACCESS
    ACCESS IS RANDOM
    ORGANIZATION IS DIRECT
    SYMBOLIC KEY IS ACCOUNT-NUMBER
    ACTUAL KEY IS TRACK-NUMBER.
I-O-CONTROL.
    APPLY RESTRICTED SEARCH OF 1 TRACKS
     ON ACCOUNTS.
```

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 data that has a constant value and is defined by the user.

3. Linkage data descriptions for communication between the main program and subprograms and the operating system's Data Management label area.

4. Report format specifications and data to be included in a report.[1]

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

```
DATA DIVISION.
FILE SECTION.
    File Description entries
       Record Description entries
    Sort Description entries[1]
       Record Description entries
WORKING-STORAGE SECTION.
    Record Description entries
LINKAGE SECTION.[2]
    Record Description entries
REPORT SECTION.[1]
    Report Description entries
       Report Group Description entries
       Report Element 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

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 other name, and a series of clauses defining the data, that 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.

The File Section describes the content and organization of files and, for COBOL F, sort-files. Each such entry is followed by related Record Description entries.

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

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 with their values.

The Linkage Section is a required part of any COBOL subprogram that contains an ENTRY statement with the USING option and serves as a data-linking mechanism between the main program and the subprogram. The Linkage Section 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 in which entries do not cause object program data storage areas to be allocated. Hence, constants defined by the programmer cannot be included in this section as Record Description entries.

The Report Section describes the physical aspects of the report format and the conceptual characteristics of the data. It has three types of entries: the Report Description entry, which specifies the information pertaining to the overall format of the named report; the Report Group Description entry, which describes the characteristics for a report group; and the Report Element Description entry, which defines the characteristics of each elementary item included within a report group.

---

[1]Implemented for COBOL F only
[2]Extension

DATA DESCRIPTION

The following material defines the basic terms and concepts used in describing data. The rules that govern the writing of data descriptions appear later in this chapter.

LEVEL INDICATORS

Level indicators are used to show (in a format similar to an outline) how data items are related to each other. The most inclusive grouping of data is the file. The level indicator for an input/output file is FD.

| F ONLY |

For a sort file, the level indicator is SD.

For purposes of processing, contents of a file are divided into logical records, with level number 01 specifying a logical record. 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 special type of entry in the Linkage Section or the Working-Storage 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 allow specification of subdivisions of a record necessary for referring to data. Once a subdivision is specified, it may be further subdivided to permit more detailed data reference. This is illustrated by Figure 1 which is a weekly time-card record divided into four major items: name, employee-number, date, and hours, with more specific information appearing for the name and the 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 refers 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 level numbers and data-names shown in Figure 2.

DATA-NAMES

A data-name is a name assigned by the user to identify a data item used in a program. A data-name always refers to a kind of data, not to a particular value; the item referred to assumes a number of different values during the course of a program.

A data-name must contain at least one alphabetic character. A data-name or the key word FILLER must be the first word following the level number in each Record Description entry.

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. The key word FILLER may be used in place of a data-name if the item is not to be referred to directly or used as a qualifier. For example, if some of the characters in a record are not used in the processing steps of a program, then the data

TIME-CARD							
NAME			EMPLOYEE-NUMBER	DATE			HOURS
LAST	FIRST-INITIAL	MIDDLE-INITIAL		MONTH	DAY	YEAR	

Figure 1. Subdivisions of a Weekly Time-Card Record

```
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.
   03 HOURS.
```

Figure 2. Example of Data Levels

description of these characters need not include a data-name. In this case, FILLER may be written instead of a data-name after the level number.

Qualification of Data-Names

It should be noted that every data-name in a COBOL source program must be unique. For example, in Figure 1, qualification may be necessary in order to make the data-name YEAR distinct from another Data Division entry (e.g., YEAR OF DATE IN TIME-CARD).

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 Environment Division, Procedure Division, or Data Division (except in the REDEFINES clause where the position of the clause will eliminate the possibility of ambiguity).

Other rules for qualification of names are given in the chapter entitled "Basic Facts."

Note: For COBOL E, the programmer should take care not to use a reserved word as a data-name. Reserved words have preassigned meanings in the COBOL language, and improper usage of a reserved word can cause program failure with unexpected diagnostic messages.

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. (For example, in the sentence "MOVE 15 TO NUMBER-OF-EMPLOYEES," 15 is a literal.) 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 EBCDIC set, except quotation marks. All spaces enclosed by the quotation marks are included as part of the literal. A non-numeric literal must not exceed 120 characters in length.

For non-numeric literals requiring more than one line of a coding sheet, see "Continuation of Non-Numeric Literals" in the chapter "Basic Facts."

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 internal representation of a numeric literal is determined by its use in Procedure Division statements. For example, in the statement MOVE 24 TO A where A is COMPUTATIONAL, the value of 24 will be generated in binary format.

The following are examples of numeric literals:

```
+12572.6
-256.75
.16
```

[EXT] Floating-Point Literals

A floating-point literal must have the form:

[±]mantissaE[±]exponent

The plus or minus signs preceding the mantissa and exponent are the only optional characters within the format. The mantissa consists of from 1 to 16 digits. A decimal point is required.

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 *(10 ** 76). A zero exponent must be written as 0 or 00. An unsigned exponent is assumed to be positive.

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 is an example of a floating-point literal:

-.34566E+17

FIGURATIVE CONSTANTS

A figurative constant is a special type of literal; it represents a value to which a standard data-name has been assigned. A figurative constant must not be bounded by quotation marks.

ZERO may be used in many places in a program as a numeric literal. 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.

In COBOL E, ZERO may not be used in arithmetic statements.

A figurative constant may appear anywhere a literal may appear. When a figurative constant is associated with another data item (e.g., when it is moved to or compared with an item), the constant is repeated until it contains the same number of characters as the data item with which it is associated.

The following are the figurative constants and their meanings:

ZERO ZEROS ZEROES	Represents one or more zeros.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Represents one or more appearances of the highest value in the computer's collating sequence (hexadecimal FF). A 1-character field whose value is HIGH-VALUE may be used to operate as a delete-code or dummy-record-code character in files with indexed data organization and as a dummy-record-code character in files with relative organization.
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 EBCDIC character bounded by quotation marks. Character may not be a quotation mark.

[F ONLY]

ALL figurative constant	Represents one or more appearances of the figurative constant. Figurative constant may be neither QUOTE (QUOTES) nor a recurrence of ALL. ALL figurative constant is not permitted in a VALUE clause.
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.

The following are examples of the use of figurative constants in the Data Division:

```
02 FILLER PICTURE A(10) VALUE SPACES.

02 HEADING PICTURE X(20) VALUE ALL '*'.
```

CONDITION-NAMES

The general form of a condition-name entry is:

```
88 condition-name VALUE IS literal.
```

A condition-name is a name assigned by the user to a value that may be assumed by a data item. A condition-name must be formed according to the rules for data-name formation. A level 88 entry must be preceded either by an elementary item or by another level 88 entry (in the case of several consecutive condition-names pertaining to 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.

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 the "Procedure Division" chapter under the subsection "Test-Conditions.")

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable. In the following example, PAYROLL-PERIOD is the conditional variable. The picture associated with it limits the value of the 88 condition-name to one digit.

```
02 PAYROLL-PERIOD PICTURE IS 9.
    88 WEEKLY VALUE IS 1.
    88 SEMI-MONTHLY VALUE IS 2.
    88 MONTHLY VALUE IS 3.
```

TYPES OF DATA ITEMS

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

```
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 REDEFINES SALARY
   PICTURE 9V999, DISPLAY.
02 PER-DIEM, REDEFINES RATE
   PICTURE 9999, DISPLAY.
Procedure Division Portion:
IF HOURLY COMPUTE GROSS = 40 * RATE
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

F ONLY

For COBOL F, the maximum length for a group or elementary item is 32,767 bytes, except for a fixed-length Working-Storage or linkage section group item, which may be as large as 131,071 bytes.

E ONLY

For COBOL E, the maximum length for any group item or elementary item is 4,092 bytes, except for a fixed-length Working-Storage group item, which may be as long as 32,767 bytes.

Group Items

A group item is defined as one having further subdivisions, that is, 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 immediately succeeding item has a level number of 88. If an item is not a group item, it is an elementary item, or, in the case of level 88, it is a condition-name.

Elementary Items

An elementary item is a data item containing no subordinate items. For example, an 03 level followed immediately by another 03 level is an elementary item.

Alphabetic Item

An alphabetic item may contain any combination of the characters A through Z and the space. Each alphabetic character is stored in a separate byte. (A byte is the smallest addressable unit of storage in System/360; it consists of eight binary digits.)

Alphanumeric Item

An alphanumeric item consists of any combination of characters in the IBM Extended Binary Coded Decimal Interchange Code 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 Clause" and "BLANK Clause"), except P and V which occupy no storage, and CR and DB which occupy two bytes each.

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 input/output units. 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.

Examples of external-decimal items are shown in Figure 4.

Internal-Decimal Item: An internal-decimal item consists of numeric characters 0 through 9 plus a sign, and represents a value not exceeding 18 digits in length. It appears in storage as packed decimal. One byte contains two digits with the low-order byte containing the low-order digit followed by the sign of the item. 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.

Examples of internal-decimal items are shown in Figure 4.

Binary Item: A binary item may be considered as consisting of numeric characters 0 through 9 plus a sign. It occupies two bytes (a halfword), four bytes (a fullword), or eight bytes (a doubleword), corresponding to specified decimal lengths of one to four digits, five to nine digits, and 10 to 18 digits, respectively. The leftmost bit of the reserved area is the operational sign.

If the item is used as a resultant data-name in an arithmetic statement and no SIZE ERROR option has been specified, the area may be set to a number greater than that specified in the PICTURE clause. 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 a binary item is shown in Figure 4.

Item	Value	Description	Internal Representation*
External-Decimal	-1234	DISPLAY PICTURE 9999	|Z1|Z2|Z3|F4| Byte
		DISPLAY PICTURE S9999	|Z1|Z2|Z3|-4| Byte
Internal-Decimal	+1234	COMPUTATIONAL-3 PICTURE 9999	|01|23|4F| Byte
		COMPUTATIONAL-3 PICTURE S9999	|01|23|4+| Byte
Binary	+1234	COMPUTATIONAL PICTURE S9999	|0000 0100|1101 0010| S Byte
External Floating-Point	+12.34E+2	DISPLAY PICTURE 99.99E-99	|+|1|2|.|3|4|E|b|0|2| Byte
Internal Floating-Point		COMPUTATIONAL-1	|S|Characteristic|Fraction 0 1 7 8 31
		COMPUTATIONAL-2	|S|Characteristic|Fraction 0 1 7 8 63

*Codes used in the Internal Representation column:
Z = zone
Hexadecimal F = nonprinting plus sign
S = the sign position of a numeric field: internally a '1' in position S means the number is negative; whereas a '0' in position S means the number is positive.
b = blank

Figure 4. Internal Representation of Numeric Items

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).

External Floating-Point Item: An external floating-point item consists of a combination of the characters plus (+), minus (-), blank, decimal point (.), the character E, and digits 0 through 9 appearing in a specific format that 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 Option" under PICTURE Clause). Each character of the PICTURE, except V, represents a single byte of storage reserved for the item. The PICTURE of an external floating-point item includes the letter E (see the example under the format of an "External Floating-Point Item" and refer also to "Fp-form Option" under PICTURE

clause). The display of an external floating-point item includes the E (which denotes the exponent) in the printout.

An example of an external floating-point item (literal) is shown in Figure 4.

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 4.

FILE SECTION

The File Section of the Data Division describes the logical characteristics of the files, and the organization of areas used for receiving input or output data.

A file comprises one or more blocks on input/output devices. A block may be described by the programmer as comprising one entity (one logical record), or comprising a group of smaller entities (logical records). A buffer is the area into which a block is read or from which a block is written.

The descriptions of the kinds of logical records that may be contained in a block are specified in the File Section by a level 01 Record Description entry and by the entries subordinate to it. A block may contain one or more logical records, each of which may conform to any of the descriptions specified for the records in the file.

The term volume is a term for a unit or reel on which a file is recorded such as a reel of magnetic tape or a disk pack. In this context, volume and the COBOL reserved words UNIT or REEL are identical in meaning.

Record Formats

The operating system's Data Management defines three record formats to be used: format F, format V, and format U. A file used within the operating system has records that are either fixed (F), variable (V) or unspecified (U).

1. A file with format F records is one in which the size of all the logical records in the file is fixed, and in which logical records are not preceded by a control word.

2. A file with format V records is one in which the sizes of the logical records are not necessarily the same. Each logical record is preceded by a control word indicating the size of the particular logical record. This control word must not be described in any record description entry and cannot be referred to by the user.

3. A file with format U records is one in which the sizes of the logical records are not necessarily the same. Unlike format V records, there is no control word preceding the logical records indicating the size of the record. Files with format U records are considered by the COBOL compiler to contain one logical record per block. The READ statement makes one block available for processing; if there is more than one logical record per block, the user must do his own deblocking.

The choice of record format, which is specified in COBOL via the RECORDING MODE clause, is dependent on the record descriptions. Files for which there is only one record description with an unchanging size (that is, no entry in the record description has an Option 2 OCCURS clause) or for which all record descriptions indicate the same unchanging size may have format F or format V records.

The sizes of the logical records of a file may vary (1) if there is more than one data record description for the file so that the size of each data record described may differ or (2) if one or more elements within the file is described with an Option 2 OCCURS clause. In the latter case, the size of the same logical record may vary from the execution of one READ or WRITE statement of the record to the next. These files may have records of format V or format U.

DATA DIVISION ENTRY FORMATS

This section describes the File Section entries and the Record Description entries within the Data Division. Each of the two subjects is further subdivided into its components.

FILE SECTION ENTRIES

This subsection briefly describes the File Description (FD)and the Sort Description (SD). General notes are given about these items, followed by a discussion of the clauses that comprise them.

File Description

The following is the format of a File Description entry, which must appear once in the File Section for each file. There may be a number of Record Description entries associated with it.

The clauses associated with each File Description entry may appear in any order. FD must appear in Margin A. All associated clauses must begin in Margin B.

```
FD  file-name

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

    [RECORDING MODE IS mode]

   [BLOCK CONTAINS integer {CHARACTERS}]
                           {RECORDS   }

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

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

  [{REPORT IS } report-name...]1 .
   {REPORTS ARE}
```

Note: If the REPORT clause appears in an FD entry, the DATA RECORD clause is optional.

An example of a File Description entry is:

```
FD FILE-1
     RECORD CONTAINS 20 TO 80 CHARACTERS
     RECORDING MODE IS U
     LABEL RECORDS ARE STANDARD
     DATA RECORD IS FILE-1-RECORD.
```

[F ONLY] Sort Description

The following is the format of a Sort Description entry. The clauses associated with each entry may appear in any order. SD must appear in Margin A. All associated clauses must begin in Margin B.

```
SD  sort-file-description-name

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

    [RECORDING MODE IS mode]

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

An example of a Sort Description entry is:

```
SD SORT-FILE-1
     DATA RECORD IS SORT-RECORD
     RECORD CONTAINS 80 CHARACTERS
     RECORDING MODE IS F.
```

For further discussion of the Sort Description entry see the chapter "Sort Feature."

File Section Notes

The following additional information should be noted:

1. The FD entry must describe each data file to be processed by the object program.
2. The SD entry is used to describe each sort-file used in the program. See the chapter "Sort Feature."
3. File-name and sort-file-description-name are the highest-level qualifiers for their respective Record Description entries.
4. If more than one level 01 description is given under an SD or FD entry, all descriptions following the first are considered to implicitly redefine the first.

---

[1]Implemented for COBOL F only

Clauses

The following are the formats and descriptions of each of the clauses that make up file and sort descriptions.

LABEL RECORDS Clause:

The LABEL RECORDS clause specifies the presence of standard or nonstandard labels on a file, or the absence of labels. The format of this clause is:

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

The file LABEL specifications are specified by the LABEL parameter of the DD card. The COBOL equivalents of the four options of the LABEL parameter are as follows:

1. NL (no labels)--OMITTED
2. NSL (nonstandard labels)--OMITTED
3. SL (standard labels)--STANDARD

Use of the LABEL RECORDS clause in a COBOL source program does not eliminate the necessity of the LABEL parameter in the DD card or modify the operating system's Data Management restrictions.

The OMITTED option must be specified for files assigned to unit record devices. It may be specified for files assigned to magnetic tape units.

The STANDARD option must be specified for files with indexed, direct, or relative organization. It may be specified for any files with standard sequential organization, except for files assigned to unit record devices. The system will bypass user labels appearing in the file if the STANDARD option is specified.

Figure 5 is a chart showing available options and their valid use. An "X" in the figure indicates that the option is permitted.

RECORDING MODE Clause:

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

```
RECORDING MODE IS mode
```

Mode must be one of the three alphabetic characters F, U, or V. Each indicates the specification of a record format.

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 clause with the DEPENDING ON option is written in a Record Description entry for the file. If more than one data record description is given following the FD entry, and these descriptions are not all of the same length, the record length of the longest one is used for input/output operations such as WRITE, READ, and blocking.

The V mode (variable-length format) may be specified for any combination of record descriptions. In this format, each logical record is preceded by a control field specifying the length of the logical record.

The U mode (unspecified format) may be used with any combination of record descriptions. Format U records are like format V records that are not blocked, except that they are not preceded by a count control field.

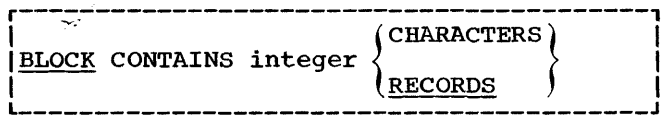
LABEL RECORDS ARE options	Device					
	Unit Record	Tape	Direct-Access Storage Device Organization is:			
			Standard Sequential	Indexed Sequential	Direct	Relative
OMITTED	X	X				
STANDARD		X	X	X	X	X

Figure 5. Relation Between Labels and Device Assignment

When the RECORDING MODE clause is omitted, RECORDING MODE IS V is assumed.

BLOCK CONTAINS Clause:

The BLOCK CONTAINS clause specifies the number of characters or the number of logical records in a block. The format of this clause is:



The BLOCK CONTAINS clause must not be written for files with direct or relative organization or when format U records are used. When the RECORDS option of the clause is used, the assumption is made that the block size provides for integer records of maximum size. The compiler then provides for additional space for any required control words.

When the RECORDS option is not written, the compiler assumes that integer specifies the number of characters contained in the block.

To determine the number of characters to be specified in the BLOCK CONTAINS clause, the following procedures should be used:

Output Files with Format F Records: Multiply the logical record length by the number of records to be contained in the block. For example, if five card images (logical record length of 80)are to be blocked, BLOCK CONTAINS 400 or BLOCK CONTAINS 400 CHARACTERS could be specified.

Output Files with Format V Records: The minimum value of integer must equal the size of the largest logical record defined for the file and must include the 4-byte count field that precedes each format V record, but not the 4-byte count field that precedes the block. (The latter count field is automatically generated at the time the program is being executed.) Note, too, that if the file contains records with COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 entries, it is the programmer's responsibility to add any necessary inter-record slack bytes. These slack bytes are part of the record description and must be included in the value of integer. (See Appendix B, "Slack Bytes.")

Therefore, if two types of records are to be written, one 400 characters long and the other 200 characters long, the minimum integer that can be specified in the CHARACTERS option of the BLOCK CONTAINS clause is 404 (BLOCK CONTAINS 404 or BLOCK CONTAINS 404 CHARACTERS).

It should be noted that Option 3 of the APPLY clause (APPLY WRITE-ONLY) is used to make optimal use of buffer space allocated for a file with format V records. In the above example, if a record 200 characters long was placed in the block specified, there would not be enough space allocated for another record even if the next record was also 200 characters long, because the 4-byte count field preceding each format V record could not be accommodated. Therefore, given the above facts, the programmer should at least specify BLOCK CONTAINS 408 and use the APPLY WRITE-ONLY option. If the APPLY WRITE-ONLY option is not specified for a file, the buffer is truncated and the block is written out whenever the space remaining in the buffer is not sufficient for the maximum size record (400 characters in the above example) defined for the file.

The programmer can specify how many maximum size format V records are to fit into a block by means of the RECORDS option of the BLOCK CONTAINS clause. The compiler uses the value of integer to compute the length of the block by multiplying the length of the longest record by integer, adding enough space to accommodate a 4-byte count field for the block and a 4-byte count field for each record. Therefore, if two types of records are to be written, one 400 characters long and the other 200 characters long, and if the programmer specifies BLOCK CONTAINS 3 RECORDS, the compiler reserves a block of 1216 characters. Depending on the actual size of the records written, more than integer records may be contained in the block. Given the above facts, it is possible for a block to contain five 200-character records (5*204+4<1216).

Note, however, in the case where there are two or more OCCURS...DEPENDING ON... clauses in the record and the programmer specifies RECORD CONTAINS integer-1 TO integer-2 CHARACTERS, the compiler determines the record length from integer-2.

Input Files with Format F Records: The value of integer for input files with format F records should be the same as the value when the file was used as an output file (see above), provided that the same option of the BLOCK CONTAINS clause is specified. However, for QSAM files only, it is possible to specify an integral multiple of the record length specified when the block was created.

Input Files with Format V Records: The value of integer for input files with format V records is the same as the value determined when the file was used as an output file, provided that the same option of the BLOCK CONTAINS clause is specified.

However, if the RECORDS option was used for an output file and the programmer desires to use the CHARACTERS option on the same file when used as input, he must perform the same calculations for the value of integer as the compiler does when the file is created, i.e., he must take the 4-byte count field preceding each record into consideration. Thus, if RECORD CONTAINS 5 TO 10 CHARACTERS and BLOCK CONTAINS 5 RECORDS were specified for output, RECORD CONTAINS 70 CHARACTERS (5*(10+4)) must be specified for input. This rule also applies when the order of the options is reversed, because the value of integer for the RECORDS option does not include the 4-byte count field for each record, whereas it does for the CHARACTERS option.

When the BLOCK CONTAINS clause is omitted, the file is considered to be unblocked.

The value of integer must never exceed the maximum specified for a device type.

The value of integer (within the RECORDS option of the BLOCK CONTAINS clause) may be zero and the blocksize specified in the DD card (or JFCB if it is an input file) at execution time if the following conditions occur:

1. The file is QSAM (F or V) and the SAME AREA clause is not specified, or
2. For QISAM (F or V), and no SAME AREA clause is specified. In this case the QISAM scan mode FD must also specify integer to be zero if the BLOCK CONTAINS clause is zero. The BISAM FD used for this file must specify the maximum blocksize the file can contain.

Note: The blocking factor for a QISAM output file must equal the blocking factor for the same file when it is used as input or I-O. The blocking factor must also remain the same if it is used as a BISAM input or I-O file. This restriction also holds if the blocksize is specified at object time via a DD card rather than at compile time in an FD entry.

RECORD CONTAINS Clause:

The RECORD CONTAINS clause specifies the sizes of the logical records contained in a file. The format of this clause is:

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

When this clause is written, integer-1 indicates the size of the smallest record described for the file, and integer-2 indicates the size of the largest record. Whether this clause is specified or omitted, the record lengths are determined by the compiler from the record descriptions unless more than one of the entries contains an OCCURS...DEPENDING ON... clause. If any of the entries contains this form of the OCCURS clause, the maximum value of the variable in that clause is used in calculating the record length. If more than one of the entries contains this form of the OCCURS clause and the maximum values of the variable in those clauses do not occur simultaneously, integer-2 may specify a maximum record size other than the size calculated from the maximum value of the variables.

DATA RECORDS Clause:

The DATA RECORDS clause specifies the names of the logical records in a file. The format of this clause is:

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

Record-name is a data-name described with an 01 level-number following the FD entry in the File Section.

F ONLY

REPORT Clause:

The REPORT clause is required in the File Description entry only when the Report Writer feature is utilized, and the file is either an output report file or is used to contain output report records.

The format of the REPORT clause is:

```
{REPORT  IS } report-name...
{REPORTS ARE}
```

The appearance of two or more report-names in this clause indicates that the file contains more than one report. These reports may be of different sizes and formats, and the order in which they are described in the Data Division is not significant.

Each report-name listed in the FD entry must have an RD (Report Description) entry in the Report Section of the Data Division. (Complete details concerning the Report Writer feature are contained in the chapter "Report Writer Feature.")

The REPORT clause may be specified only for files whose organization is standard sequential.

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} [redefines-clause]
             {FILLER   }

    [PICTURE-clause]    [BLANK-clause]

    [OCCURS-clause]     [VALUE-clause]

    [JUSTIFIED RIGHT]   [USAGE-clause].
```

When this format is applied to 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 not shown in a format are forbidden. Clauses that are mandatory in the description of certain data items are written without brackets.

Group Item Format

The format of the Record Description entry for a group item is:

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

   [OCCURS-clause] [USAGE-clause].
```

Sample coding for a group item and its associated subordinate (elementary) items is:

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

Note: A group item, by definition, must have items (FIELD-B and FIELD-C in the above example) that are subordinate to it. An item is subordinate to another by having a level number that is higher than the immediately preceding item. FIELD-B and FIELD-C have 02 level numbers, whereas GROUP-ITEM has an 01 level number.

Formats for Elementary Items

An elementary item is one having no items subordinate to it.

Alphabetic Item:

The format of the Record Description entry for an alphabetic item is:

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

    [OCCURS-clause] PICTURE IS alpha-form

    [USAGE IS DISPLAY]

    [VALUE IS alphabetic-literal]

    [JUSTIFIED RIGHT].
```

Sample coding for an elementary alphabetic item is:

```
02 EMPLOYEE-NAME PICTURE A(20).
```

Alphanumeric Item:

The format of the Record Description entry for an alphanumeric item is:

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. It should be noted, however, that if both of the above statements are executed successively in the order specified, the value Y will overlay the value X. However, redefinition itself does not cause any data to be erased and does not supersede a previous description.

Altering the USAGE of an area through redefinition does not cause any change in existing data. Consider the example:

```
02  B  PICTURE 99 USAGE DISPLAY
       VALUE IS 8.
02  C  REDEFINES B PICTURE 99
       USAGE COMPUTATIONAL-3.
```

The bit configuration of the value 8, when used as a display item, is 1111 0000 1111 1000. Redefining B does not change its appearance in storage. Therefore, a great difference results from the two statements, ADD B TO A and ADD C TO A. In the former case, the value 8 is added to A because B is a display item. In the latter case, the bit configuration does not represent a valid internal-decimal (COMPUTATIONAL-3) number and the results of the addition are invalid.

Moving a data item to a second data item that redefines the first one (for example, MOVE B TO C when C redefines B), may produce results that are not those expected by the programmer. The reverse (MOVE B TO C when B redefines C) is also true.

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. However, the REDEFINES clause may appear in 01 levels in the Working-Storage Section. The level number of data-name-2 must be identical to that of the item containing the REDEFINES clause.

The entries giving the new description of the area must immediately follow the entries describing the area being redefined. The description of an area can mean a group item and all associated elementary items. However, in the case where more than one Record Description entry is redefining the same entry, these additional entries may intervene. For example both of the following are valid uses of the REDEFINES clause:

```
02  ARRAY-1 DISPLAY.
      03  A PICTURE X(2).
      03  B PICTURE X(2).

02  ARRAY-2 REDEFINES ARRAY-1
      USAGE COMPUTATIONAL-1.

02  A PICTURE 9999.
02  B REDEFINES A PICTURE 9V999.
02  C REDEFINES A PICTURE 99V99.
```

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 which are themselves redefined. The following would therefore be a valid use of the REDEFINES clause:

```
02  REGULAR-EMPLOYEE DISPLAY.
      03  LOCATION PICTURE A(8).
      03  STATUS PICTURE X(4).
      03  SEMI-MONTHLY-PAY
          PICTURE 9999V99.
      03  WEEKLY-PAY REDEFINES SEMI-
           MONTHLY-PAY PICTURE 999V999.

02  TEMPORARY-EMPLOYEE REDEFINES
     REGULAR-EMPLOYEE DISPLAY.
      03  LOCATION PICTURE A(8).
      03  FILLER PICTURE X(6).
      03  HOURLY-PAY PICTURE 99V99.
```

REDEFINES clauses may also be specified for items subordinate to items containing REDEFINES clauses. For example:

```
02  REGULAR-EMPLOYEE DISPLAY.
      03  LOCATION PICTURE A(8).
      03  STATUS PICTURE X(4).
      03  SEMI-MONTHLY-PAY
          PICTURE 9999V99.
      03  WEEKLY-PAY REDEFINES SEMI-
           MONTHLY-PAY PICTURE 999V999.

02  TEMPORARY-EMPLOYEE REDEFINES
     REGULAR-EMPLOYEE.
      03  LOCATION PICTURE A(8).
      03  FILLER PICTURE X(6).
      03  HOURLY-PAY PICTURE 99V99.
      03  PAY-CODE REDEFINES HOURLY-
           PAY PICTURE 9999.
```

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

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

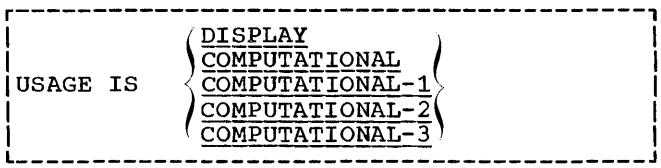
data-name-2 may be described by an OCCURS clause with a DEPENDING ON option.

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

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. For this reason, when representing internal-decimal, binary, or internal floating-point items, the USAGE clause must be specified. The format of the USAGE clause is:



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.

For example, if

```
02 FICA PICTURE IS S999V99 COMPUTATIONAL.
```

(a decimal length of five) is specified, the binary data item will occupy four character positions. The leftmost bit of the reserved area is the operational sign. Computational items are aligned at the next halfword or fullword boundary, as appropriate.

The COMPUTATIONAL-1 option specifies a data item stored in short-precision floating-point format, aligned on the next fullword boundary. The COMPUTATIONAL-2 option specifies a data item stored in long-precision floating-point format, aligned on the next doubleword boundary. The COMPUTATIONAL-3 option specifies that the item is stored in packed (internal) decimal format: two digits per character position, with the low-order half-character containing the sign.

If a data hierarchy contains binary or floating-point items intermixed with other elementary items, slack bytes may be present. Slack bytes are introduced to assure the necessary byte alignment.

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 full discussion of slack bytes is contained in the appendix entitled "Slack Bytes."

Table 3 shows the requirements pertaining to binary and floating-point items. m is the amount by which the address of the first byte of the item must be divisible.

Table 3. Values for Calculating Record Length

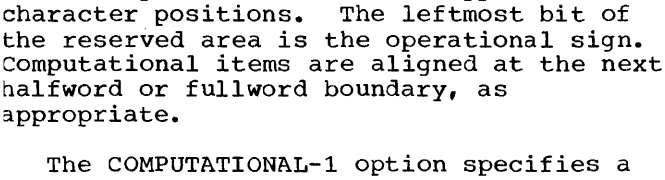
Type of Data Item	Value of m
Binary halfword	2
Binary fullword	4
Binary doubleword	4
Floating point, short	4
Floating point, long	8

When an item of the types listed in Table 3 immediately follows an item whose last byte is at X relative to the first byte of the COBOL record, then the first byte of this item is located at X+n, where n is the number which must be added to X in order to make X+n a multiple of m.

The number of slack bytes (n), may be zero or a positive number less than 8.

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:



F ONLY

In the following discussion of the PICTURE clause, the rules for the period apply to the comma, and the rules for the

---

[1]Extension

comma apply to the period, if the clause DECIMAL-POINT IS COMMA is present in the Special-Names paragraph of the Environment Division.

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 always contains 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 may contain any character from the Extended Binary Coded Decimal Interchange Code 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 scaling 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 999PPPV; 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. The presence of S is required where the USAGE clause is COMPUTATIONAL, since a sign is always present in a binary item. For internal and external decimal items, the presence of an S in the PICTURE indicates that the item contains a sign, i.e., the item can assume positive or negative values. The absence of an S in the PICTURE indicates that the item is to be treated as an absolute value, i.e., the item can assume only positive values. Its sign position is then occupied by a bit configuration which is interpreted as positive, but 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 as follows:

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 as follows:

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 actual decimal

Character	Meaning
	point or a V. Z may appear to the right of an actual decimal point only if all digit positions are represented by Z's. If all the digit positions are represented by Z's and the value of the data is zero, the entire data item will be blanks. A Z cannot appear anywhere to the right of a 9. The PICTURE ZZZ.ZZ is equivalent to a combination of the BLANK clause and the PICTURE ZZZ.99.
*	Each asterisk in a PICTURE represents a digit position. If a digit position contains a leading zero, but is defined by *, the zero is replaced by an asterisk. An asterisk replaces each digit position so defined until the actual decimal point or a V is encountered. Asterisks may appear to the right of an actual decimal point only if all digit positions are defined by asterisks. If all the digit positions are defined by asterisks, and the value of the data is zero, the entire data item, except for the actual decimal point, will be asterisks. If the decimal point character is present, the actual decimal point will always appear. An asterisk 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, respectively, 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 are insertion characters specifying insertion of comma, zero, or 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 suppression of leading insertion characters, with replacement by

Character	Meaning
	spaces or asterisks except for the insertion characters appearing to the left of the first Z or *. The comma, zero, and B may also appear in conjunction with a floating string.

A floating string is defined as a leading, continuous series of either $ or + or -, or a string composed of one, or a repetition of one such character interrupted by one or more insertion characters (, 0 B) and/or V or an actual decimal point. For example:

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

A floating string containing n + 1 occurrences of $ or + 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 + or - immediately to the left of the most significant nonzero digit, in one of the positions indicated by $ or + or - in the PICTURE. Blanks are placed in all character positions to the left of the single developed $ or + 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 + 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 non-zero, 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 or zero in a floating string operates similarly, except that the character comma or zero appears in the developed item instead of a blank.

Character	Meaning
$ + -	The character $ or + 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 contain asterisks.

4. The appearance of a floating string or fixed plus or minus sign insertion characters precludes the appearance of any other of the sign control insertion characters, namely, + - CR DB.

5. The characters in a PICTURE to the left of an actual or assumed 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 $ or Z or * then all the numeric character positions in the PICTURE must be represented by the same character.

7. A floating string must begin with at least two consecutive appearances of the + or - or $ character.

8. The PICTURE character 9 can never appear to the left of a floating or replacement character.

9. Floating or replacement characters + - Z $ * 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 the mantissa, with a leading, embedded, or trailing decimal point or V

3. The letter E, which denotes the exponent, occupies one byte of storage and is included in the printout

4. + or - (same as point 1 above)

5. Two 9s representing the exponent

Additional Notes on the PICTURE Clause

The following considerations pertain to use of the PICTURE clause.

1. A PICTURE clause may be used only 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 two consecutive appearances of the + or - or $ character.

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. Except for a floating-point item, a report item can possess only one sign.

The examples in Table 4 illustrate the use of PICTURE to edit data. In each example, a movement of data is implied, as indicated by the column headings.

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 be used only for report items specified at an elementary level.

The format of the BLANK clause is:

```
BLANK WHEN ZERO
```

Table 4. Editing Applications of the PICTURE Clause

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
S9(5)V	00000	-ZZ,ZZ9.99	0.00
9(5)	00000	$ZZ,ZZZ.99	$ .00
9(5)	00000	$ZZ,ZZZ.ZZ	
999V99	12345	$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	$**,***.99	$******.00
9(5)	00000	$**,***,**	**********
99V999	12345	$**,**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	12345	$$$,$$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)	00123	------.99	123.00
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	99BB999	12 345
9(5)	12345	9900999	1200345
S9(5)	-12345	ZZZZZ.99CR	12345.00CR
S9(5)	12345	$$$$$.99CR	$12345.00

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, as specified in the data item formats earlier in this text.

If the literal specified is a figurative constant, the size of the item generated is the size specified in the PICTURE clause.

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 appear only 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. The VALUE clause cannot be used to specify the initial value of an item following a variable portion of a Working-Storage record defined by the DEPENDING option of the OCCURS clause.

OCCURS Clause

The OCCURS clause is used in defining related sets of repeated data, such as tables, lists, vectors, matrixes, 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. The subject of an OCCURS clause is the data-name described by this clause and represents a table element. A table element must be fewer than 32,767 bytes. The subject of the OCCURS clause must be subscripted whenever it is referred to in any procedure division statement. Further, if the subject of the OCCURS clause is the name of a group item, all data items within 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. Option 1 of the OCCURS clause has the following format:

Option 1

```
OCCURS integer TIMES
```

Integer represents the exact number of occurrences.

Option 2 of the OCCURS clause has the following format:

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 or zero. 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.

When a record description contains more than one Option 2 OCCURS clause, the record items affected by OCCURS DEPENDING ON should be initialized prior to making a reference to the record.

Option 2 may be used in COBOL (E) with the following restrictions:

1. Only one OCCURS 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 rec-

ord, or in the description of the last elementary item itself.

3. The item having an OCCURS clause with a DEPENDING ON option must not itself be contained in a group having any OCCURS clause.

During the course of program execution, if the value of data-name changes, the following should be noted:

1. The size of any group described by or containing the related OCCURS clause will reflect the new value of data-name.

2. [F ONLY] If the item described with the OCCURS clause is followed by any nonsubordinate items, their location will be affected by the new value of data-name.

Note: Data-name can also change because of a change in the value of an item that redefines it. In this case, the group size and the location of nonsubordinate items as described in the two preceding paragraphs cannot be determined.

For more information on the use of the OCCURS...DEPENDING ON clause, see the following publications: IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form GC28-6380, and IBM System/360 Operating System: COBOL (E) Programmer's Guide, Form GC24-5029.

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 represented by a data-name must be an internal decimal, external decimal, or binary item.

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. A space should also separate the data-name from the subscript expression. The following coding would result in a storage layout as shown in Figure 6.

```
01  TABLE-A.
  02  ARRAY OCCURS 2 TIMES.
    03  VECTOR OCCURS 2 TIMES.
      04  ELEMENT OCCURS 3 TIMES USAGE
          COMPUTATIONAL PICTURE S9(9).
```

TABLE-A contains three levels of subscripting. Reference to elementary items are made by subscripted name. In the Procedure Division a typical statement might be:

```
ADD ELEMENT (2, 1, 3) TO SUM.
```

A data-name may not be subscripted when it is being used as:

1. A subscript
2. A qualifier
3. The defining name of a record description entry
4. Data-name-2 in a REDEFINES clause
5. A data-name in the DEPENDING ON option of the OCCURS clause
6. A data-name in a RECORD KEY, SYMBOLIC KEY, or ACTUAL KEY clause
7. A data-name in a LABEL RECORDS clause
8. The data-name option of a TRACK-AREA clause

Subscripting A Qualified Data-Name

Qualification is necessary when the same data-name is used for several different items of data; subscripting is necessary when some of the elements of a table or list have not been assigned individual names.

In subscripting a qualified data-name the following rules are significant:

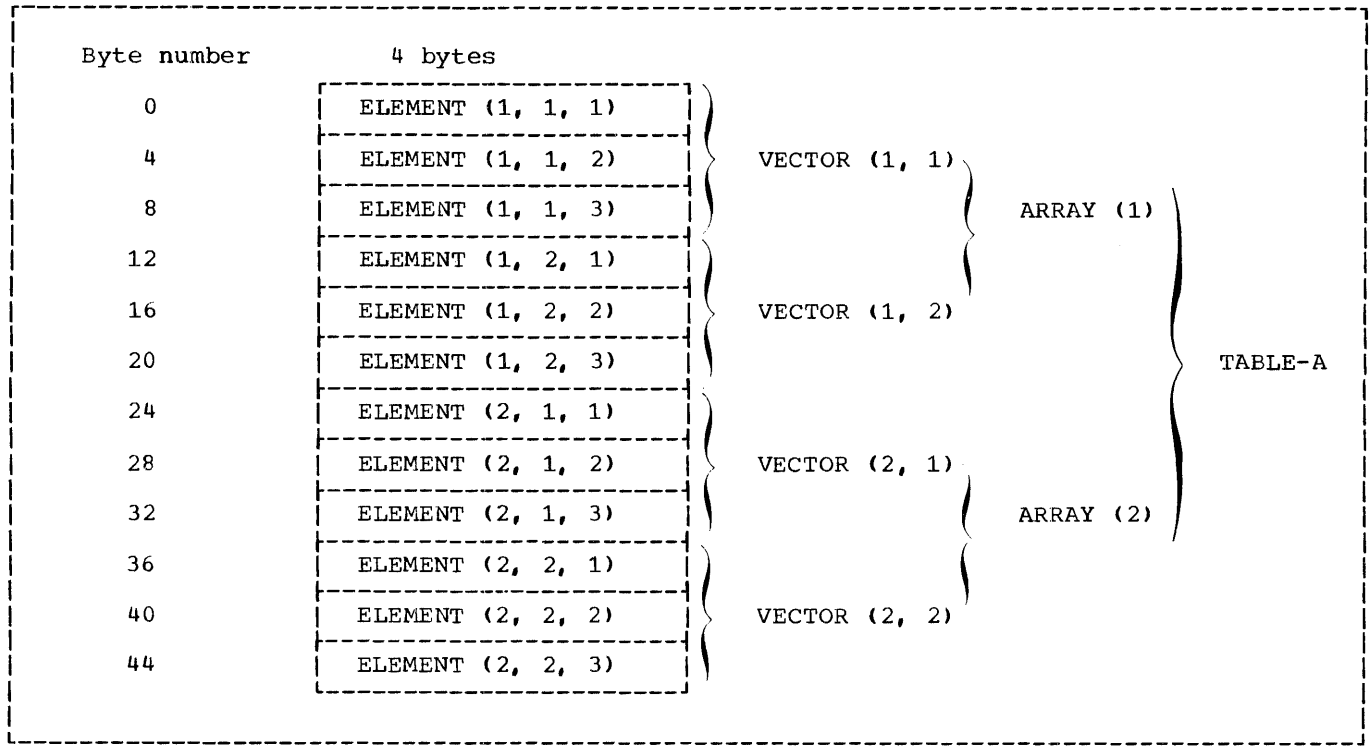


Figure 6. Storage Layout for Subscripting Example

1. A data-name can be qualified even though it does not need qualification to make it unique.

2. Data-names used as qualifiers are considered part of the data-name being qualified and cannot be subscripted. In the example below, the higher level data-names 'VECTOR' and 'ARRAY' are used to qualify the data-name 'ELEMENT', forming the new data-name 'ELEMENT IN VECTOR IN ARRAY' followed by its subscripts. Neither 'VECTOR' nor 'ARRAY' can be subscripted.

As a result of these rules there are several correct ways of expressing subscripted data-names. For example, referring to Figure 6, the following expressions are all correct references to the 2nd ELEMENT in the 2nd VECTOR in the 1st ARRAY:

```
ELEMENT IN VECTOR IN ARRAY (1, 2, 2)
ELEMENT IN VECTOR (1, 2, 2)
ELEMENT IN ARRAY (1, 2, 2)
ELEMENT (1, 2, 2)
```

The first three of these are examples of unnecessary, although permissible, qualification assuming that ELEMENT and VECTOR occur only in this hierarchy. However, if the name ELEMENT is used elsewhere, the qualification must be used. Note that the following forms of expression are incorrect:

```
ELEMENT (1, 2, 2) IN VECTOR IN ARRAY
ELEMENT (2) IN VECTOR (2) IN ARRAY (1)
ELEMENT (2) IN VECTOR (1, 2)
```

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 data is so aligned that rightmost source characters are accommodated in the rightmost positions of the receiving field. If the receiving field is shorter than the source field an appropriate number of leftmost source characters are truncated. If the receiving field is longer than the source field excess leftmost positions in 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.

[EXT] LINKAGE SECTION

The Linkage Section describes data passed from another program.

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 doubleword boundary.

The Linkage Section is required in any program in which an ENTRY statement with a USING option appears. A complete discussion of the ENTRY statement is contained in the chapter entitled "Procedure Division."

Figure 10 is a flowchart indicating the logical flow of the conditional statement in Figure 9.

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 which when preceded by the word IF, constitute one of the five types of tests: relation test, sign test, class test, condition-name test, and 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.

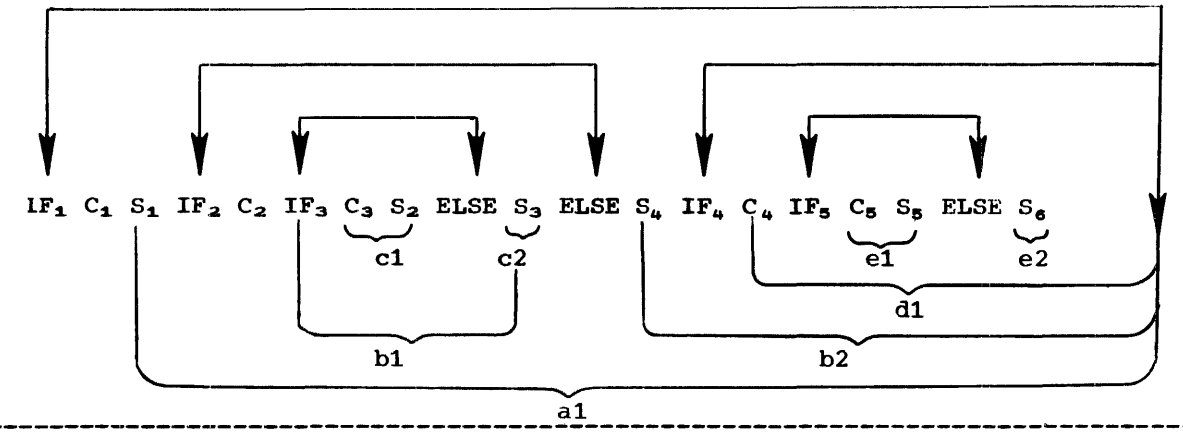
Relation Test

A relation test involves the comparison of two operands, either of which can be a data-name, a literal, or an arithmetic expression. Neither the comparison of two literals nor the comparison of an arithmetic expression to a non-numeric data-name is 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:

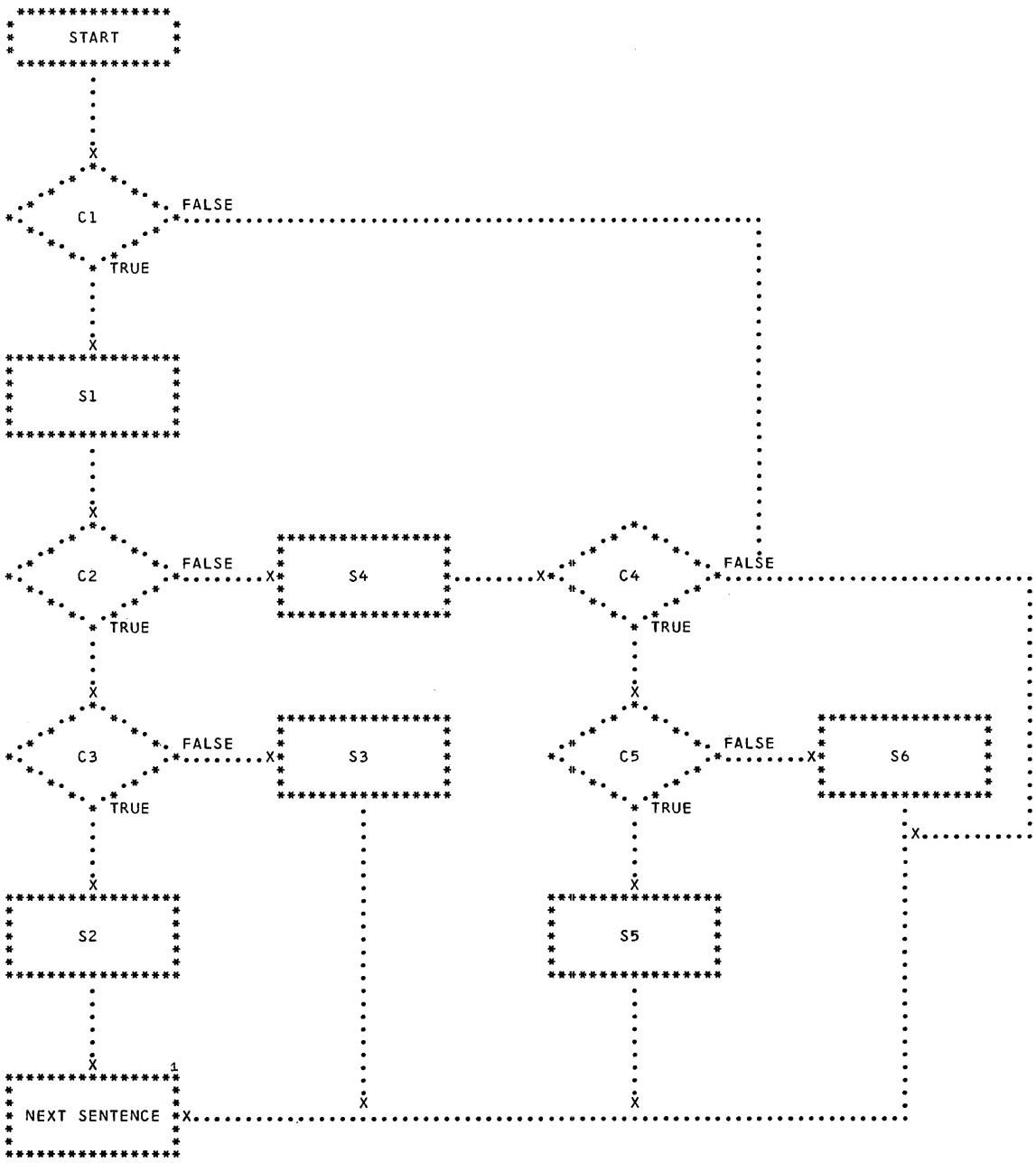
```
{data-name-1              }
{arithmetic-expression-1  } IS [NOT]
{figurative-constant-1    }
{literal-1                }

{>                  }   {data-name-2              }
{<                  }   {arithmetic-expression-2  }
{=                  }   {figurative-constant-2    }
{GREATER            }   {literal-2                }
{LESS THAN          }
{EQUAL TO           }
```



a1--Statement-1 for IF_1 (If C_1 is false, the next sentence is executed, since there is no ELSE for it.)
b1--Statement-1 for IF_2
b2--Statement-2 for IF_2
c1--Statement-1 for IF_3
c2--Statement-2 for IF_3
d1--Statement-1 for IF_4 (If C_4 is false, the next sentence is executed, since there is no ELSE for it.)
e1--Statement-1 for IF_5
e2--Statement-2 for IF_5

Figure 9. Conditional Statements with Nested IF Statements



[1]Control flows to the next sentence unless otherwise directed

Figure 10. Logical Flow of Conditional Statement with Nested IF Statements

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. In the statement, IF SALES EQUAL TO QUOTA GO TO A, the relation test SALES EQUAL TO QUOTA would be evaluated as follows:

Data-name	PICTURE	Value at time of compare
SALES	9999V99	212.00
QUOTA	999	212

Evaluation is TRUE.

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 EBCDIC set.

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. In the statement, IF ALPHA EQUAL TO BETA GO TO A, the relation test ALPHA EQUAL TO BETA would be evaluated as follows:

Data-name	PICTURE	Value at time of compare
ALPHA	X(4)	0212
BETA	X(3)	212

Evaluation is FALSE.

Consider the following example (lower case b indicates blank):

Data-Name	PICTURE	Value at time of compare
ALPHA	X(5)	ABCDE
BETA	X(7)	ABCDEbb

Evaluation is TRUE.

Table 5 indicates the characteristics of the items being compared and the type of comparison made. A blank box in Table 5 indicates that the test is not permitted.

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:

```
{data-name              }IS [NOT]{POSITIVE}
{arithmetic-expression  }        {ZERO    }
                                 {NEGATIVE}
```

The following are examples of the use of the SIGN test:

```
PERFORM A UNTIL RESULT IS NEGATIVE.

IF A * B - C IS NOT ZERO GO TO D.
```

Class Test

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

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

The item to be tested must be an elementary item and either alphanumeric, alphabetic, internal decimal, or external decimal. The valid forms of the class test are shown in Table 6.

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 overpunch, 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 whereas the NOT form of the tests will both be considered false. For example, if a 1-character

alphanumeric field called FIELD is being tested and contains the hexadecimal configuration C1, both of the following will be true because hexadecimal C1 could be interpreted either as an A or as a +1:

```
IF FIELD IS ALPHABETIC MOVE 'A' TO CODE-A.

IF FIELD IS NUMERIC MOVE 'N' TO CODE-N.
```

• Table 5. Permissible Comparisons

	Second Operand											
First Operand	GR	AL	AN	ED	ID	BI	EF	IF	RP	FC	SN[5]	SR[5]
Group Item (GR)	NN	NN	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					NN	NN	NN	NN
External Decimal Item (ED)	NN		NN	NU	NU	NU	NU	NU		NU[3] [6]	NU	
Internal Decimal Item (ID)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
Binary Item (BI)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
External Floating-point Item (EF)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
Internal Floating-point Item (IF)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
Report Item (RP)	NN		NN						NN	NN[4]		
Figurative Constant (FC)	NN	NN[1]	NN	NU[3] [6]	NU[2]	NU[2]	NU[2]	NU[2]	NN[4]		NN[3]	NN[4]
Sterling Nonreport Item (SN)[5]	NN		NN	NU	NU	NU	NU	NU		NN[3]	NU	
Sterling Report Item (SR)[5]	NN		NN							NN[4]		NN

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]F only
[6]A non-numeric comparison is generated for the figurative constant ALL 'character' in COBOL F only

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: unary plus or minus, then exponentiation, then multiplication and division, and finally addition and subtraction. Thus, the expression A + B / C + D ** E * F - G is read as 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, consecutive additions or subtractions, or consecutive exponentiation) 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.

The plus and minus signs are the only allowable unary operators. A unary operator is an operator having only one operand. The unary plus or minus sign must be the first character of an arithmetic expression, or must be immediately preceded by a left parenthesis.

COMPILER-DIRECTING DECLARATIVE SECTIONS

Declarative sections are identified by compiler-directing statements that specify the circumstances under which a procedure is to be executed in the object program.

A declarative section consists 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 each be followed by a period. A declarative section is terminated by the occurrence of another section or the words END DECLARATIVES.

Although declarative sections are located at the beginning of the Procedure Division, execution of the object program starts with the first procedure following the termination of the declarative sections.

The general form for a declarative section is:

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

The procedure-branching statements ALTER, GO TO, PERFORM, STOP RUN, and STOP literal can refer to a declarative section or to paragraph names within it or can appear within declaratives. Restrictions on the appearance of these statements are given in Table 18.

┌-------┐
|F ONLY|
└-------┘ The SORT statement should not appear within a declarative section.

USE Sentence

The USE sentence identifies the type of declarative.

There are two options of the USE sentence. Each is associated with one of the following types of procedures:

1. Report-writing procedures
2. Input/output error-checking procedures

┌-------┐
|F ONLY|
└-------┘ Option 1

Option 1 of the USE declarative is used to designate procedures that are to be executed by the Report Writer before the report group specified by data-name is produced; data-name may be the name of any type of report group except DETAIL. The

at for this option of the USE declara-
 is:

```
 BEFORE REPORTING data-name.
```

eport Writer statements must not be
 in procedures associated with this
on of the USE sentence. Further infor-
on on the Report Writer is contained in
chapter entitled "Report Writer
ure."

ll logical paths within the declarative
ion must lead to a common exit point.
 PERFORM statements may refer to
edure-names outside the section.

on 2

ption 2 of the USE declarative is used
rovide user input/output error-
essing procedures in addition to the
edures supplied by the operating sys-
s Data Management. The format of this
on of the USE declarative is:

```
 AFTER STANDARD ERROR PROCEDURE

N file-name.
```

ithin the section, the file associated
 the USE sentence may not be referred
y an OPEN, READ, WRITE, or REWRITE .
 a CLOSE statement may be given for the
.

n exit from this type of declarative
ion can be effected by executing the
 statement in the section (normal
rn), or by means of a GO TO statement.
e 9 summarizes the facilities and limi-
ons associated with each file-
essing technique when an error occurs.

eferences to procedure-names outside
declarative section are permitted, and
mmon exit point is not required.

L VERBS

he COBOL verbs are the basis of the
edure Division of a source program.

he organization of the remainder of
 section is based on the classifica-
s used in the following list:

Input/Output Verbs
- OPEN
- READ
- WRITE
- REWRITE[1]
- CLOSE
- DISPLAY
- ACCEPT

Data Manipulation Verbs
- MOVE
- EXAMINE
- TRANSFORM[1]

Arithmetic Verbs
- COMPUTE
- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE

Procedure-Branching Verbs
- STOP
- GO TO
- ALTER
- PERFORM

Compiler-Directing Verbs
- EXIT
- ENTER
- NOTE

INPUT/OUTPUT STATEMENTS

The COBOL input/output verbs provide the means of storing data on an external device (such as magnetic tape, disk units, etc.) and getting such data from the external devices. The following is a discussion of the verbs associated with these functions.

OPEN Statement

The OPEN statement initiates the processing of both input and output files; its format is:

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

---

[1]Extension

Table 9. Error-Processing Summary

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 (QSAM)	DD card option if one is specified. Otherwise, end of task	READ WRITE REWRITE	DD card option, if one exists. Otherwise, end of task(1)	User limited to CLOSE for file-name
	INDEXED (QISAM)	End of task	READ REWRITE	Processing of The file may yield unpredictable results	
			WRITE	Processing of the file may yield unpredictable results	
	DIRECT (BSAM)		READ	Processing of file is limited to CLOSE	
			WRITE		
	RELATIVE (BSAM)		READ	Processing of file is limited to CLOSE	
			WRITE		
RANDOM	INDEXED (BISAM)	Error is ignored	READ REWRITE	Processing of the file may yield unpredictable results	Processing of the file may yield unpredictable results
			WRITE	Processing of the file may yield unpredictable results	Processing of the file may yield unpredictable results
	DIRECT (BDAM)	End of task	READ WRITE REWRITE	Continued processing of file permitted	Continued processing of file permitted
	RELATIVE (BDAM)				

(1) File must not be closed if normal return is taken.

The OPEN statement must be specified for all files, except sort-files, used by a COBOL program. A sort-file is the file referred to by sort-file-description-name in a SORT statement.[1] The OPEN statement must be executed prior to any other statement referencing a file. A second OPEN statement for a given file cannot be executed until a CLOSE statement without the UNIT or REEL option has been executed for that file.

The OPEN statement does not obtain or release the first data record. A READ or

[1]Implemented for COBOL F only

WRITE statement, respectively, must be executed to perform these functions. A READ must be executed before any other reference can be made to a file opened for input.

At least one of the options (INPUT, OUTPUT, or I-O) must be specified; however, there may be no more than one instance of each option for a given OPEN statement. For example, if two files, AFILE and BFILE, are both to be opened as INPUT, the coding might be:

```
OPEN INPUT AFILE BFILE.
```

The INPUT option initiates Data Management label checking and permits reading the file.

The OUTPUT option initiates Data Management label creation and permits creating the file.

The I-O option permits the opening of a direct-access file for reading, updating, or adding records. It also performs the same label checking functions as those for a file opened as INPUT.

The NO REWIND option has no effect on file positioning at the time a file is opened. It appears in the format for language consistency. When either the NO REWIND option or no option whatever is specified, positioning of the file at the time the file is opened is controlled by the operating system in conjunction with the DD statement associated with the file.

A file may be opened as INPUT and OUTPUT and I-O (with intervening CLOSE statements without the UNIT or REEL option) unless it has both indexed organization and sequential access. A file that has both indexed organization and sequential access may be opened as INPUT and I-O (with intervening CLOSE statements without the UNIT or REEL option).

The REVERSED Option: When the REVERSED option is specified, subsequent READ statements for the file make the data records of the file available in reverse order.

The REVERSED option can be used only for sequential reel (i.e., magnetic tape) processing. When the REVERSED option is specified, the file must be positioned at its end; that is, the file must have been previously closed with the NO REWIND option. Files with non-standard labels should not be opened for reversed reading unless the last label is followed by a tape mark. Otherwise, the system reads labels as if they were data records.

The REVERSED option cannot be used for a file containing type V records. If the option is specified for a file containing type U records, doubleword boundary alignment will be obtained only if the lengths of the logical records are divisible by eight.

An example of an OPEN statement is:

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

Note: Z-FILE is not opened for reversed reading.

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 the end-of-file is detected.

The format for this option of the READ statement is:

Option 1

```
READ file-name RECORD [INTO data-name]
    AT END imperative-statement...
```

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 for this option of the READ statement is:

Option 2

```
READ file-name RECORD [INTO data-name]
    [INVALID KEY imperative-statement...]
```

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 is available in the input area until the next READ statement (or a CLOSE statement) for that file is executed.

If more than one record description is given following the FD-entry for the file, it is the programmer's responsibility to recognize which record is in the input area at any given time.

Regardless of the method used to overlap access time with processing time, an input record is made available by a READ statement prior to execution of the next statement.

The INTO data-name option of the READ statement is equivalent to a READ statement and a MOVE statement. Data-name must be the name of a sort-file description[1], a Working-Storage record, or a previously opened output record. When this option is used, the current record becomes available in the area specified by data-name as well as in the input area. Data is moved into that area in accordance with the rules for the MOVE statement without the CORRESPONDING option.

The AT END clause is required for files for which access is sequential. The imperative statements in this clause are executed when an end-of-file condition is detected.

Once the imperative statements in the AT END clause of a READ statement have 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 for that file have been executed. In particular, when the Report Writer feature[1] is used and REPORT FOOTING refers to the input (e.g., SOURCE from input), a TERMINATE statement following the end-of-file is a violation of the above-mentioned rule. (The reason for this is that TERMINATE, by definition, produces REPORT FOOTING.)

For COBOL F, the INVALID KEY option may be given for files specified as ACCESS IS RANDOM when no error-processing declarative sections have been specified for the files. For COBOL E, the INVALID KEY clause is required for files specified as ACCESS IS RANDOM. The error-processing declarative may also be specified. The statements following INVALID KEY are executed when the contents of the actual key and/or symbolic key are invalid.

The keys are considered invalid under the following conditions:

1. When indexed organization and random access are specified, and no record exists whose RECORD KEY field matches the contents of the SYMBOLIC KEY field.
2. When relative organization and random access are specified, if the relative record number is outside the limits of the file.
3. When direct organization and random access are specified, if the record is not found within the search limits or the relative track number is outside the limits of the file.

If ACCESS IS 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 procedures of Data Management are executed.
2. A volume switch occurs.
3. The volume header label-checking procedures of Data Management are executed.
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, and then the statements following AT END, are executed.

The following are examples of READ statements:

```
READ INVENTORY AT END GO TO FINISH.

READ PAYROLL-FILE INTO AREA-1 AT END GO TO
  CALC-2.

READ PERSONNEL-FILE INVALID KEY GO TO 3.
```

WRITE Statement

The function of Option 1 of the WRITE statement is to release a logical record for a file specified as OUTPUT or I-O in an OPEN statement and to allow performance of specified imperative statements if, for random access files, the contents of the associated actual key and/or symbolic key are found to be invalid.

---

[1]Implemented for COBOL F only

Option 1 of the WRITE statement has the following format:

Option 1

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

Option 2 of the WRITE statement is used for output destined to be printed or punched if the user wishes to control line spacing or pocket selection.

Option 2 of the WRITE statement has the following format:

Option 2

```
WRITE record-name [FROM data-name-1]
    AFTER ADVANCING {data-name-2} LINES
                    {integer    }
```

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.

When the FROM option is used, data-name-1 must not be the name of an item in the file containing record-name. This form of the WRITE statement 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 rules specified for the MOVE statement without the CORRESPONDING option.

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:

1. The trailer-label writing procedure of Data Management is executed.
2. A volume switch occurs.
3. The header-label writing procedure of Data Management is executed.
4. The next logical record area in the output file is made available.

For COBOL F, the INVALID KEY option may be written for a file specified with either or both of the clauses ORGANIZATION IS INDEXED and ACCESS IS RANDOM when no error-processing declarative has been written for that file. For COBOL E, the INVALID KEY clause is required for files specified with either or both of the clauses ORGANIZATION IS INDEXED and ACCESS IS RANDOM. The error-processing declarative may also be specified. The statements following INVALID KEY are executed when the contents of actual key and/or symbolic key are invalid. The INVALID KEY sequence could be executed when:

1. The file has been opened as OUTPUT, organization is indexed, and access is either sequential or not specified. In this case the INVALID KEY sequence would be executed when either (a) the contents of the record key field are not in ascending order when compared with the contents of the record key field of the preceding record, or (b) the contents of the record key field duplicate that of the preceding record.
2. The file has been opened as I-O, organization is indexed, access is random, and a record is being added to the file. In this case the INVALID KEY sequence would be executed when the contents of the SYMBOLIC KEY field associated with the record to be added duplicate contents of a record key field already in the file.
3. The file has been opened as I-O, organization is direct, access is random, and a record is being added to the file. In this case the INVALID KEY sequence would be executed when either (a) the relative track specified in the ACTUAL KEY field is outside the limits of the file, or (b) for files with format F records, the figurative constant HIGH-VALUE or its equivalent has been moved into the first character position of the contents of the SYMBOLIC KEY field.

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

When Option 2 of the WRITE statement is used, the first character in each logical record for the file must be reserved by the user for the control character. The compiler will generate instructions to insert the appropriate carriage control character as the first character in the record. If the records are to be punched, the first character is used for pocket selection. It is the user's responsibility to see that the appropriate characters are punched on the carriage control tape.

If Option 2 of the WRITE statement is written for a record in a file, every WRITE statement for records in the same file must also contain this option.

When Option 2 of the WRITE statement is used, integer must be unsigned, and must have the value 0, 1, 2, or 3. The value 0 designates a carriage-control "eject" (i.e., skip to channel 1 of the next page). The value 1 designates single spacing; the value 2, double spacing; and the value 3, triple spacing.

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

Value	Interpretation
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.

The following are examples of WRITE statements:

```
WRITE SALARY-RECORD FROM OLD-RECORD AFIER
  ADVANCING 0 LINES.

WRITE NEW-WORK-CARD FROM WORK-CARD INVALID
  KEY GO TO END.
```

[EXT] REWRITE Statement

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

The format of the REWRITE statement is:

```
REWRITE record-name [FROM data-name]

   [INVALID KEY imperative-statement ...]
```

A REWRITE statement may be issued only after a READ statement for the file has been executed. A REWRITE statement can be written only for files opened as I-O. In COBOL E, but not in COBOL F, if such a file has indexed organization and random access (BISAM), a REWRITE statement must be given after every valid READ and previous to any other I-O statement for the file. (A REWRITE statement should not be executed if an invalid READ occurs.) The user cannot change the size of a record between the time it is read and the time it is rewritten.

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 the rules specified for the MOVE statement without the CORRESPONDING option.

For COBOL F, the INVALID KEY option may be given for files specified as ACCESS IS RANDOM when no error-processing declarative sections have been specified for the files. For COBOL E, the INVALID KEY clause is required for files specified as ACCESS IS RANDOM. The error-processing declarative may also be specified. The statements following INVALID KEY are executed when the contents of the actual key and/or symbolic key are invalid.

The keys are considered invalid under the following conditions:

1. When organization is relative and access is random, if the relative record number given in the symbolic key field is outside the limits of the file.

2. When organization is direct, access is random, and the record is not found within the search limits of the file; or the relative track number in the actual key field is outside the limits of the file.

Note: When a file's organization is indexed and access is random, the INVALID KEY clause may be used for language consistency, thus allowing the programmer to change access methods without rewriting the Procedure Division statements. When the clause is specified for such a file, the statements following INVALID KEY are never executed.

If ACCESS IS 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.

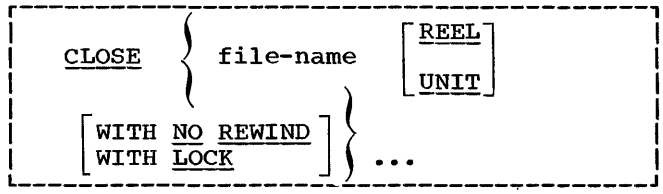
Since a REWRITE statement must always apply to the last record read, the values associated with the symbolic and/or actual key must be the same for the REWRITE statement as they were for the READ statement.

The following is an example of the REWRITE statement:

```
REWRITE SALARY-RECORD FROM HEAD-1 INVALID
  KEY GO TO RESTART.
```

CLOSE Statement

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



A CLOSE statement may be executed only for files that have been opened. A CLOSE statement for a file that was opened with the I-O option performs the same label-checking functions (on trailer labels) as a CLOSE statement for a file that was opened with the INPUT option.

The effects of a CLOSE statement depend upon whether or not the REEL or UNIT option is specified. A CLOSE statement without the REEL or UNIT option causes Data Management closing procedures to be executed; a CLOSE statement with the REEL or UNIT option causes Data Management volume-switching procedures to be executed.

The words REEL and UNIT are interchangeable; they are treated as the same COBOL reserved word.

A CLOSE statement without the UNIT or REEL option must be specified to terminate processing of the file. Because a CLOSE statement without the UNIT or REEL option causes Data Management closing procedures to be executed for the current volume (that is, the volume just processed) of the file, once such a statement is executed, an OPEN statement must be executed before any other reference can be made to that file.

Because a CLOSE statement written with the UNIT or REEL option causes volume-switching procedures to be instituted, such a statement should be written only for multi-volume files assigned to specific devices on which removable volumes (e.g., tape reels) may be mounted.

When a CLOSE statement with the UNIT or REEL option is executed for an OUTPUT file for which ORGANIZATION IS RELATIVE is specified or assumed, and ACCESS IS SEQUENTIAL is specified or assumed, Data Management volume-switching procedures will be initiated.

When a CLOSE statement with UNIT or REEL option is executed for an OUTPUT file for which ORGANIZATION IS DIRECT is specified, and ACCESS IS SEQUENTIAL is specified or assumed, the following actions will be taken: the current volume will be completed with dummy or capacity records up to the end of the current extent (whose tracks are now considered to be part of the file), Data Management volume-switching procedures will be initiated, and the contents of the ACTUAL KEY will be updated to reflect the relative track number of the last track of the old volume.

The creation of multi-volume files is highly dependent upon the SPACE parameter of the associated DD statement and the number of records written by the programmer. For further information, see the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form GC28-6380 under "Processing with BSAM" in the chapter "Execution Time Data Set Requirements."

Note: A CLOSE statement with the UNIT or REEL option should not be written for a file for which either ACCESS IS RANDOM or ORGANIZATION IS INDEXED (or both) has been specified.

|F ONLY| The WITH NO REWIND or WITH LOCK option can be specified in a CLOSE statement in conjunction with the UNIT or REEL option only in COBOL F. If more than one such statement is written for a file, the WITH NO REWIND or WITH LOCK option must be repeated in each statement. In other words, all CLOSE statements that institute volume switching for a file must specify identical options.

|F ONLY| If a CLOSE statement with the UNIT or REEL option is written for a file anywhere in the program, it specifies the positioning of the current volume at volume switching time whether or not the volume switching was initiated by the execution of this statement.

VOLUME POSITIONING: The effects of the CLOSE statement on the positioning of the volumes depend on whether or not the WITH NO REWIND or the WITH LOCK option, or neither of these, is specified. When a CLOSE statement without the UNIT or REEL option is specified, the only volume affected by the statement is the volume for

which the Data Management closing procedures are executed. If a CLOSE statement with the UNIT or REEL option is specified, the volumes affected are all volumes for which Data Management volume switching procedures are executed. Volume switching occurs when another volume is available and either an end-of-volume condition is detected during the execution of a READ or WRITE statement (automatic end-of-volume) or a CLOSE statement with the UNIT or REEL option is executed (forced end-of-volume). The options specified in the CLOSE statement for a file assigned to a direct-access device do not affect the positioning of the file.

If the CLOSE statement is specified without any option, there is an implied rewind of the current volume, and repositioning to the beginning of the file on the current volume takes place. However, if the REVERSED option of the OPEN statement associated with the file has been specified, repositioning to the end (i.e., the logical beginning) of the file on the current volume takes place.

If the CLOSE statement is specified with only the WITH NO REWIND option, positioning to the end of the file on the current volume takes place. If the REVERSED option of the OPEN statement has been specified, repositioning to the beginning (i.e., the logical end) of the file on the current volume takes place.

If the CLOSE statement is specified with only the WITH LOCK option, the action taken (e.g., rewind, unload, etc.) is a function of the DISP parameter of the associated DD statement. The action is the same whether or not the REVERSED option of the OPEN statement has been specified.

If a CLOSE statement is specified with only the REEL or UNIT option, at end-of-volume (automatic or forced) the current volume is rewound. However, if the REVERSED option of the OPEN statement has been specified, positioning to the end of the file on the volume takes place.

┌-------┐
|F ONLY|
└-------┘ If a CLOSE statement is specified with the REEL or UNIT option plus the WITH NO REWIND option, at the end-of-volume the current volume is positioned to the end of the file. If the REVERSED option of the OPEN statement has been specified, the volume is rewound.

┌-------┐
|F ONLY|
└-------┘ If a CLOSE statement is specified with the REEL or UNIT option plus the WITH LOCK option, the action taken at automatic end-of-volume is a function of the DISP parameter. At forced end-of-volume, positioning to the end of the file takes place unless the file was opened REVERSED, in which case the volume is rewound.

The examples in Table 10 show the effects of different options of the CLOSE statement on the volume positioning for intermediate reels (i.e., any but the last reel in a multi-volume file). The words REEL and UNIT are interchangeable as noted previously; therefore, the CLOSE statements could be coded CLOSE UNIT... and the same volume positioning would occur.

Note: The number of devices assigned to a multi-volume file may be less than the number of volumes in the file. In this case, units will be reused in serial order starting with the first. When specifying positioning options (e.g., CLOSE REEL WITH NO REWIND), the user should note that efficiency may be reduced due to the fact that positioning will take place before the volume is rewound and unloaded.

Table 10. Volume Positioning for Intermediate Reels

CLOSE Option	Volume Positioning for Intermediate Reels
CLOSE REEL WITH LOCK	Option determined from DISP parameter of DD card
CLOSE REEL WITH NO REWIND	Logical end of data on volume
CLOSE REEL (no option)	Logical beginning of data on volume
No CLOSE REEL specified (end-of-volume detected as a result of a READ or WRITE statement)	Rewind; new volume determined from DISP parameter of DD card

EXAMPLES:

If the programmer decides, at a point in the program, that no more information is needed from the first volume of a three-tape file, he may force end-of-volume switching procedures for the current volume by specifying

```
CLOSE INPUT-FILE-A REEL.
```

Repositioning to the beginning of the file on the current volume will take place. (A rewind of the volume is implied.) If the second volume is to be read in its entirety, no CLOSE statement need be given for the volume. However, if the programmer wants to force volume switching for the second volume, he must specify CLOSE INPUT-FILE-A REEL once again. In either case, repositioning to the beginning of the file on the current (i.e., second) volume will take place.

As stated previously, REEL or UNIT must be specified to force volume-switching procedures, and a CLOSE statement without the REEL or UNIT option must be specified to terminate the processing of INPUT-FILE-A. Therefore, the CLOSE statement at the logical end of the program might be written

```
CLOSE INPUT-FILE-A WITH LOCK.
```

Positioning of the current volume is then a function of the DISP parameter.

| F ONLY |

Another way of coding the above situation would be to specify the following CLOSE statement for the current volume:

```
CLOSE INPUT-FILE-A REEL WITH NO REWIND.
```

In this case, repositioning to the end of the file on the current volume will take place unless the device is needed for mounting the next volume. (Note that INPUT-FILE-A will not be rewound.) For the second volume, to force end-of-volume, the programmer would have to code an identical CLOSE statement, since all CLOSE statements that institute volume-switching for a file must specify identical options. Positioning for the second volume will be the same as for the first, whether or not the CLOSE statement is executed for the second volume.

DISPLAY Statement

The function of the DISPLAY statement is to write data on a low-volume output device. The format of the DISPLAY statement is:

```
DISPLAY { data-name }...
        { literal   }

[ UPON { CONSOLE       } ]
       { SYSPUNCH      }
       { mnemonic-name1}
```

When the UPON option is omitted, the system logical output device (SYSOUT) is assumed. When UPON CONSOLE is written, the system console device is specified. When UPON SYSPUNCH is written, the system logical punch device is specified.

If the input/output device specified by a DISPLAY statement is the same one designated by a WRITE statement, the output resulting from the statements may not be in the order in which the statements were encountered. For example, suppose the system logical output device was designated and the statements

```
WRITE WEEKS-PAY.
DISPLAY 'ABC'.
```

(where the contents of WEEKS-PAY is 123.00) were encountered. The output on SYSOUT might be

```
ABC
123.00
```

The operands of the DISPLAY statement, after any required conversion to external form, appear from left to right with no spaces between them. Any spaces desired between multiple operands must be explicitly specified.

For the system console and punch devices, a maximum logical record size is assumed. The maximum logical record size assumed for the system console device is either 72 characters (COBOL E) or 100 characters (COBOL F). The maximum logical record size assumed for the system logical punch device is 80 characters. (Positions 73 through 80 of the record are reserved, however, for the PROGRAM-ID name.) In COBOL E, the logical record size must be specified, for the system logical output device, on the associated SYSOUT DD statement. In COBOL F, a maximum logical record size of 120 characters is assumed. This assumed size is overridden if a logical record size is specified on the associated SYSOUT DD statement. See the publications IBM System/360 Operating System, COBOL (E) Programmers Guide, Form C24-5029 and IBM System/360 Operating System: (COBOL (F) Programmers Guide, Form C28-6380 for

---

[1]Implemented for COBOL F only

further information concerning the SYSOUT DD statement.

In COBOL E the number of operands in the DISPLAY statement may not exceed 19 on any device, even if the sum of the sizes of the operands does not exceed the specified maximum.

For both COBOL E and COBOL F, if the total character count of all the operands is less than the maximum logical record size or for SYSPUNCH, 72 characters, the remaining positions are padded with blanks. However, if the count exceeds the maximum logical record size or for SYSPUNCH, 72 characters, the processing differs between COBOL E and COBOL F.

Moreover, DISPLAY causes the printer to space prior to printing. WRITE AFTER ADVANCING also causes the printer to space before printing. However, a WRITE without the AFTER ADVANCING option does not cause the printer to space before printing. Possibly, mixed DISPLAYs and WRITEs without the AFTER ADVANCING option may cause overprinting.

In COBOL E truncation occurs at the end of the record if the DISPCK option has been used. Otherwise the result is unpredictable. In COBOL F operands are continued in the next record. As many records as necessary are written to display all of the operands specified. Those operands pending at the time of the break which are not numeric are split between lines if necessary. Numeric operands are not split between lines. Rather, the line is padded out with blanks and the numeric item starts the next line.

F ONLY

Mnemonic-name must be specified in the Special-Names paragraph of the Environment Division. Mnemonic-name may only be associated with the reserved words CONSOLE, SYSPUNCH, and SYSOUT.

When data-name specifies an item whose description is USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-3, it is 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.

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

Note: Group items, however, are treated as alphanumeric items; no conversion of elementary items within the group takes place.

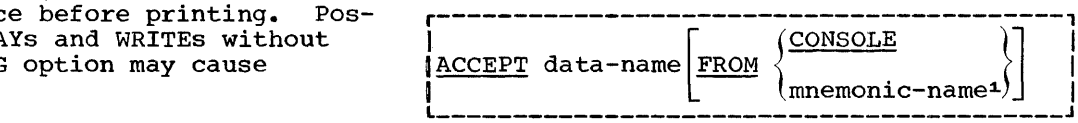
Literal may be any type of literal or figurative constant, with the exception of the figurative constant ALL.

Note: It is the programmer's responsibility to ensure that any characters within the data-name or literal being displayed are acceptable to the character set of the printer or device specified; otherwise, the data displayed is unpredictable.

ACCEPT Statement

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

The format of the ACCEPT statement is:



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 is moved into the area reserved for data-name. No editing or error-checking of the incoming data is performed.

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 must not exceed 72 character positions in length for COBOL E, and 255 character positions in length for COBOL F.

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

1. A system-generated message code is automatically displayed followed by the literal 'AWAITING REPLY'.

2. Execution is suspended. When a console input message, preceded by the same message code as in point 1 above, is identified by the control program, execution of the ACCEPT statement is resumed and the message is moved to the specified data-name.

---

[1]Implemented for COBOL F only

The message code serves as a key by which the control program correlates console input with the proper program.

F ONLY

Mnemonic-name must be specified in the Special-Names paragraph of the Environment Division. Mnemonic-name may only be associated with the reserved words CONSOLE and SYSIN. If it is associated with CONSOLE, data-name must not exceed 255 character positions in length. When the FROM CONSOLE option is not written and mnemonic-name is not associated with CONSOLE, one logical record is read from the system logical input device (SYSIN).

The following are examples of ACCEPT statements:

```
ACCEPT CONTROL-CARD-AREA.

ACCEPT IN-REC FROM CONSOLE.
```

Table 11 states restrictions on use of input/output statements. Y means that the statement may appear.

DATA MANIPULATION STATEMENTS

MOVE Statement

The MOVE statement is used to transfer data from one area of main storage to another and to perform conversions and/or editing on the data that is moved. Option 1 of the MOVE statement has the following format:

Option 1

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

If this option (the simple move) is used, 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 without regard to the level structure of the group items involved and without editing.

Table 11. Restrictions on Use of Input/Output Statements

	Statement						
Appearing in	OPEN	CLOSE	READ WRITE REWRITE	DISPLAY UPON CONSOLE SYSOUT SYSPUNCH	EXHIBIT TRACE	ACCEPT FROM CONSOLE	ACCEPT (from SYSIN)
Report Writer	Y	Y	Y	Y	Y	Y	Y
Error Processing	Y(1)	Y	Y(1)	Y	Y	Y	Y
Main Body of Procedure Division	Y	Y	Y	Y	Y	Y	Y
Debug Packet	Y	Y	Y	Y	Y	Y	Y

(1) Only permitted for files other than the one for which entry into the declarative section was made.

The following considerations pertain to moving items:

1. Numeric (external decimal, internal decimal, binary, external floating-point, internal floating-point, 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 clauses of the source field and receiving field differ, 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.

3. The MOVE statement may be used with the Sterling Currency feature. See the section entitled "Sterling Currency Feature."

Table 12 contains several examples illustrating MOVE. 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.

```
┌──────┐
│F ONLY│
└──────┘ Option 2
```

When Option 2 of the MOVE statement is used, selected items within data-name-1 ar moved, along with any required editing, to selected areas within data-name-2. Items are selected by matching the data-names of areas defined within data-name-1 with like data-names of areas defined within data-name-2. Option 2 of the MOVE statement ha the following format:

```
┌──────────────────────────────────────
│MOVE CORRESPONDING data-name-1
│
│   TO data-name-2...
└──────────────────────────────────────
```

The rules stated for the simple MOVE apply to each pair of corresponding items in the MOVE CORRESPONDING statement; thus, the effect of a MOVE CORRESPONDING statement is equivalent to a series of simple MOVE statements.

The following rules apply to the CORRESPONDING option:

1. At least one of the items of a pair o matching items must be an elementary item.

2. Items are corresponding data items if the respective data-names are the same, including all qualification up to but not including data-name-1 and data-name-2.

Table 12. Examples of Data Movement

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

3. Data-name-1, data-name-2, etc., must be group items.

4. Of the items subordinate to data-name-1 or data-name-2 the following are not considered CORRESPONDING items:

   a. An item named by the key word FILLER and any items subordinate to it.

   b. An item described by a REDEFINES or OCCURS clause, and any items subordinate to it.

   However, the items designated by data-name-1 and data-name-2 may be described with REDEFINES or OCCURS clauses or be subordinate to items described with REDEFINES or OCCURS clauses.

5. If either data-name-1 or data-name-2 is described with an OCCURS clause, it must be subscripted; each data item that corresponds will be subscripted by the compiler.

6. In determining which are corresponding data items, only the first complete description of any area will be considered in the case where a REDEFINES clause has been used. Consider the following data organization:

```
01  A
  02  B
  02  C REDEFINES B
    03  D
    03  E
  02  F
```

   Only B or F can be considered as potential corresponding items. This restriction does not preclude data-name-1 or data-name-2 from having REDEFINES clauses, or from being subordinate to data-names with REDEFINES clauses.

7. Neither data-name-1 nor data-name-2 can be data items with level numbers of 77 or 88.

8. Each corresponding source item is moved in conformity with the description of the receiving area.

To illustrate the use of MOVE CORRESPONDING, suppose that the programmer wishes to transfer corresponding items from a work area named INVENTORY-POSTING to an output area designated INVENTORY-RECORD. He could write this statement:

```
MOVE CORRESPONDING INVENTORY-POSTING TO
  INVENTORY-RECORD
```

Figure 11 shows the movement of data that might result from this statement. Note that non-corresponding items in the source area are not moved and that non-corresponding items in the receiving area are not affected.

Table 13 represents all permissible moves using the MOVE statement. The letter Y indicates a valid move and the letter N indicates an invalid move. A detailed description of the types of fields represented may be found in the chapter entitled "Data Division".

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} 'character-1'
                   {UNTIL FIRST}
  [REPLACING BY 'character-2']
```

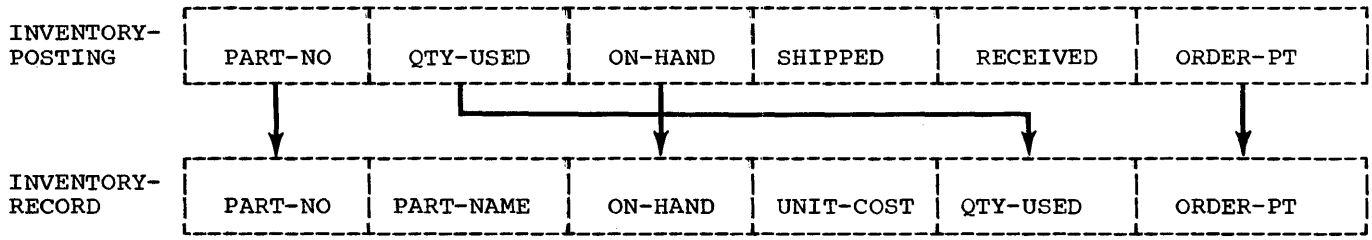


Figure 11. Data Movement Effected by MOVE CORRESPONDING Statement

• Table 13. Permissible Moves

Source Field	Receiving Field: GR	AL	AN	ED	ID	BI	EF	IF	RP	SR	SN
Group (GR)	Y	Y	Y	N	N	N	N	N	N	N	N
Alphabetic (AL)	Y	Y	Y	N	N	N	N	N	N	N	N
Alphanumeric (AN)	Y	Y	Y	N	N	N	N	N	N	N	N
External Decimal (ED)	Y	N	Y(1)	Y	Y	Y	Y	Y	Y	Y	Y
Sterling Nonreport (SN)	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
Internal Decimal (ID)	Y	N	Y(1,2)	Y	Y	Y	Y	Y	Y	Y	Y
Binary (BI)	Y	N	Y(1,2)	Y	Y	Y	Y	Y	Y	Y	Y
External Floating-Point (EF)	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y
Internal Floating-Point (IF)	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y
Report (RP)	Y	N	Y	N	N	N	N	N	N	N	N
Sterling Report(SR)	Y	N	Y	N	N	N	N	N	N	N	N
ZEROS	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPACES	Y	Y	Y	N	N	N	N	N	N	N	N
ALL 'character', HIGH-VALUES, LOW-VALUES, QUOTES	Y	N	Y	N	N	N	N	N	N	N	N

(1) For integers only
(2) COBOL E converts to External Decimal, COBOL F does not

Option 2

```
EXAMINE data-name  REPLACING

    { ALL       }
    { LEADING   }  'character-1'
    { UNTIL FIRST}
    { FIRST     }

    BY 'character-2'
```

Data-name in each option must refer to a data item containing the option 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 the Option 1 EXAMINE statement 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 right-hand 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 right-hand 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 Table 14.

[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 shown in Table 15.

Table 14. Examples of Data Examination

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

Table 15. Combinations of the FROM and TO Options of the TRANSFORM Statement

Operands	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 data-name-1 TO data-name-2	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. These items can be one or more characters, but must be equal in length.

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

1. 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 appears as an operand of the transformation rule, the user can in effect change the transformation rule during object time by changing data-name-1 or data-name-2.

Table 16 contains examples of data-name-3 results, using the figurative-constant-1 to figurative-constant-2, non-numeric-literal-2, and data-name-1 to data-name-2 combinations, respectively. The lower case letter b represents a blank.

Table 16. Examples of Data Transformation

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	IHGFEDCBA

ARITHMETIC STATEMENTS

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 that:

   a. If the data-name that follows GIVING is not used in the computation it may contain editing symbols.

   b. If the CORRESPONDING option is written, the operands must be group items.

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. A discussion of intermediate results is contained in Appendix D.

4. Decimal point alignment is supplied automatically throughout the computations.

5. In COBOL E, the figurative constant ZERO (ZEROS, ZEROES) may not be used in arithmetic statements.

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

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.

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 calculated for the result is greater than the number of places in the data item that 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 applies only to COMPUTATIONAL, COMPUTATIONAL-3, DISPLAY, or edited result fields. However, for COBOL F only, if within the arithmetic statement one or more

of the operands is in COMPUTATIONAL-1 or COMPUTATIONAL-2 format and this result field is in a format to which rounding applies, automatic rounding will take place whether ROUNDED is specified or not.

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

When this option is not specified, truncation occurs if, after decimal-point alignment, the number of places calculated for the result is greater than the number of places in the data item that is to be set equal to the calculated result.

Table 17 illustrates the relationship between a calculated result and the value stored in an item that is to receive it.

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.

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.

A discussion of intermediate results appears in the appendix entitled "Intermediate Results."

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 following are examples of the COMPUTE verb:

```
COMPUTE OVERTIME-PAY = REGULAR-PAY * 1.5.

COMPUTE TOTAL-WAGE = A.
```

Note that the second statement gives the same result as MOVE A TO TOTAL-WAGE.

Table 17. Rounding or Truncation of Calculations

	Item to Receive Calculated Result		
Calculated Result	PICTURE	Value After Rounding	Value After Truncating
-12.36	S99V9	-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

ADD Statement

The ADD statement adds two or more numeric values and substitutes the resulting sum for the current value of an item. Option 1 of the ADD statement has the following format:

Option 1

```
ADD {numeric-literal        } ...
    {floating-point-literal}
    {data-name-1           }

 {TO     }
 {GIVING}  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.

The following are examples of Option 1 of the ADD statements:

```
ADD INTEREST, DEPOSIT TO BALANCE.

ADD REGULAR-TIME OVERTIME GIVING
NEW-WEEKLY.
```

The first would result in the total sum of INTEREST, DEPOSIT, and BALANCE being placed at BALANCE, while the second would result in the sum of REGULAR-TIME and OVERTIME being placed at the location NEW-WEEKLY.

F ONLY

Option 2

Option 2 of the ADD statement has the following format:

```
ADD CORRESPONDING data-name-1

 TO data-name-2 [ROUNDED]

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

The CORRESPONDING option of the ADD statement allows the programmer to specify the addition of corresponding data items in one operation, similar to a MOVE statement with a CORRESPONDING option.

Numeric elementary items within data-name-1 are added to numeric elementary items with matching data-names within data-name-2. The rules stated for arithmetic statements apply to each pair of items in the ADD CORRESPONDING option.

Rules 2 through 7 listed for the MOVE statement with the CORRESPONDING option apply to Option 2 of the ADD statement.

When ON SIZE ERROR is used in conjunction with CORRESPONDING, the size error test is made only after the completion of all the ADD operations. If any of the additions produced a size error condition, the resultant field for that addition remains unchanged and the imperative statements specified in the SIZE ERROR clause is executed.

When the ROUNDED option is used in conjunction with CORRESPONDING, it applies to all of the add operations.

An example of Option 2 of the ADD statement is:

```
ADD CORRESPONDING WORK-RECORD TO UPDATE.
```

Consider the case where both group items are subdivided as follows:

```
01  WORK-RECORD DISPLAY.
      02  HOURS PICTURE 999V9.
      02  EMPLOYEE-NUMBER PICTURE 9(6).
      02  PAY-RATE PICTURE 9999V99.
      02  LOCATION PICTURE XX.
      02  PERIOD-OF-EMPLOYMENT
          PICTURE 999.

01  UPDATE DISPLAY.
      02  MAN-NO PICTURE 9(6).
      02  HOURS PICTURE 999V9.
      02  DIVISION PICTURE XX.
      02  PAY-RATE PICTURE 9999V99.
      02  PERIOD-OF-EMPLOYMENT
          PICTURE 999.
```

Elementary items with the same name (HOURS, PAY-RATE, and PERIOD-OF-EMPLOYMENT) are added and the result is placed at the corresponding location within UPDATE.

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.

Option 1 of the SUBTRACT statement has the following format:

Option 1

```
SUBTRACT {data-name-1           } ...
         {numeric-literal-1     }
         {floating-point-literal-1}

     FROM {data-name-m [GIVING data-name-n]}
          {numeric-literal-m GIVING data-  }
          {  name-n                        }
          {floating-point-literal-m        }
          {  GIVING data-name-n            }

     [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.

F ONLY

Option 2

Option 2 of the SUBTRACT statement has the following format:

```
SUBTRACT CORRESPONDING data-name-1

     FROM data-name-2 [ROUNDED]

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

The CORRESPONDING option of the SUBTRACT statement is analogous to the CORRESPONDING option of the ADD statement.

MULTIPLY Statement

The MULTIPLY statement multiplies two numeric data items and sets the value of a data item 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...]
```

When the GIVING option is omitted and the second operand is a data-name, the product replaces the value of the data-name. For example, the following would result in the product being placed at BALANCE.

```
MULTIPLY INTEREST-RATE BY BALANCE.
```

DIVIDE Statement

The DIVIDE statement divides one numeric data item by another and sets the value of an item 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...]
```

If the GIVING option is not used, the second operand must be a data-name.

Division by zero results in a SIZE ERROR condition.

When the GIVING option is omitted and the second operand is a data-name, division results in this data-name being set equal to the quotient. For example, the following would result in the quotient being placed at HOURS:

```
DIVIDE COUNT INTO HOURS.
```

PROCEDURE BRANCHING STATEMENTS

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

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 2

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

  DEPENDING ON data-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 an unconditional GO TO (Option 1) is used and a procedure-name is not specified, the GO TO statement must have a paragraph-name, be the only statement in the paragraph, and 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 will lead to erroneous results. Refer to the discussion of the ALTER statement for its effect on the unconditional GO TO. The maximum number of paragraph-names or procedure-names is 2031.

In Option 2 of the GO TO statement, 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.

In COBOL E, data-name may not be subscripted.

Consider the following example:

```
GO TO TOTAL DEVIATION ERR-ROUTINE DEPENDING
  ON ERRCODE.
MOVE RATE TO CALC-AREA.
```

If ERRCODE had a value at execution time of 2, a branch to procedure-name DEVIATION would take place. If the value of ERRCODE exceeded 3, no branch would be taken and control would pass to the MOVE 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 unconditional GO TO (Option 1) statement. The effect of an ALTER statement is to replace the procedure-name specified in the Option 1 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.

The following are examples of the use of the ALTER statement:

Example 1

```
            ALTER STEP-1 TO PROCEED TO
            PROCESS-2.
              .
              .
  STEP-1.   GO TO PROCESS-1.
              .
              .
```

Example 2

```
            ALTER STEP-1 TO PROCEED TO
            PROCESS-2.
              .
              .
  STEP-1.   GO TO.
              .
              .
```

In both cases, when STEP-1 is executed, an unconditional branch is taken to PROCESS-2.

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 four formats, each discussed below:

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   } TIMES
                         {data-name}
```

Option 2 is the TIMES option, of the PERFORM statement. 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 value, less than 32,768. If the value of the data-name is negative or zero, 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 of the PERFORM statement. 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.

The following is an example of an Option 3 PERFORM statement:

```
PERFORM ROUTINE-1 UNTIL ITEM-1 IS LESS
THAN ITEM-2.
```

Option 4

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

Option 4 is the VARYING option, of the PERFORM statement. Test-condition may be simple or compound.

The VARYING option may be used to increment or decrement the value of one or more data-names depending on whether the BY value is positive or negative.

When one data-name (data-name-1) 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 BY value 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 in Option 4 of the PERFORM statement must represent numeric values but need not be integers; they may be positive, negative, or zero.

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

Data-name-1, data-name-4 and data-name-7 must not be alternate names for the same data item. For all options, the first statement of procedure-name-1 is the point to which sequence control is transferred by 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. The time sequence of execution of exits established by PERFORM statements must be in the inverse order in which they were established. It is permissible for two or more PERFORM statements to have a common exit.

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 can be referred to by more than one PERFORM statement and 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 one time 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 12, 13 and 14 illustrate the logical flow of Option 4 of the PERFORM statements, varying one, two, and three data-names, respectively.

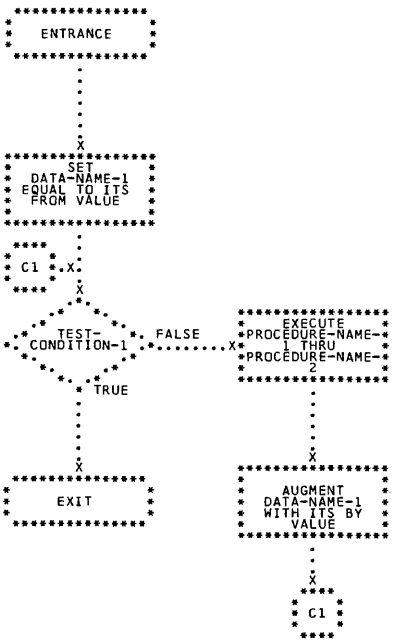


Figure 12. Logical Flow of Option 4 PERFORM statement Varying One Data-Name

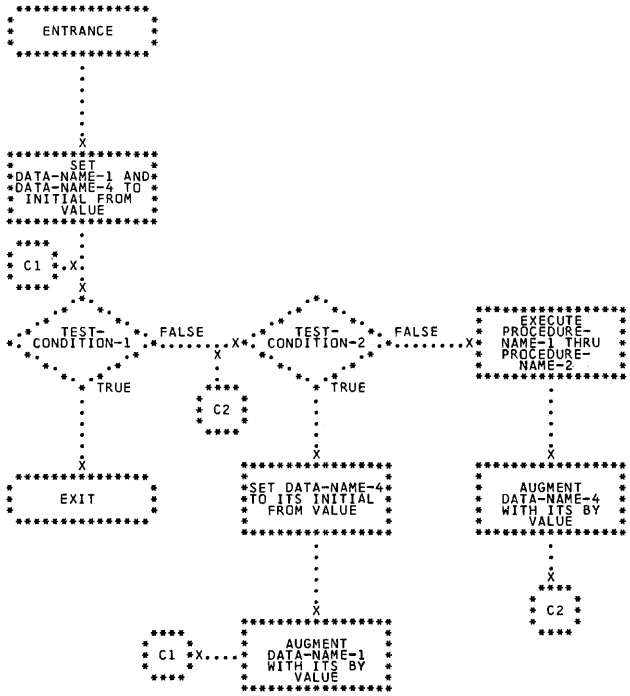


Figure 13. Logical Flow of Option 4 PERFORM Statement Varying Two Data-Names

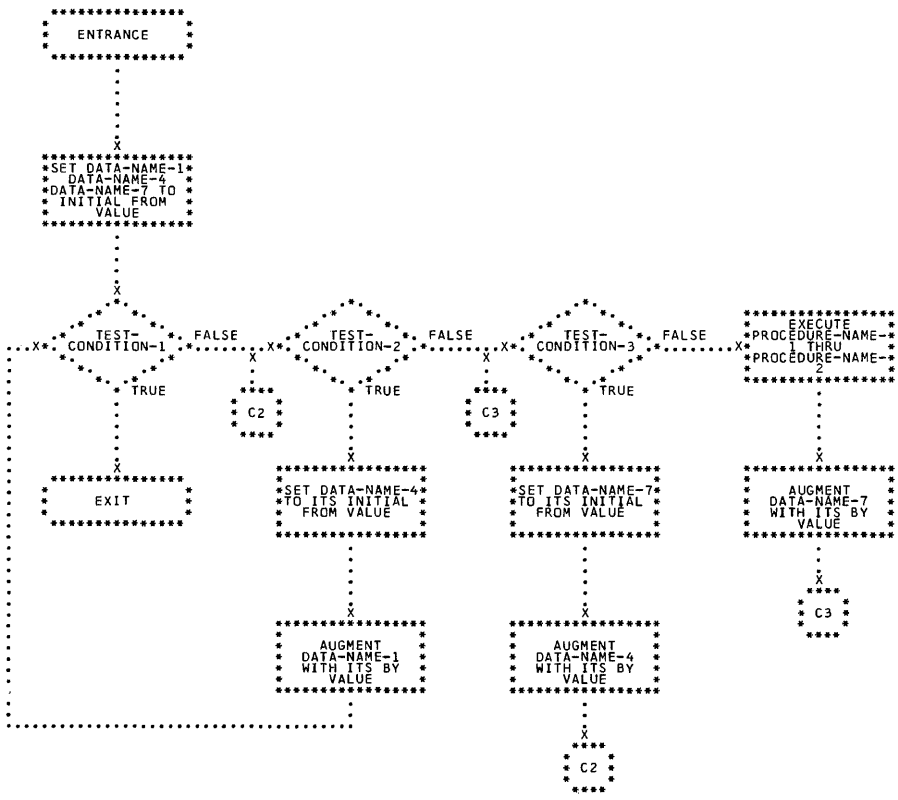


Figure 14. Logical Flow of Option 4 PERFORM Statement Varying Two Data-Names

Table 18. Restrictions for Procedure-Branching Statements

Appearing in:	Statement				
	ALTER	GO TO	PERFORM	STOP RUN	STOP LITERAL
Report Declarative	Y	Y(1)	Y	Y	Y
Error Declarative	Y	Y	Y	Y	Y
Debug Packet	Y	Y	Y	Y	Y
Main Body of the Procedure Division	Y	Y	Y	Y	Y
SORT Input or Output Procedure	Y(1)	Y(1)	Y(1)	N	Y

(1) Operands of these statements should be procedure-names appearing in the same section.

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. It returns control to the operating system or, if the program containing the STOP RUN has been invoked by another program, to the invoking program.

The STOP literal statement causes a system-generated message code and the specified literal to be displayed on the console and the object program to pause. The program may be resumed only by operator intervention. A reply including the system-generated code must be keyed in on the console in order to resume execution. In COBOL E, the size of the literal is restricted to 72 characters. See the publication IBM System/360 Operating System: Messages and Codes, Form GC28-6631.

Table 18 states restrictions on the appearance of procedure-branching statements. Y means that the statement may appear; N indicates that it must not; the text indicates the outcome if the statement does appear.

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 one or more COBOL subprograms or other language subprograms. See the following publications for more information on calling and called programs:

IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form GC28-6380 and IBM System/360 Operating System: COBOL (E) Programmer's Guide, Form GC24-5029.

The ENTER statement has the following two formats:

Format 1 (Used in calling program)

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

Format 2 (Used in a COBOL subprogram)

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

subprogram statements
    .
    .
    .
ENTER LINKAGE.
RETURN
ENTER COBOL.
```

Entry-name is an external-name; therefore, it consists of single quotation marks enclosing no more than eight alphabetic and numeric characters, the first of which must be alphabetic. It must not be the same as the program-name specified in the PROGRAM-ID clause.

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

In the USING clause of Format 1, 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

Format 2 of the ENTER statement 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 linkage parameters are saved for eventual return and address parameters are obtained. In this form, each data-name in the USING portion of the ENTRY statement must be defined in the Linkage Section of the Data Division of the called program, must have a level-number of 01 or 77, and must not be subscripted.

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.

Data-names or other arguments specified in a CALL statement may be qualified or subscripted. When group items with level-numbers other than 01 are specified, proper word-boundary alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2. See the appendix entitled "Slack Bytes" for information on adjustment of word-boundary alignment.

The RETURN statement enables restoration of the necessary registers saved at an entry point. The return from a subprogram is always to the first instruction after 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, unless it is the first statement to be executed.

Refer to the appendix entitled "Examples of COBOL Programs" for sample coding illustrating linkage between a calling and a called program.

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.

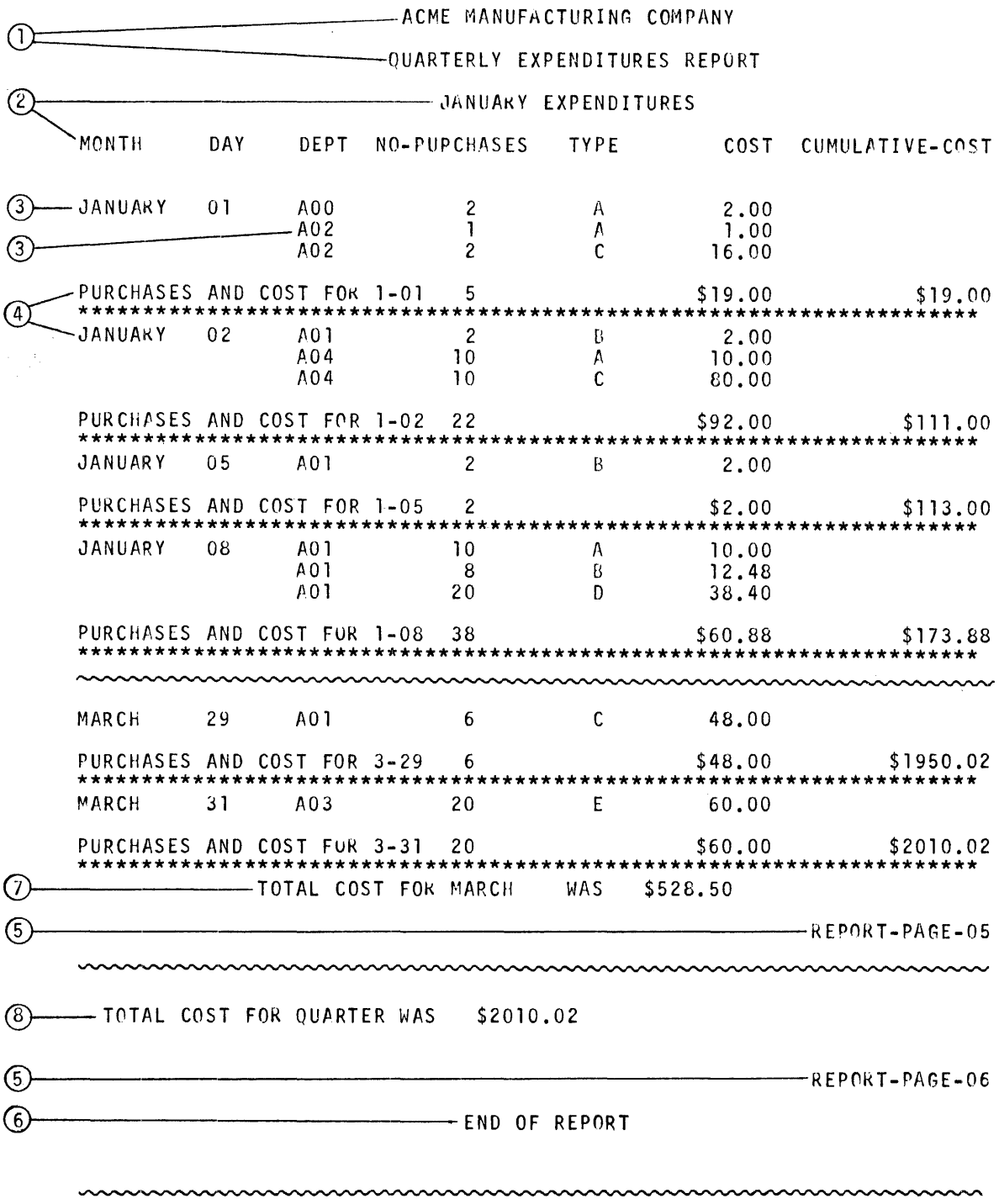
If control reaches an EXIT paragraph and no associated PERFORM or USE statement is active, control passes through the exit point to the first sentence of the next paragraph.

NOTE Statement

The NOTE statement permits the programmer to write explanatory comments in the Procedure Division of a source program that 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 EBCDIC set may appear. If NOTE is not the first word in a paragraph, the comments end with a period followed by a space. However, if NOTE is the first word of a paragraph, any subsequent sentences within the paragraph are also considered notes. Proper format rules for paragraph structure must be observed. The paragraph-name should begin in margin A and one or more sentences should appear in margin B. NOTE itself should not start in margin A.



• Figure 15. Sample Portions of a Report Produced by Report Writer Feature

INTRODUCTION

The COBOL F Report Writer Feature enables the programmer to generate written reports with a minimum of Procedure Division coding, thus simplifying the task of writing programs to produce such reports.

Figure 15, on the opposite page, shows several parts of a report produced by the Report Writer feature. (The complete report, as well as the coding used to produce it, and detailed explanatory remarks, are presented in Figures 16 and 17, at the end of this chapter.)

A report by the Report Writer feature may be thought of as consisting of seven distinct types of information, known as report groups. Six of these are illustrated by the circled numbers in Figure 15. The programmer specifies the content of each report group by means of special statements in the Report Section of the Data Division. These statements provide not only the content of the various report groups, but also their physical format on the page and instructions as to when each group is to be written. In the Procedure Division the programmer must include only three types of statements to cause the report to be written: an INITIATE statement, which is executed just before the report is to be started; any number of GENERATE statements, which cause individual lines of the report to be written; and a TERMINATE statement, executed when the report is completed.

The circled number 1 in Figure 15 points to a Report Heading report group, which is written once at the beginning of the report. Similarly, Number 6 is a Report Footing report group, written once at the end of the report. Number 2 is a Page Heading, written at the top of each page, and number 5 a Page Footing, written at the bottom. The Report Writer automatically handles the end-of-page condition (at which time the Page Footing for the old page and the Page Heading for the new one are written) according to information supplied by the programmer in the Report Section. The page number need not be maintained by the programmer; it is written from a counter that is kept by the Report Writer.

The basic parts of the Report Writer feature are illustrated by the circled numbers 3 and 4. Number 3 points to two items of the report called Detail lines. Detail lines, which may occupy one or more physical lines, consist of the actual data of the report gathered from one or more input files, or from information otherwise made available to the Report Writer. The Detail lines in Figure 15 were from punched card input, but were reformatted before being written. (The data used in this example, with explanatory text, is illustrated in Figure 16, at the end of this chapter.) The Detail line is the only one of the seven types of report information that the programmer specifically requests to be written, by the use of a GENERATE statement in the Procedure Division.

The Detail line, like the other report groups, is formatted once in the Report Section of the Data Division.

In this illustration number 3 is pointing to two separate Detail lines, even though they were both written (at different times) by the same GENERATE statement.

Number 4 points to two lines that make up a report group called a Control Footing. (As with the Detail line and all other report groups, the Control Footing may consist of any number of lines.) The Control Footing is essentially a summation of the data that has been presented in one or more previous Detail lines, or previous Control Footings. The programmer does not specifically call for the Control Footing to be written, but indicates through statements in the Report Section the circumstances under which the Report Writer is to automatically write the group.

The writing of Control Footings is governed by the values of data items known as control fields. A control field is a data item which may be expected to change. In Figure 15, the control field governing the Control Footing pointed to by number 4 is the data item in the column labeled DAY. This Control Footing is written every time the value of the data item DAY changes, but before the Detail line with the new value is written. In addition to the text, "PURCHASES AND COST FOR (date)," and the line of asterisks, the Control Footing writes three totals: the number of purchases for the day, the cost of those purchases, and a running total of the cost for the month.

Number 7 points to a Control Footing governed by the control field MONTH. Thus when the month changes, the total cost for the preceding month is written. Similarly, number 8 points to a Control Footing which is written only when the report is ended by the execution of a TERMINATE statement. It writes the total expenditure for the quarter.

The programmer does not code routines to compute any of the above-mentioned totals. He merely indicates in the Report Section that the source of certain information in the various Control Footings is the sum of certain data items. The Report Writer generates the instructions which perform the necessary calculations.

The writing of the seventh type of report group, called a Control Heading, (not illustrated), is, like the Control Footing, governed by a control field. It is often written in conjuction with a Control Footing, and may be governed by the same control field. In such case the Control Heading would be written immediately following the Control Footing and would appear above subsequent Detail lines, including the originally-referenced Detail line that caused both the Control Heading and Control Footing to be written.

The remainder of this chapter is devoted to a detailed presentation and explanation of the Report Writer verbs, and the rules governing their use. The reader may find it helpful to refer to the coding example (Figure 16) and Report Writer output (Figure 17) when reading this material.

DATA DIVISION CONSIDERATIONS

The names of all the reports to be produced must be named in the File Section of the Data Division. A Report Section must be added at the end of the Data Division to define the format of each report.

FILE SECTION

Each FD entry in the File Section furnishes information concerning the physical structure, indentification and record-names pertaining to a given file.

General Format

```
FD file-name
|
|  LABEL RECORDS-CLAUSE
|  [BLOCK CONTAINS-clause]
|  [RECORD CONTAINS-clause]
|  [RECORDING MODE-clause]
|  [DATA RECORDS-clause]
|  [REPORT(S)-clause.]
```

A discussion of all the above-mentioned clauses appears in the "Data Division" chapter. In an FD entry with a REPORT clause, the RECORD CONTAINS clause has no effect and may be omitted. A description of the REPORT clause follows.

REPORT Clause

When the Report Writer is used, the FD entries of the files on which the reports are to be written must include a REPORT clause containing the names of each of the reports to be produced. The reports may be of different sizes, formats, etc.c and the order in which their names appear is not significant.

The format for the REPORT clause is:

```
{REPORT IS  }
{REPORTS ARE} report-name-1...
```

The REPORT clause cross references the Report Description entries with their associated File Description (FD) entry. Each unique report-name must appear in one and only one FD entry and be the subject of one and only one RD entry in the Report Section.

REPORT SECTION

The Report Section must be the last section in the Data Division. It consists of two types of entries for each report; one describes the physical aspects of the report format -- number of lines per page, location on the page of various groupings, etc. The other type describes the conceptual characteristics of the report -- descriptions of the group items which make up the document. There are seven classes of group items, which are called Report Group Description Entries:

1. Report Heading
2. Report Footing
3. Page Heading
4. Page Footing
5. Control Heading
6. Control Footing
7. Detail.

A discussion of each of these items follows. The REPORT Section must begin with the header REPORT SECTION.

General Format

```
REPORT SECTION.
RD Report-name
    [CODE-clause]
    [CONTROL-clause]
    [PAGE LIMIT-clause].
```

RD is the level indicator.

Report-name is the name of the report and must be unique. The report-name must be specified in a REPORT clause in the File Description entry for the file on which the report is to be written.

CODE Clause

The CODE clause is used to specify an identifying character that is appended to each line produced. The identification is meaningful when more than one report is written on a file. The format for the CODE clause is:

```
WITH CODE mnemonic-name
```

Mnemonic-name Must be associated with a single character in the Special-Names paragraph in the Environment Division. The identifying character is appended to the beginning of the line, preceding the carriage-control/line-spacing character. This clause should not be specified if the report is to be printed on-line.

Note: The first character of each print line is always reserved for a code even if one has not been specified. Thus the presence or absence of a code has no effect on the size of the logical record, which will always be 144 characters.

CONTROL Clause

The CONTROL clause indicates the data-names which specify the control hierarchy for the report.

A control is a data item in the printed report. All control items for the report are tested each time a Detail Report group is to be written. If the test indicates that the value of a control item has changed, a control break is said to occur and special action (described below) is taken before the Detail group is written. The format for the CONTROL clause is:

```
{CONTROL IS  }   {FINAL                    }
{            }   {data-name-1...           }
{CONTROLS ARE}   {FINAL data-name-1...     }
```

The data-names specify the control hierarchy of the report and are listed in order from major to minor. FINAL, when it is present, is the highest control item, data-name-1 is the major control, and subsequent data-names would be intermediate controls, down to the last, which would be the minor control.

Although FINAL is the highest level control item, its presence is never required in a CONTROL clause.

For further discussion of FINAL, see the sections pertaining to the CONTROL HEADING, CONTROL FOOTING and RESET clauses, below.

The data-names must be defined in the File or Working-Storage Sections of the Data Division. FINAL is the exception to the rule that all control items must be data-names.

In general, control items (other than FINAL) are ordered by the frequency that their values are expected to change (although such ordering is not mandatory). For example, if a report has three control items, Year, Month, Day, would most likely be the minor (lowest) control and Year the major (highest) control.

The action to be taken as a result of a control break depends on what the programmer defines in the Report Section. He may define a Control Heading Report group and/or a Control Footing, or neither, for each control item.

Control Headings and Footings represent action to be taken (data to be summed, lines to be written etc.) before or after Detail lines are written when a control break has occurred. Thus, if a report was being written listing daily expenditure for

each department of a company for a period of three months, the Control Heading for the data item (control) Day might be a header, below which each of the Detail lines would be written. The Control Footing for Day might be a line or lines summing the total expenditure:

```
          Dept.      Amt.         Total

March 26  A1       $23.92         $23.92
          A2        $6.07
                    $7.13         $13.20
          B16     $105.90        $105.90
**********************************
Expenditure for March 26:        $143.02
```

In the above illustration, the line of asterisks and the line directly below it which begins "Expenditure..." is the Control Footing for Day. It was written because the last input record to be processed contained a date other than March 26. After the two Control Footing lines, a new header (Dept....) will be written.

The above example was produced as a result of a series of GENERATE statements and two control breaks. The GENERATE statements produced the four Detail lines (between the heading and the line of asterisks). A control break for Day (when March 26 became March 27) resulted in the writing of the Control Footing (the asterisks and the line beginning "Expenditure"). The previous control break for Day (when March 25 became March 26) caused the heading to be written.

The order of writing of control groups is as follows: first the Control Footing of the lowest-level control, followed in ascending order by the Control Footings, if any, up to and including the level at which the break occurred. Then Control Headings, if any, will be written starting with the Heading for the control which caused the break (the Footing of which had just been written) and proceeding in descending order down to the Heading of the lowest control. Then the original Detail group is written.

For example, in the program with the controls Year, Month, and Day (each of which are assumed to have Control Headings and Footings), if a control break occurred for Year (i.e., if the input data for Year has changed), the first item to be written would be the Control Footing for Day, followed by the Control Footings for Month and Year, then the Control Headings for Year, Month and Day. Finally, the Detail line would be written. When a control break at an intermediate or major level occurs, breaks at all lower levels are implicity assumed.

If, in the course of writing Control Headings, Footings or Detail groups, a page overflow condition is detected, a Page Footing is written (if specified), the next page is skipped to and a Page Heading (if specified) is written on the new page

It should be noted that the only group that the programmer specifically requests to be written is the Detail group. Control Headings and Footings, Page Headings and Footings, and Report Headings and Footings are written automatically (if they are specified) when the Report Writer determines that they are required.

PAGE LIMIT Clause

The PAGE clause is used to describe the format of a page of the report. The format of the PAGE clause is:

```
PAGE [LIMIT IS  ] integer-p {LINE  }
     [LIMITS ARE]           {LINES }

   [HEADING integer-h]

   [FIRST DETAIL integer-d]

   [LAST DETAIL integer-e]

   [FOOTING integer-f]
```

where:

integer-h is the number of the first line on which anything may be written. No report group can start before line h. Integer-h in effect determines the size of the top margin of the page.

integer-d is the number of the first line on which a Detail or a Control Heading line may be written. No Detail or Control group can start before line d.

integer-e is the number of the last line on which a Detail or Control Heading line may be written. No Detail or Control Heading extends beyond line e.

integer-f is the number of the last line on which a Control Footing may be written. No Control Footing can start before line d or extend beyond line f. Page Footings follow line f, but do not extend beyond line p.

integer-p is the number of the last line on which anything may be written. Integer-p in effect determines the size of the bottom margin of the page.

e: LIMIT(S) is an optional word.

The PAGE clause entry values can be ned up as follows:

integer-h must be greater than or equal to 1.

integer-d must be greater than or equal to h.

integer-e must be greater than or equal to d.

integer-f must be greater than or equal to e.

integer-p must be greater than or equal to f.

ndividual parts of the PAGE clause may mitted. The following are assumed for omitted values:

integer-h = 1

integer-d = 1

integer-e = 48

integer-f = 48

integer-p = 52

rt Group Description Entry

. report may be devided into report ps. A report group is a set of data s that is to be presented as an indi- al unit, irrespective of its physical at. It may consist of several report s containing many data items, or of one rt line containing a single data item.

hree categories of report group defini- s are provided: Heading groups, Foot- groups, and Detail groups.

he data items comprising a report group be identified by the level number 01 a TYPE clause. All group description ies are located immediately following Report Description (RD) entry for the rt of which they are a part.

he description of the report group is ogous to that of a data record in that s formatted by level numbers. It con- s of a set of entries defining the acteristics of the elements. The ement of an item in relation to the re report group and to the overall for- as well as to any control factors ciated with the groups, is defined by report group description entry.

The 01 level description indicates the type of report group being described. The 02 level descriptions describe the information to be written when conditions indicate that a report group is to be written.

The formats of the 01 and 02 level descriptions of report groups are:

```
01 [data-name-1]  TYPE-clause
   [LINE-clause]  [NEXT GROUP-clause]

02 [data-name-2]  [LINE-clause]
   [COLUMN-clause][GROUP INDICATE]
   [BLANK-clause] [JUSTIFIED RIGHT]

   PICTURE-clause  { SOURCE-clause }
                   { SUM-clause    }
                   { VALUE-clause  }
```

Data-name-1 is the name of the report group being defined. It need be specified only if either the USE BEFORE REPORTING or the GENERATE statement (both described below) is used within the Procedure Division to refer to it. If data-name-1 is specified, it must be unique and may not be qualified.

Data-name-2 is the name of the elementary item being defined. It need be named only if the description contains a SUM clause which is referred to in the Procedure Division, or by another SUM clause. Data-name-2, if specified, must be unique within the report description. If data-name-2 is common to more than one RD entry, any reference to it outside the Report Section must be qualified.

The BLANK, JUSTIFIED RIGHT, PICTURE and VALUE clauses are described in the chapter "Data Division."

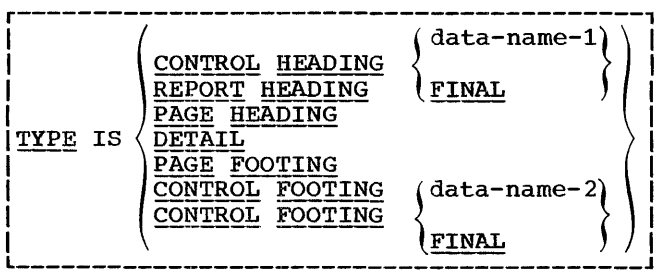
Report group names are required when reference is made in the Procedure Division to:

1. A DETAIL report group by a GENERATE statement.
2. A HEADING or FOOTING report group by a USE statement.

Note: level numbers greater than 02 should not be used.

TYPE Clause

The TYPE clause specifies the type of report group being described. The format of the TYPE clause is:



The following list of abbreviations may be used in the TYPE clause:

RH	REPORT HEADING
PH	PAGE HEADING
CH	CONTROL HEADING
DE	DETAIL
CF	CONTROL FOOTING
PF	PAGE FOOTING
RF	REPORT FOOTING

A REPORT HEADING report group is written only once -- at the beginning of the report. The programmer does not expressly specify that it be written; if a Report Heading has been specified in the Report Section, it will be written automatically when the first GENERATE statement is executed for the report. Only one Report Heading report group may be defined for a report.

A PAGE HEADING report group is written at the beginning of each page. Only one Page Heading report group may be defined for a report. Like the Report Heading, the Page Heading is written automatically; the programmer does not specifically call for it. (On the first page the Page Heading is written after the Report Heading, if a Report Heading has been specified.)

A CONTROL HEADING report group is written each time there is a control break in the control for which the Heading is specified. Only one Control Heading report group may be defined for each control specified in the CONTROL clause of the Record Description entry. However, a Control Heading need not be defined for each control. Control Headings are not written automatically after a page break. If Control Heading information is desired at the top of each page, it should be included as part of the Page Heading report group. When CONTROL HEADING FINAL is specified, the indicated source material will be written after the first GENERATE statement is executed. If the GENERATE statement refers to a report-name, the Control Heading will be written after the Report Heading and Page Heading, if any. If the GENERATE statement refers to a Detail group, the Control Heading will be written before the desired Detail line, but after a Report Heading or Page Heading, if specified.

Note that when CONTROL HEADING FINAL is not specified, a Control Heading will not be written above any Detail lines until a control break occurs for which a Control Heading has been specified.

A DETAIL report group is written each time a GENERATE statement that refers to the report group is executed. The GENERATE statement refers to the data-name specified in the level 01 description. There is virtually no limit to the number of Detail report groups that may be defined for a report.

A CONTROL FOOTING report group is written automatically each time there is a control break in the control specified. Only one Control Footing report group may be specified for each control listed in the CONTROL clause of the Report Description entry. However, a Control Footing need not be defined for each control. When CONTROL FOOTING FINAL is specified, it is written upon execution of the TERMINATE statement, after all other Control Footings, if any.

A PAGE FOOTING report group is written at the end of each page. Only one Page Footing report group may be defined for a report.

A REPORT FOOTING control group is written at the end of the report. It is printed only when the TERMINATE statement is executed.

LINE Clause

The LINE clause indicates the absolute or relative line number of the entry in reference to the page or previous entry. All data defined up to the next LINE clause (or up to the end of the report group description) is printed on the line specified. The LINE clause must be specified for each line printed. A LINE clause must be specified in either the 01 or first 02 level description of a report group.

If the LINE clause is specified at the report group (01) level, all entries for the first report line within the report group are presented on the specified line number, unless another LINE clause is present at the 02 level.

The format for the LINE clause is:

```
LINE IS { integer-1      }
        { PLUS integer-2 }
        { NEXT PAGE      }
```

Integer-1 is an absolute line number. It specifies the number of the line on which to write the first line of the group of data being defined.

Integer-2 is a relative line number. It specifies the number of lines to be skipped before writing the group of data being defined. (If the group being defined is a Page Heading, then integer is relative to line h, where h is defined in the PAGE clause.)

NEXT PAGE indicates that the group of data being defined is to appear on the next page. This option should be specified only for the 01 level entry of a report group. It causes the Page Footing and Page Heading, if specified, to be written.

If more than one LINE clause is specified within the definition of a report group, the following restrictions apply: (1) absolute line numbers must be specified in ascending order and (2) an absolute line number must not follow a relative line number.

Note: LINE NEXT PAGE should not be specified for Report Heading, Page Heading, Page Footing, or Report Footing report groups.

NEXT GROUP Clause

The NEXT GROUP clause specifies line control following the writing of the report group being defined. The format of the NEXT GROUP clause is:

```
NEXT GROUP { integer-1      }
           { PLUS integer-2 }
           { NEXT PAGE      }
```

Integer-1 is an absolute line number and indicates the number of the line to be skipped to.

Integer-2 is a relative line number and indicates the number of lines to be skipped. Integer must be greater than 0; a value of 0 will not cause overprinting.

NEXT PAGE indicates that the next page is to be skipped to. The Page Footing and Page Heading, if indicated, will be written. NEXT PAGE may be specified for an 01 level entry only.

Note: NEXT GROUP NEXT PAGE should not be specified for Report Heading, Page Heading, Page Footing, or Report Footing groups. If NEXT GROUP is specified for an intermediate control, it will be applied if a control break occurs for a higher level control.

COLUMN Clause

The COLUMN clause specifies the position in the print line of the first character of the data item being defined. The format of the COLUMN clause is:

```
COLUMN integer
```

Integer specifies the position in the print line where the leftmost character of the data item is to be placed.

If the COLUMN clause is not specified, the data item will not be written, although SOURCE material will be moved and data will be summed.

GROUP INDICATE Clause

The GROUP INDICATE clause specifies that the data item being defined is to be written only after a control break. This clause may only be specified for a data item within a Detail report group description. Use of the clause will result in the data item so specified being written only once between control breaks governing the Detail group being defined. The format of the GROUP INDICATE clause is:

```
[GROUP INDICATE]
```

GROUP INDICATE is not applied after a page break.

SOURCE Clause

The SOURCE clause specifies the source of the data to be moved to the data item being defined. (The moving is done in accordance with the rules of the MOVE verb.) The format of the SOURCE clause is:

```
[SOURCE data-name]
```

Data-name is the name of the data to be moved to the data item being defined and must itself be defined in the File, Working-Storage or Linkage Section. It may be qualified and/or subscripted.

SUM Clause

The SUM clause is used to cause automatic summation of data and may only appear in a CONTROL FOOTING report group description. The format of the SUM clause is:

```
SUM   data-name-1 ... [UPON data-name-2]
      [RESET ON { data-name-3 }]
                { FINAL       }
```

The data-names following SUM are the names of the items to be summed. They must be defined in the File, Working-Storage or Linkage Sections, or may be the names of other sum counters defined in the same report. Data-name-1 and data-name-3 may be qualified but not subscripted. The rules that apply to data-names used in arithmetic statements also apply to data-names used in the SUM clause of the Report Writer.

When the SUM clause is specified, it causes the definition of two data items: one is within the print line (this is the same type of field that is defined when the VALUE or SOURCE clauses are specified rather than the SUM clause); the other is a sum counter. The sum counter is used to contain the sum of the data-names specified in the SUM clause. When the report group containing the SUM clause is printed, the value from the sum counter is moved to the print line.

If sum counters are operands of SUM clauses in a higher position in the control hierarchy, the lower hierarchy sum counters are not reset until their values have been summed by the higher level SUM clause.

Summing sum counters at an equal position in the control hierarchy is known as cross footing and may occur in a Control Footing report group.

A SUM clause causes the automatic summation of data. The effect is as if the programmer had calculated the sum himself with one or more ADD statements and referred to it in the Control Footing report group by a SOURCE clause.

Each time a GENERATE statement that refers to a Detail report group in the report is executed, the values of data-name-1, etc., are added to the sum counter (except as noted below).

When the UPON option is used, the values in data-name-1, etc., are added to the sum counter only when a GENERATE statement that refers to data-name-2 is executed.

When the RESET option is specified, the automatic resetting of the sum counter is overridden; instead, the sum counter is reset to zero only when a control break occurs for the control item specified in the RESET clause. Thus, data-name-3 must be the the name of a control item and must be a higher level control than the Control Footing which contains the RESET option. When RESET ON FINAL is specified, the counter is not reset, and any accompanying Control Footings are written upon execution of the TERMINATE statement.

PAGE-COUNTER AND LINE-COUNTER

PAGE-COUNTER is a fixed data-name specifying a compiler-generated counter to be used by the Report Writer. PAGE-COUNTER is initially set to 1 and is incremented by 1 each time a Page Break occurs.

LINE-COUNTER is also a fixed data-name generated by the Report Writer. It is used to format lines on the page and to determine when Page Headings and Footings should be produced. LINE-COUNTER is tested before each individual group is written and is reset to zero after every page break. At any given time, the value of LINE-COUNTER represents the number of the last line written or skipped to.

One PAGE-COUNTER and LINE-COUNTER are generated for each report. Both may be referred to in Procedure Division statements. Both must be qualified by the report-name if the program contains more than one report.

If a starting value for PAGE-COUNTER other than 1 is desired, it may be set by a Procedure Division statement after the INITIATE statement for the report has been executed.

LINE-COUNTER is used by the Report Writer for test and control purposes. Therefore, the programmer should be careful when changing its value by Procedure Division statements because the ensuing page format control may be unpredictable.

PROCEDURE DIVISION CONSIDERATIONS

To produce a report, the INITIATE, GENERATE and TERMINATE statements must be used in the Procedure Division. In addition, a USE BEFORE REPORTING sentence may be written in the Declaratives portion of the the Procedure Division. This option allows the programmer to manipulate or alter data immediately before it is written, or to suppress writing entirely.

INITIATE Statement

The INITIATE statement is used to initialize the PAGE-COUNTER and LINE-COUNTER before producing the report. The format of the INITIATE statement is:

```
┌──────────────────────────────────────┐
│INITIATE report-name...               │
└──────────────────────────────────────┘
```

The names of the reports to be initiated must have been defined by Report Description entries in the Report Section of the Data Division.

Only one INITIATE statement should be executed for each report-name. The INITIATE statement does not produce any written lines, nor does it open the file with which it is associated. An OPEN statement for the file must be given prior to the INITIATE statement for the report. The INITIATE statement must precede any GENERATE statements which refer to the report to be produced.

GENERATE Statement

A series of one or more GENERATE statements is executed to produce the report. The format of the GENERATE statement is:

```
┌──────────────────────────────────────┐
│GENERATE data-name                    │
└──────────────────────────────────────┘
```

Data-name may be the name of a Detail report group or the name of the report.

If data-name is a report-name, the Report Heading, Page Heading, and CONTROL HEADING FINAL will be produced if they have been specified. However, GENERATE report-name will not produce a Detail line.

If data-name is the name of a Detail report group, the GENERATE statement will recognize any control or page breaks and will produce the appropriate Headings and Footings. After any Control Headings and Footings are produced, the GENERATE statement will write the Detail line. All data items specified in the Detail report group will be accumulated into the sum counters before writing and the counters will be reset after writing unless a RESET clause has been specified. Any routines defined by a USE statement (see below) will be executed before generation of the associated report group.

If the first GENERATE data-name statement associated with a report refers to a Detail report group, Report and Page Headings and CONTROL HEADING FINAL, if specified, will be written before the first Detail line.

Data is moved into the data items in the Report Group Description entry of the Report Section, and is edited under the control of the Report Writer according to the same rules for movement and editing as described for the MOVE statement.

TERMINATE Statement

The TERMINATE statement is used to end a previously initiated report. It produces all of the Control Footings associated with the report as if a control break had occurred at the highest level. The format of the TERMINATE statement is:

```
┌──────────────────────────────────────┐
│TERMINATE report-name...              │
└──────────────────────────────────────┘
```

Although the TERMINATE statement completes all the Report Writer functions for a given report, it does not close the file with which the report is associated. A CLOSE statement for the file must be executed after the TERMINATE statement.

Only one TERMINATE statement may be executed for each report-name.

USE BEFORE REPORTING Sentence

A USE BEFORE REPORTING sentence is used to perform procedures immediately before a specified report group is produced. The USE sentence, when present, must immediately follow a section header in the Declaratives portion of the Procedure Division. The remainder of the section consists of one or more procedural paragraphs that define the procedures to be used.

The format of a USE BEFORE REPORTING declarative is:

```
DECLARATIVES.
Section-name SECTION.
        USE BEFORE REPORTING data-name.
          .
          .
          .
END DECLARATIVES.
```

Data-name is the name of a report group other than a Detail report grup. It must not appear in more than one USE statement. A separate USE statement must be written for each data-name.

Within a USE procedure there must not be any reference to any non-declarative procedures. Conversely, in the non-declarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE declarative or to the procedures associated with a USE declarative.

A common use of the USE sentence is to suppress writing of a specified report group under certain conditions. To do this, the statement

```
    MOVE 1 TO PRINT-SWITCH
```

is used in the USE BEFORE REPORTING declarative section. When this statement is encountered, the PRINT-SWITCH will be tested before printing takes place. Writing is suppressed if the switch is 1. After the USE BEFORE REPORTING section is executed, the PRINT-SWITCH is automatically reset to 0. The user should not reset it himself.

The following coding examples illustrate the use and operation of this declarative section. The coding below shows the definition of two Control Footing report groups in the Report Section of the Data Division.

```
01 MINOR TYPE CF C-1 LINE PLUS 2.

    02 A SUM P PICTURE $$$.99

        BLANK WHEN ZERO COLUMN 3.

    02 SUM Q PICTURE $$$99.99CR COLUMN 10.

    02 SOURCE E PICTURE **,***.**

        COLUMN 25.


01 TYPE CF C-2 LINE PLUS 1.

    02 SUM A PICTURE $$$$$.99

        BLANK WHEN ZERO COLUMN 5.
```

P, Q, and E are assumed to have been defined in either the File, Working-Storage, or Linkage Sections. The following coding shows a method of using the USE BEFORE REPORTING declarative to suppress writing of the above data under specified conditions.

```
DECLARATIVES.

RW SECTION.  USE BEFORE REPORTING MINOR.

    IF P = 0 AND Q = 0 MOVE 1 TO PRINT

    SWITCH.

END DECLARATIVES.
```

When a control break occurs for lower level control C-1, the following operations are performed:

1. The sum counter A is added to the sum counter defined in the second report group.

2. The RW Declarative Section is executed immediately before the 01 group for MINOR is entered. If P and Q are 0, the PRINT-SWITCH will be set to 1.

3. Upon entering the 01 group for MINOR, the PRINT-SWITCH is tested. If it is 1, steps 3 and 4 are bypassed.

4. The print line is constructed by:

   a. testing the line counter;

   b. moving the correct character to the first part of the record -- to cause a double space;

   c. moving the sum counters to the print line (edited) and

   d. moving E to the print line (edited).

5. The line is written and 2 is added to the LINE-COUNTER.

6. The sum counters in the first report group are set to zero.

7. If the PRINT-SWITCH was set to 1, it is reset to 0.

SAMPLE REPORT WRITER PROGRAM AND OUTPUT

Figures 16 and 17 illustrate a Report Writer source program and the report it would produce. In the program the input data begins after line 001170. The records are sorted by Date, Department Number and Type of Purchase.

In Figure 16, the program coding is found on the left-hand page. The coding is annotated on the right-hand page.

Sections of the report illustrated in Figure 17 are keyed by the circled numbers to a table after the figure. This table relates the sections to the statements in the program that produce them.

```
000000 IDENTIFICATION DIVISION.
000010 PROGRAM-ID. 'ACME'.
000020 INSTALLATION. ACME ACCOUNTING DEPARTMENT.
000030 REMARKS. THE REPORT WAS PRODUCED BY THE REPORT WRITER FEATURE.
000040 ENVIRONMENT DIVISION.
000050 CONFIGURATION SECTION.
000060 SOURCE-COMPUTER. IBM-360 F50.
000070 OBJECT-COMPUTER. IBM-360 F50.
000080 INPUT-OUTPUT SECTION.
000090 FILE-CONTROL.
000100     SELECT INFILE ASSIGN TO 'SYSIN'  UTILITY.
000110     SELECT REPORT-FILE ASSIGN TO 'SYSPRINT'  UTILITY.
000120 DATA DIVISION.
000130 FILE SECTION.
000140 FD  INFILE LABEL RECORDS ARE OMITTED DATA RECORD IS INPUT-RECORD
000150     RECORDING MODE F.
000180 01  INPUT-RECORD.
000190     02  FILLER          PICTURE AA.
000200     02  DEPT            PICTURE XXX.
000210     02  FILLER          PICTURE AA.
000220     02  NO-PURCHASES    PICTURE 99.
000230     02  FILLER          PICTURE A.
000240     02  TYPE-PURCHASE   PICTURE A.
000250     02  MONTH           PICTURE 99.
000260     02  DAY             PICTURE 99.
000270     02  FILLER          PICTURE A.
000280     02  COST            PICTURE 999V99.
000281     02  FILLER PICTURE X(59).
000290 FD  REPORT-FILE, REPORT IS EXPENSE-REPORT
000300     LABEL RECORDS ARE STANDARD.
000310 WORKING-STORAGE SECTION.
000320 77  SAVED-MONTH PICTURE 99 VALUE 0.
000330 77  SAVED-DAY   PICTURE 99 VALUE 0.
000340 77  CONTINUED   PICTURE X(11) VALUE SPACE.
000380 01  FILLER.
000390     02  RECORD-MONTH.
000400         03  FILLER PICTURE A(9) VALUE IS 'JANUARY  '.
000410         03  FILLER PICTURE A(9) VALUE IS 'FEBRUARY '.
000420         03  FILLER PICTURE A(9) VALUE IS 'MARCH    '.
000430         03  FILLER PICTURE A(9) VALUE IS 'APRIL    '.
000440         03  FILLER PICTURE A(9) VALUE IS 'MAY      '.
000450         03  FILLER PICTURE A(9) VALUE IS 'JUNE     '.
000460         03  FILLER PICTURE A(9) VALUE IS 'JULY     '.
000470         03  FILLER PICTURE A(9) VALUE IS 'AUGUST   '.
000480         03  FILLER PICTURE A(9) VALUE IS 'SEPTEMBER'.
000490         03  FILLER PICTURE A(9) VALUE IS 'OCTOBER  '.
000500         03  FILLER PICTURE A(9) VALUE IS 'NOVEMBER '.
000510         03  FILLER PICTURE A(9) VALUE IS 'DECEMBER '.
000520     02  RECORD-AREA REDEFINES RECORD-MONTH.
000530         03  MONTHNAME PICTURE A(9) OCCURS 12 TIMES.
000540 REPORT SECTION.
000550 RD  EXPENSE-REPORT CONTROLS ARE FINAL, MONTH, DAY
000560       PAGE 59 LINES HEADING 1 FIRST DETAIL 9 LAST DETAIL 48
000570         FOOTING 52.
000580 01  TYPE REPORT HEADING.
000590     02 LINE 1 COLUMN 27 PICTURE A(26) VALUE IS
000600     'ACME MANUFACTURING COMPANY'.
000610     02  LINE 3 COLUMN 26 PICTURE A(29) VALUE IS
000620     'QUARTERLY EXPENDITURES REPORT'.
000630 01  PAGE-HEAD TYPE PAGE HEADING LINE 5.
000640     02  COLUMN 30 PICTURE A(9) SOURCE MONTHNAME OF
000650     RECORD-AREA (MONTH).
000660     02  COLUMN 39 PICTURE A(12) VALUE IS 'EXPENDITURES'.
000661     02  COLUMN 52 PICTURE X(11) SOURCE CONTINUED.
000670     02  LINE 7 COLUMN 2 PICTURE X(35) VALUE IS
000680     'MONTH     DAY    DEPT  NO-PURCHASES'.
000690     02  COLUMN 40 PICTURE X(33) VALUE IS
000700       'TYPE        COST  CUMULATIVE-COST'.
000710 01  DETAIL-LINE TYPE DETAIL LINE PLUS 1.
000720     02  COLUMN 2 GROUP INDICATE PICTURE A(9)  SOURCE MONTHNAME
000730     OF RECORD-AREA (MONTH).
000740     02  COLUMN 13 GROUP INDICATE PICTURE 99 SOURCE DAY.
000750     02  COLUMN 19 PICTURE XXX SOURCE DEPT.
000760     02  COLUMN 31 PICTURE Z9 SOURCE NO-PURCHASES.
000770     02  COLUMN 42 PICTURE A SOURCE TYPE-PURCHASE.
000780     02  COLUMN 50 PICTURE ZZ9.99 SOURCE COST.
000790 01  TYPE CONTROL FOOTING DAY LINE PLUS 2.
000800     02  COLUMN 2 PICTURE X(22) VALUE IS 'PURCHASES AND COST FOR'.
000810     02  COLUMN 24 PICTURE Z9 SOURCE SAVED-MONTH.
000820     02  COLUMN 26 PICTURE X VALUE IS '-'.
000830     02  COLUMN 27 PICTURE 99 SOURCE SAVED-DAY.
000840     02  COLUMN 30 PICTURE ZZ9 SUM NO-PURCHASES.
000850     02  MIN COLUMN 49 PICTURE $$$9.99 SUM COST.
000860     02  COLUMN 65 PICTURE $$$$9.99 SUM COST RESET ON FINAL.
000870     02  LINE PLUS 1 COLUMN 2 PICTURE X(70) VALUE ALL '*'.
```

Figure 16. COBOL Program with Report Writer Feature (Part 1 of 2)

000290-000300 File Description entry for the file on which the report is to be written. REPORT clause lists the name of the report, "Expense-Report".

000320-000330 Source Data to be used after control and page breaks (see below, lines 000810, 000830, 000900, 000910).

000540 Report Section header. In this section the format of the report is defined.

000550 Report Description entry for the report. The three controls are listed in descending order of significance.

000560-000570 Page format: 59 lines deep; Heading on line 1; first Detail line on line 9; last Detail line on line 48; last Control Footing line on line 52. The Page Footing will be printed on line 59. The contents of these lines will be described in the Report Group Description entries.

000580-000620 Group description entry for the Report Heading. On line 1 of page 1 "Acme Manufacturing Company" will be printed, starting in column 27. Line 2 will be skipped, and line 3 will contain "Quarterly Expenditures Report", starting in column 26. These lines will be printed when the first GENERATE statement is executed.

000630-000700 Group Description entry for the Page Heading. A name, "Page-Head" is given to the entry because it will be referred to in the Declaratives Section of the Procedure Division. This group will be printed starting on line 5 after the first GENERATE statement and each time a page break occurs.

000710-000780 Group Description entry for the Detail group, the name of which is "Detail-Line". "Line plus 1" indicates that a) the group is to be printed on the next available line, and b) the entire group will be printed on one line, since only one line number is specified.

000720 The first item in the Detail group is to start in column 2 and is to be printed only by the first GENERATE statement executed after a control break (GROUP INDICATE). The source of the data to be printed is the appropriate entry of the array of month names that was defined in the Working-Storage Section of the Data Division (lines 000380-000530). "Monthname" is the name of 12 areas which redefine 03-level entries of "Record-Month". "(Month)" is the subscript. The remaining 02-level entries format the data from each input record (Input data is listed following line 001170).

000790-000870 This is the Control Footing for the minor control, "Day", and is printed each time "Day" changes. All of the information except the line of asterisks (line 000870) is printed two lines below the last Detail line. There are three sum counters. Two (lines 000840 and 000850) are reset after the break. One (line 000860) keeps a running total of cost for the quarter and is not reset until FINAL. Note that this counter is not fed data from "Min", which sums the daily expenditure (line 000850), but from cost, a field in the input record. The value of "Min" is added to "Int" (line 000930) before it is reset. Note also the the source for the day of the month (line 000830) is Saved-Day rather than Day. The control break for Day occurred when the value of Day changed. If Day was used as the source for its own Control Footing, the wrong date would always be printed. This is also why Saved-Month is the source for the month name (line 000810). When a control break occurs for Month, one is assumed for Day as well. Thus, the Control Footing for Day would cause the wrong month name to be printed if its source was Month.

```
000880 01  TYPE CONTROL FOOTING MONTH LINE PLUS 1 NEXT GROUP NEXT PAGE.
000890     02  COLUMN 16 PICTURE A(14) VALUE IS 'TOTAL COST FOR'.
000900     02  COLUMN 31 PICTURE A(9) SOURCE MONTHNAME OF RECORD-AREA
000910         (SAVED-MONTH).
000920     02  COLUMN 40 PICTURE AAA VALUE 'WAS'.
000930     02  INT COLUMN 46 PICTURE $$$9.99 SUM MIN.
000940 01  TYPE CONTROL FOOTING FINAL LINE PLUS 1.
000950     02  COLUMN 16 PICTURE A(26) VALUE IS
000960     'TOTAL COST FOR QUARTER WAS'.
000970     02  COLUMN 45 PICTURE $$$$9.99 SUM INT.
000950 01  TYPE PAGE FOOTING LINE 55.
001000     02  LINE 57 COLUMN 59 PICTURE X(12) VALUE IS, 'REPORT-PAGE- .
001010     02  COLUMN 71 PICTURE 99 SOURCE PAGE-COUNTER.
001020 01  TYPE REPORT FOOTING.
001030     02  LINE PLUS 1 COLUMN 32 PICTURE A(13) VALUE IS
001040     'END OF REPORT'.
001050 PROCEDURE DIVISION.
001060 DECLARATIVES.
001070 PAGE-HEAD-RTN SECTION. USE BEFORE REPORTING PAGE-HEAD.
001080   PAGE-HEAD-RTN-SWITCH. GO TO PAGE-HEAD-RTN-TEST.
001090   PAGE-HEAD-RTN-TEST. IF MONTH = SAVED-MONTH MOVE '(CONTINUED)'
001100     TO CONTINUED ELSE MOVE SPACES TO CONTINUED
001101     MOVE MONTH TO SAVED-MONTH.
001102     GO TO PAGE-HEAD-RTN-EXIT.
001103   PAGE-HEAD-RTN-SUPPRESS. MOVE 1 TO PRINT-SWITCH.
001104   PAGE-HEAD-RTN-EXIT. EXIT.
001110 END DECLARATIVES.
001120     OPEN INPUT INFILE, OUTPUT REPORT-FILE.
001121         READ INFILE AT END GO TO COMPLETE.
001130     INITIATE EXPENSE-REPORT.
001140 READATA. GENERATE DETAIL-LINE MOVE DAY TO SAVED-DAY
001141         READ INFILE AT END GO TO
001150             COMPLETE. GO TO READATA.
001160 COMPLETE. ALTER PAGE-HEAD-RTN-SWITCH TO PROCEED TO
001161                 PAGE-HEAD-RTN-SUPPRESS, TERMINATE EXPENSE-REPORT
001170     CLOSE INFILE, REPORT-FILE. STOP RUN.
  A00 02 A0101 00200
  A02  01 A0101 00100
  A02  02 C0101 01600
  A01  02 B0102 00200
  A04  10 A0102 01000
  A04  10 C0102 08000
  A01  02 B0105 00200
  A01  10 A0108 01000
  A01  08 B0108 01248
  A01  20 D0108 03840
  A01  06 C0329 04800
  A03  20 E0331 06000
  A03  10 G0331 05000
```

Figure 16. COBOL Program with Report Writer Feature (Part 2 of 2)

000880-000930 Control Footing entry for the major control "Month". Following the printing of this entry, a new page will be skipped to and a Page Heading will be printed before the next Detail line (however, printing of the Page Heading will be suppressed at the end of the report, as is seen below). Note that the source for the month name (lines 000900-000910) is Saved-Month rather than Month. A control break for Month occurs when its value changes. Thus, if Month was used as the source for its own Control Footing, the wrong month name would always be printed.

000940-000970 When the TERMINATE statement is executed, this group will be printed.

000980-001010 The Page Footing is printed automatically and the page counter incremented automatically each time a page condition is detected.

001020-001040 This group is the last to be printed.

001070-001110 These statements, when executed, cause alteration in, or suppression of the printing of the Heading. The various statements are executed after a page break or control break for "Month", and before the Page Heading is printed.

001080 This paragraph is included because the GO TO is altered under certain conditions (lines 001160-001161).

001090-001102 These lines test the cause of the break. If it was a simple page break, the month will not have changed, and "(Continued)" will be included in the Page Heading (see lines 000630-000700 for Page Heading format). However, since the report for each month starts at the top of a new page, the new month name must be included in the Page Heading after a control break for Month.

001103 Upon completion of the report, this instruction is executed to suppress the printing of a Page Heading (see lines 001160-001161).

001120-001170 These statements read in data, generate Detail lines of the report, and close files after

001120 processing. First, input and output files are opened

001121 and the first data record

001130 is read. Then the INITIATE statement for the report is executed.

001140-001150 This loop generates all Detail lines and reads cards 2 - n. The control for "Day" uses the day stored in "Saved-Day" as part of its output.

001160-001170 When all the data records have been read, the report is terminated. This involves printing the Report Footing on a new page without printing a Page Heading.

Following 001170 This is the input data used in the report. Using the first record as an example, the data fields are arranged in the following format:

Department Number	Number of Purchases	Type of Purchase
A00	C2	A

Month	Day	Cost
01	01	00200

The decimal point in the cost field is assumed to be two places from the right.

The input data are sorted by Date, Department Number and Type of Purchase (minor key).

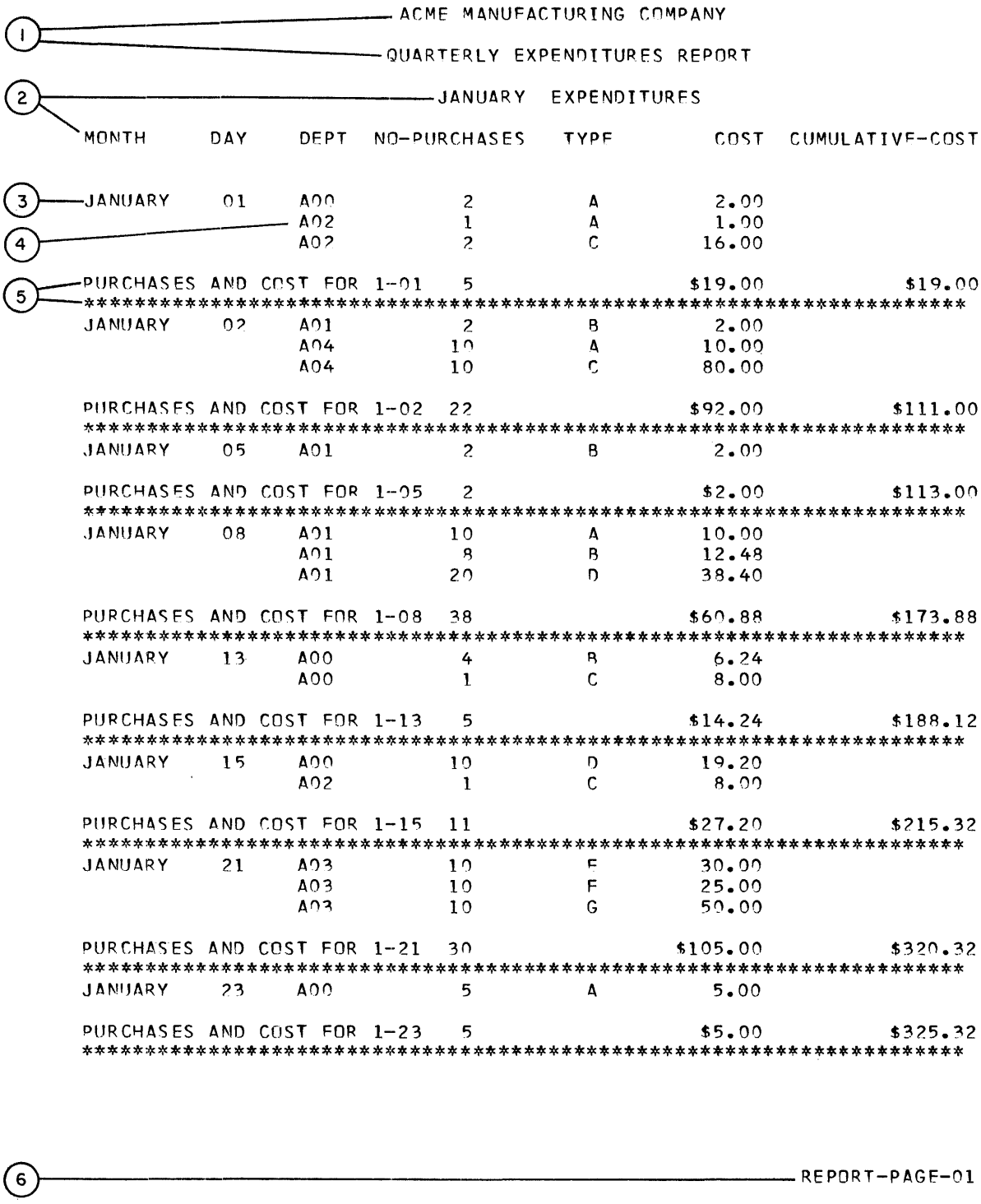


Figure 17. Report Produced by Report Writer Feature (Part 1 of 5)

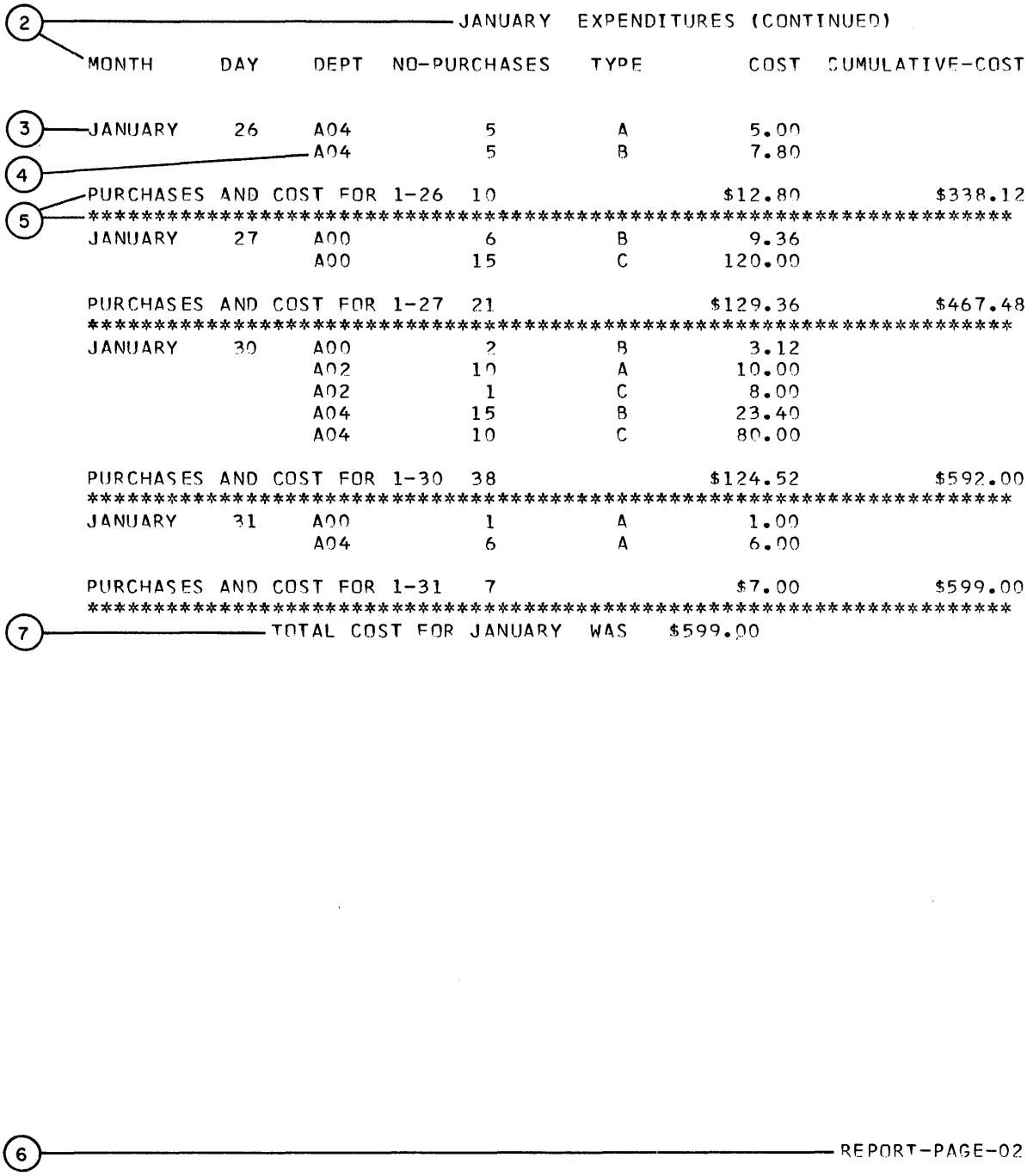


Figure 17. Report Produced by Report Writer Feature (Part 2 of 5)

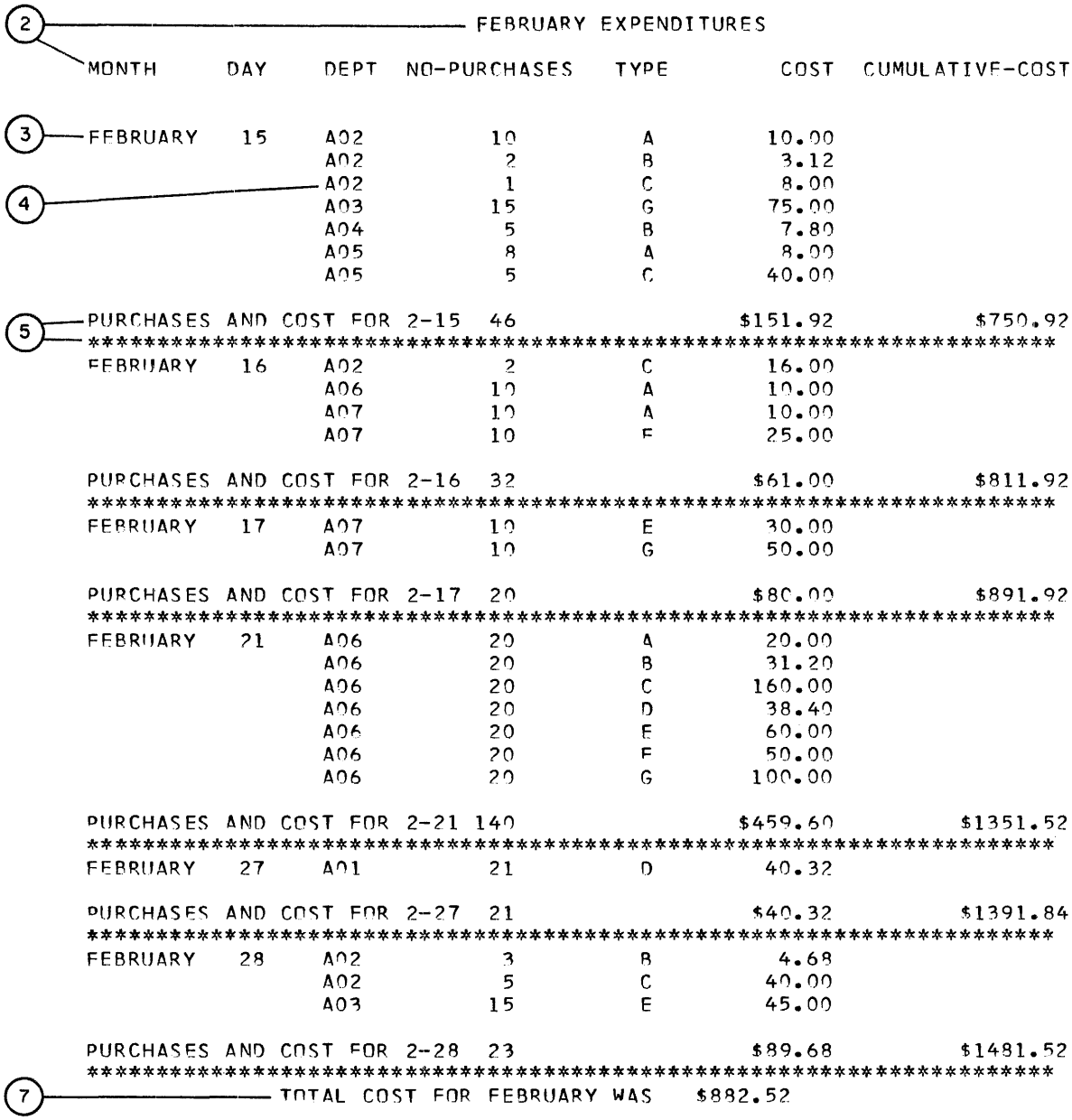


Figure 17. Report Produced by Report Writer Feature (Part 3 of 5)

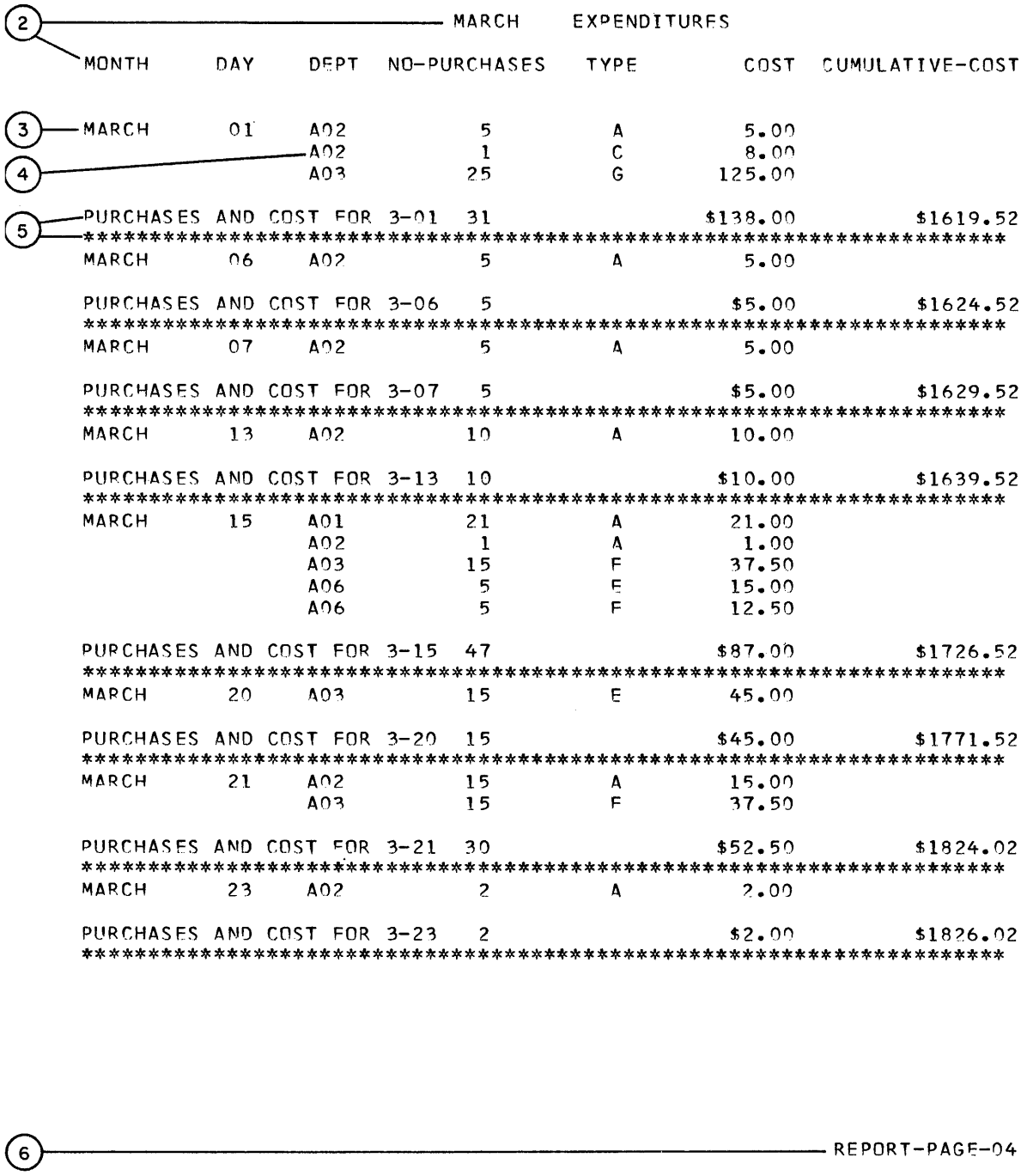


Figure 17. Report Produced by Report Writer Feature (Part 4 of 5)

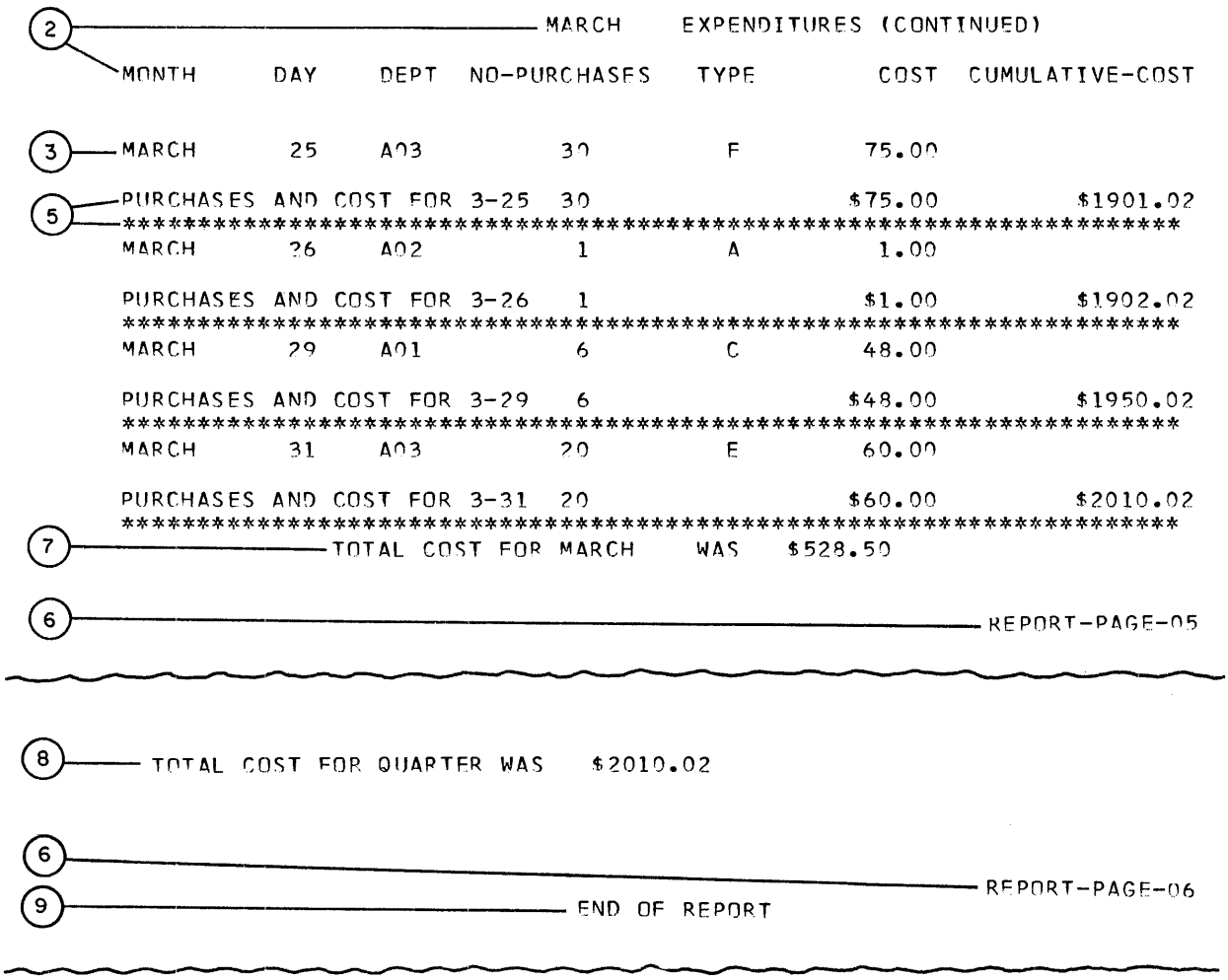


Figure 17. Report Produced by Report Writer Feature (Part 5 of 5)

Key Relating Report to Report Writer Source Program

① is the Report Heading resulting from source lines 00580-00620.

② is the Page Heading resulting from source lines 00630-00700.

③ is the Detail line resulting from source lines 00710-00780 (note that since it is the first Detail line after a control break, the fields defined with 'group indicate' lines 00720-00740, appear).

④ is a Detail line resulting from the same source lines as 3. In this case, however, the fields described as 'group indicate' do not appear (since the control break did not immediately precede the Detail line).

⑤ is the Control Footing (for Day) resulting from source lines 00790-00870.

⑥ is the Page Footing resulting from source lines 00980-01010.

⑦ is the Control Footing (for MONTH) resulting from source lines 00880-00930.

⑧ is the Control Footing (for FINAL) resulting from source lines 00940-00970.

⑨ is the Report Footing resulting from source lines 01020-01040.

Lines 01070-01104 of the example illustrate a use of 'USE BEFORE REPORTING'. The effect of the source is that each time a new page is started, a test is made to determine if the new page is being started because a change in Month has been recognized (the definition for the Control Footing for Month specifies 'NEXT GROUP NEXT PAGE') or because the physical limits of the page were exhausted. The calculation involved sets up a fixed ('PAGE GROUP') which is referenced by a SOURCE clause in the Page Footing description. Consequently, two page counters can be maintained: one indicating physical pages and one indicating logical pages.

The COBOL programmer can obtain convenient access to the SORT feature of the Operating System/360 Sort/Merge Program by writing a COBOL SORT statement and other elements of the COBOL SORT feature in his source program. The SORT feature permits him to request execution of a sorting operation within his program and to specify in that operation the sorting of fixed-length or variable-length records that may be in binary, floating-point, packed decimal, or EBCDIC mode. He can also summarize, insert, delete, shorten, or otherwise alter records during the initial and final phases of the sort.

The basic elements of the SORT feature are the SORT statement in the Procedure Division and the sort-file description entry with its associated record description entries in the Data Division. A sort is based on sort-keys named in the SORT statement. Sort-keys are defined only in the record descriptions associated with the sort-file-description. When the SORT feature is being executed, these record descriptions may be considered as redefining the records being sorted. The records may be sorted in ascending or descending order or in a mixture of the two; that is, the sort-keys may be specified as ascending or descending independently of one another, and the order of the records will conform to the mixture specified. The standard sequential file processing technique is required for use of the SORT feature. (Additional information concerning the SORT feature is contained in the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form C28-6380.)

ELEMENTS OF THE SORT FEATURE

To make use of the SORT feature, the programmer must provide certain information in the Environment, Data, and Procedure Divisions of the COBOL source program.

In the Environment Division, he must write SELECT sentences for files used as input and output to the Sort/Merge Program whenever a USING or GIVING option (described in "SORT Statement") is to be used in the Procedure Division.

In the Data Division, the programmer writes File Description entries for all files that are used to provide input to or output from the SORT feature. He must also write a Sort Description entry to describe the records that are to be sorted, including their sort-key fields.

In the Procedure Division the programmer specifies the records to be sorted, the sort-key names, whether the sort is to be in ascending or descending sequence by key, and whether records are to be processed before or after the sort. If there is to be such processing, he also includes in the Procedure Division the program sections that perform the processing.

Sort Work Files

At least three files must be available for use as intermediate work files by the Sort/Merge Program. A discussion is contained in the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form C28-6380.

ENVIRONMENT DIVISION STATEMENTS

SELECT entries must be included in the FILE-CONTROL paragraph of the Input-Output Section in the Environment Division for any files named in the USING or GIVING options in the Procedure Division. In this case, all options for standard sequential files are permitted with the restrictions indicated for the SELECT entry (see "File Processing Techniques"). Note that no SELECT entry is permitted for the sort-file description entry itself.

SELECT Entry

The format of the SELECT entry as used with the SORT feature is:

```
SELECT file-name
  ASSIGN TO external-name {DIRECT-ACCESS}
                          {UTILITY      }

 [device-number UNIT [S]]

 [ACCESS IS SEQUENTIAL]

 [RESERVE-clause]
```

The external-name must be 'SORTIN' or 'SORTOUT'. (For use of other external-names see the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form C28-6380.) 'SORTIN' refers to the sort input file and is related to the USING option; 'SORTOUT' refers to the sort output file and is related to the GIVING option.

DATA DIVISION STATEMENTS

Sort Description entries and related Record Description entries must be written in the File Section of the Data Division (in addition to the File Description entries for input/output files).

Sort Description Entry

The Sort Description entry provides data descriptions of the records to be processed by the SORT feature. The Sort Description entry and its record description (see "Record Description Entry," which follows) are similar in structure to a File Description entry and its record description. Unlike the File Description, however, the Sort Description does not describe any physical file; rather, it provides a kind of "template" that the SORT feature can apply to all of the real files that are involved in the sorting operation. The format of the Sort Description entry is:

```
SD sort-file-description-name

 [RECORDING MODE IS mode]

 [RECORD CONTAINS [integer-1 TO]

  integer-2 CHARACTERS]

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

Sort-file-description-name is the name the programmer assigns for communication with the SORT feature. It follows the rules for data-name, and must not be subscripted or qualified.

RECORDING MODE IS specifies the format of the logical records that are to be sorted.

Mode must be either F or V. F specifies that all records are of the same fixed length; V should be used when the records are either variable length (contain an OCCURS DEPENDING ON clause) or are of differing fixed lengths. If neither is specified, V is assumed. (For further information see "Record Formats" in the chapter "Data Division.")

The RECORD CONTAINS and DATA RECORDS clauses are described under "File Section Entries" in the chapter "Data Division."

The SORT feature assumes that all records to be sorted are described under the Sort Description entry for the particular sort-file-description-name given in the SORT statement. For example, if the input is provided by the USING clause of the SORT statement, it is assumed that the record descriptions given under the Sort Description entry of the sort-file-description-name describe the input records, regardless of how they were described under that input file's File Description entry. Similarly, the Sort Description record descriptions are assumed to describe output file records (the result of using the GIVING clause of the SORT statement) and all other records involved in the sorting operation. Thus, the Sort Description entry does not describe any one physical file, but rather provides an overriding description for all files involved in the sorting process. Accordingly, a SELECT clause should not be given for Sort Description entries.

Record Description Entry

The format of a Record Description entry will vary according to the type of item being described (see "Group Item Format" and "Formats for Elementary Items" in the chapter "Data Division").

SORT-KEYS: Sort-keys must have a fixed length, must not be subscripted, and must be one of the types of data items listed below. A collating sequence that corresponds to each type of data item is used for sorting (see Table 19).

A character in the EBCDIC collating sequence (used with alphabetic, alphanumeric, etc., data items) is interpreted as not being signed. For packed decimal, zoned decimal, fixed-point, and floating-point data, characters are collated algebraically (that is, as being signed).

Table 19. Collating Sequence for Specific Data Items

Data Item	Collating Sequence
Internal floating point	Floating point
External decimal (not exceeding 16 characters)	Zoned Decimal
Internal decimal	Packed Decimal
Binary	Fixed point
Alphabetic	EBCDIC
Alphanumeric	EBCDIC
Group	EBCDIC
External floating point	EBCDIC
Report	EBCDIC

Each of the preceding types of data items is discussed in detail under "Types of Data Items" in the chapter "Data Division."

Sort-keys must be in the same relative position and must be of the same length within each record, and that position cannot be more than 4,092 bytes from the beginning of the record. (The relative position of the sort-keys can be determined by using the formulas discussed under the USAGE clause in the section "Data Division.") Records are limited to 9999 bytes in length. From one to twelve keys may be specified for a file. Sort-keys may overlap, but they must not coincide in position; that is, no two sort-keys may have precisely the same displacement and length within a record. The sum of the lengths of all the sort-keys must not exceed 256 bytes. An external decimal sort-key must not exceed 16 digits.

Sort-keys are identified by the data-names assigned to each of the fields upon which the sorting is to be based. This is accomplished by the Record Description entries that specify the characteristics of each section (Record Description entries are described in the chapter "Data Division").

When assigned to sort-keys, data-names may be qualified but must be unique. Also, once a data-name has been assigned to a sort-key, it is assumed that the key appears in the same location in every record that is to be sorted.

File Description Entry

If the USING or GIVING options of the SORT statement are used, a File Description entry must be given for each file named in these options.

A detailed description of each of the clauses, together with applicable additional information concerning their use with standard sequential files, is contained under "File Section" and "File Section Entries" in the chapter "Data Division."

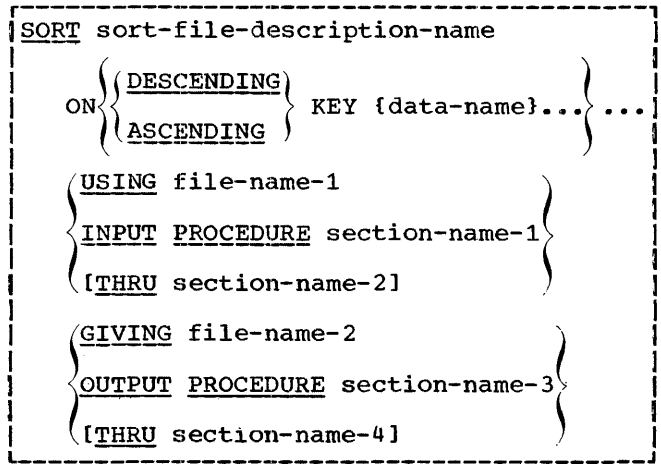
PROCEDURE DIVISION STATEMENTS

The Procedure Division contains a SORT statement to describe the sorting operation, and INPUT or OUTPUT PROCEDUREs to describe any processing to be performed.

SORT Statement

The SORT statement provides information that controls the SORT feature. The SORT feature uses this information to obtain records to be sorted from either an input procedure or an input file, sorts the records on a set of specified keys, and in the final phase of the sort operation, makes each record available, in sorted order, either to some output procedure or to an output file.

The SORT statement specifies: (1) the sort-file-description-name assigned to the Sort Description entry, (2) whether the records are to be sorted into an ascending or descending sequence in relation to each sort-key, (3) the data-names that have been assigned to the sort-keys, and (4) the section names of any input or output procedures that are to be used before or after the sorting operation or the names of the input or output files. The format of the SORT statement is:



Sort-file-description-name is the name given in the Sort Description entry which describes the records to be sorted.

ASCENDING and DESCENDING specify whether the records are to be sorted, respectively, into an ascending or descending sequence, based on one or more sort-keys. The sequence specified is applicable to all sort-keys immediately following the clause. One of these clauses must be specified. Both may be specified in the same statement.

Data-name is the data-name assigned to a sort-key and is required. More than one data-name may be specified following either ASCENDING or DESCENDING. If more than one data-name is specified, the key associated with the data-name specified first is checked first, etc. Every data-name must have been defined in a Record Description entry associated with the Sort Description entry. The same data-name or coinciding data-names must not be used twice in the same SORT statement. Sort-keys must be specified in the logical order in which they are referred to during the sorting operation. This means that the key that is to be checked first is logically specified first, the key that is to be checked next is logically specified second, etc. The sort-keys are specified by entering, in the desired order, the data-names assigned to them in the SORT statement.

USING indicates that the records that are to be sorted are on file-name-1 and that they are all to be passed to the sorting operation as one file when the SORT statement is executed. If the programmer specifies the USING option, all the records to be sorted must be in the same file. This file will be automatically opened, read, and closed by the SORT feature without the programmer's having to provide any additional coding.

File-name-1 is the name of the file that contains the records that are to be sorted. This file must have been described by a File Description entry. It must have standard sequential organization and can reside on any input/output device that operates with QSAM. The file must be named in a SELECT sentence (See "Input-Output Section"), and the external-name in the ASSIGN clause must be 'SORTIN'.

INPUT PROCEDURE indicates that the programmer has written an input procedure to process records before they are sorted and has included the procedure in the Procedure Division in the form of one or more distinct sections. The input procedure passes records one at a time to the SORT feature after it has completed its processing.

Section-name-1 is the name of the first, or only, section in the COBOL main program that contains the input procedure. It is required if INPUT PROCEDURE is used.

Section-name-2 is the name of the last section that contains the input procedure in the COBOL main program. It is required if the procedure terminates in a section other than that in which it is started.

GIVING indicates that, after the records have been sorted, they are to be written as a file on file-name-2. If the programmer specifies the GIVING option, all records that have been sorted will be placed on one file. This file will be automatically opened, written into, and closed by the SORT feature without the programmer's having to provide additional coding.

File-name-2 is the name of the file on which the records are to be written after they have been sorted. This

must have been described by a File Description entry. The file must be named in a SELECT sentence and the external-name entered in the ASSIGN clause of the SELECT sentence. The external-name must be 'SORTOUT' except in the special case where file-name-1 and file-name-2 refer to the same file. In this case the external-name in the ASSIGN clause for the file will be 'SORTIN'. For details on execution DD statement requirements, refer to the publication IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form C28-6380. This file must have standard sequential organization and can reside on any input/output device that operates with QSAM.

OUTPUT PROCEDURE indicates that the programmer has written an output procedure to process records after they have been sorted and has included the procedure in the Procedure Division in the form of one or more distinct sections. The output procedure returns the records one at a time from the SORT feature after they have been sorted.

Section-name-3 is the name of the first, or only, section in the main COBOL program that contains the output procedure.

Section-name-4 is the name of the last section that contains the output procedure in the main program. It is required if the procedure terminates in a section other than that in which it is started.

Note: The SORT statement and the input and output procedure sections are permitted in any part of the Procedure Division, excluding the declarative sections.

The Input Procedure

The input procedure consists of one or more sections that are written in the program. The input procedure can include any statements needed to select, create, or modify records. Control must not be passed to the input procedure except when a related SORT statement is being executed, since the RELEASE statements (see "RELEASE Statement" below) in the input procedure have no meaning unless they are controlled by a SORT statement.

There are two restrictions on the procedural statements within the input procedure, as follows:

1. The input procedure must not contain any SORT or CALL statements.

2. Any files in use as sort work files may not be opened or referred to in the input procedure. If file-name-2 is not a work file, it may be opened and referred to, provided that it is closed again before the end of the input procedure.

The programmer must code the input procedure so that it incorporates three specific functions:

1. It must build the records that are to be sorted, one at a time, in the data record that has been described and assigned data-names in the Record Description entry associated with the Sort Description. This can be accomplished by using COBOL instructions such as READ...INTO... or MOVE. If the input is to come initially from a file, the programmer must write an OPEN statement to open the file prior to executing the SORT statement or in the input procedure itself.

2. Once a record has been processed, the input procedure must make that record available to the sorting operation by means of the RELEASE statement (see "RELEASE Statement" below), after which the record just built is no longer available and either step 1 or step 3 is performed next.

3. When all the records have been released, the input procedure must direct control to the last statement in the procedure in order to terminate the procedure (see "Control of Input and Output Procedures").

RELEASE Statement

The RELEASE statement causes one record to be transferred to the sorting operation. It can only appear in an input procedure. If an input procedure is specified, the RELEASE statement must be included in that procedure. The format of the RELEASE statement is:

```
RELEASE record-name
```

Record-name is the name of the data record that has been described and assigned data-names in the Record Description entry associated with the Sort Description entry. The record that is to be sorted is automatically trans-

ferred to the sorting operation. The record-name must be defined at the 01 level under the Sort Description entry.

The Output Procedure

The output procedure consists of one or more sections that are written in the program. The output procedure may consist of any statements needed to select, modify, or copy the records that are being returned one at a time, in sorted order, from the sort-file. Control must not be passed to the output procedure except when a related SORT statement is being executed, since the RETURN statements (see "RETURN Statement" below) in the output procedure have no meaning unless they are controlled by a SORT statement.

There are two restrictions on the procedural statements within the output procedure:

1. The output procedure must not contain any SORT or CALL statements.
2. Any files in use as sort work files may not be opened or referred to in the output procedure. Note that if file-name-1 is not a sort work file, it is not affected by this restriction.

The programmer must code the output procedure so that it incorporates three specific functions:

1. It must obtain sorted records, one at a time, from the Sort/Merge Program via the RETURN statement. Once a record has been returned, the previously returned record is no longer available.
2. The output procedure must manipulate the record just returned by referring to the data record that has been described and assigned data-names in the Record Description entry. Such manipulation, for example, including a record in a summary to be made of all sorted records, may make use of COBOL instructions such as MOVE and WRITE... FROM. If records are to be written into an output file, the programmer must write an OPEN statement to open the file prior to execution of the SORT statement or in the output procedure itself.
3. When the SORT feature has returned all records and the output procedure attempts to execute another RETURN statement (as in step 1), the AT END clause of the RETURN statement will be executed. The imperative statement in the AT END clause must direct control to the last statement of the output procedure in order to terminate the output procedure (see "Control of Input and Output Procedures").

RETURN Statement

The RETURN statement causes individual records to be obtained from the sorting operation after all the records have been sorted, and indicates what action is to be taken with regard to each. The format of the RETURN statement is:

```
RETURN sort-file-description-name
    AT END imperative statement...
```

Sort-file-description-name is the name given in the Sort Description entry which describes the records to be sorted.

Imperative statement specifies what is to be done once all the sorted records have been disposed of. (Imperative statements are discussed in detail earlier in this publication in the chapter "Procedure Division.")

Control of Input/Output Procedures

The INPUT and OUTPUT PROCEDURE clauses function in a manner similar to the PERFORM statement (option 1); for example, naming one or more sections in an INPUT PROCEDURE clause causes execution of that section during the sorting operation to proceed as if that section had been the subject of a PERFORM statement. As in the PERFORM, the execution of the section is terminated after execution of its last statement.

The STOP RUN statement may not appear in a Sort Input or Output procedure. Certain other procedure branching statements may be used with restrictions. See Table 18 in

the chapter "Procedure Division" for specific qualifications.

The EXIT statement may be used as a common end point for input and output procedures as with the PERFORM statement. When used in this manner, it must appear as the only statement in the last paragraph of the input or output procedure.

```
paragraph-name.  EXIT.
```

CONTROL FLOW

The sequence through which control passes during a sorting operation is shown in Figure 18.

Examples of a SORT Statement

The following text contains three examples of SORT statements. The next section, "Sample Program Using the Sort Feature," contains an example of a complete Sort program.

EXAMPLE 1: Consider a group of records which represent sales to customers on various days. If they are initially in no particular order and they are to be placed in order by customer number with the latest date first within each customer's records, the hierarchy of the sort becomes:

```
CUSTOMER-NUMBER, DATE
```

Since the lowest customer number is to come first and the highest last, the order for customer number is ascending, but for date the latest date (highest numeric) comes first and the earliest date (lowest numeric) comes last so the order for date is descending. This can be stated as:

```
ASCENDING KEY CUSTOMER-NUMBER, DESCENDING
  KEY DATE
```

If, in addition, the unsorted records have been written on a file called FN-1 which is to be used by the sorting process and the sorted records are desired to be given on FN-2, this sorting requirement can be stated as:

```
SORT SALES-RECORDS ON ASCENDING KEY
  CUSTOMER-NUMBER, DESCENDING KEY DATE,

     USING FN-1,

     GIVING FN-2.
```

The name SALES-RECORDS and the two data-names must be defined in a Sort Description entry, while the file-names FN-1 and FN-2 must be defined in File Description entries.

EXAMPLE 2: The following is an example containing an input procedure that could be used in connection with a company net-sales contest to exclude the records of salesmen in two departments.

```
ELIM-DEPT-7-9-NO-PRINTOUT.
         SORT NET-FILES ASCENDING KEY DEPT,
           DESCENDING KEY NET-SALES, INPUT
           PROCEDURE SCREEN-DEPT, GIVING
           NET-FILE-OUT.
       STOP RUN.
SCREEN-DEPT SECTION.
S-D-1.  OPEN INPUT NET-FILE-IN.
S-D-2.  READ NET-FILE-IN AT END GO TO
          S-D-FINAL.
S-D-3.  IF DEPT-IN = 7 OR 9 GO TO S-D-2
          ELSE MOVE NET-CARD-IN TO
             SALES-RECORDS, RELEASE
             SALES-RECORDS, GO TO S-D-2.
S-D-FINAL.  CLOSE NET-FILE-IN.
```

EXAMPLE 3: The following is a variation on the net-sales contest example referred to and illustrates a simple report produced by the use of the DISPLAY verb. The following illustrates how this may be done in an output procedure:

```
SORT-NET-SALES SECTION.

S-N-S-1.        SORT NET-FILES ON ASCENDING
                  KEY DEPT,

                DESCENDING KEY NET-SALES,

                USING NET-SALES-IN,

                OUTPUT PROCEDURE REPORT-NET.

S-N-S-2.        STOP RUN.

REPORT-NET SECTION.

R-N-1-1.        DISPLAY 'NET SALES REPORT.'.

R-N-1-2.        RETURN NET-FILES AT END GO
                  TO R-N-1-3.

                DISPLAY NET-SALES, SPACE,
                  DEPT, SPACE, EMPL-NO,
                  SPACE, NAME-ADDR.

                GO TO R-N-1-2.

R-N-1-3.        EXIT.
```

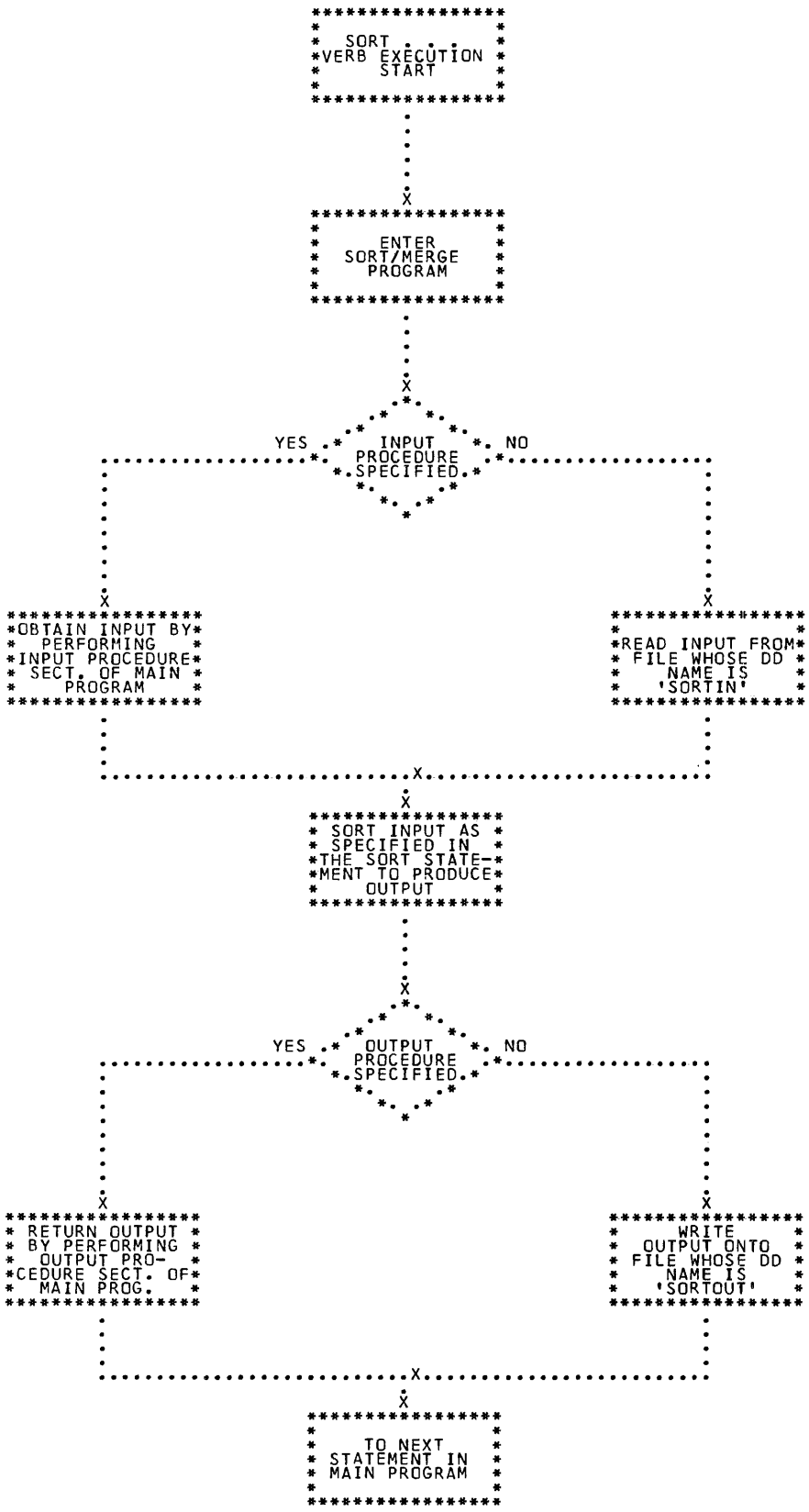


Figure 18. Flow of Data Through a Sorting Operation

Sample Program Using the Sort Feature

The example in Figure 19 illustrates a sort based on a sales contest. The records to be sorted contain data on salesmen: name and address, employee number, department number, and pre-calculated net sales for the contest period.

The salesman with the highest net sales in each department wins a prize, and smaller prizes are awarded for second highest sales, third highest, etc. The order of sort is (1) by department, the lowest numbered first; and (2) by net sales within each department, the highest net sales first.

The records for the employees of departments 7 and 9 are eliminated before sorting begins. The remaining records are then sorted, and the output is placed on another file for use in a later job step.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  'CONTEST1'.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
             SELECT NET-FILE-IN          ASSIGN TO 'SYSIN' UTILITY.
             SELECT NET-FILE-OUT         ASSIGN TO 'SORTOUT' UTILITY.
       DATA DIVISION.
       FILE SECTION.
       FD    NET-FILE-IN  RECORDING MODE IS U LABEL RECORDS
                ARE OMITTED, DATA RECORD IS NET-CARD-IN.
        01   NET-CARD-IN.
          02    EMPL-NO-IN           PICTURE 9(6).
          02    DEPT-IN              PICTURE 9(2).
          02    NET-SALES-IN         PICTURE 9(7)V99.
          02    NAME-ADDR-IN         PICTURE X(55).
       FD    NET-FILE-OUT RECORDING MODE IS F LABEL RECORDS
                ARE OMITTED, DATA RECORD IS NET-CARD-OUT.
        01   NET-CARD-OUT.
          02    EMPL-NO-OUT          PICTURE 9(6).
          02    DEPT-OUT             PICTURE 9(2).
          02    NET-SALES-OUT        PICTURE 9(7)V99.
          02    NAME-ADDR-OUT        PICTURE X(55).
       SD    NET-FILES RECORDING MODE F DATA RECORDS SALES-RECORDS.
        01    SALES-RECORDS.
          02    EMPL-NO   PICTURE 9(6).
          02    DEPT      PICTURE 9(2).
          02    NET-SALES PICTURE 9(7)V99.
          02    NAME-ADDR PICTURE X(55).

       PROCEDURE DIVISION.

01350  ELIM-DEPT-7-9-NO-PRINTOUT.
01360            SORT NET-FILES ASCENDING KEY DEPT, DESCENDING KEY
01361          NET-SALES, INPUT PROCEDURE SCREEN-DEPT,
01370          GIVING NET-FILE-OUT.

       CHECK-RESULTS SECTION.
       C-R-1.  OPEN INPUT NET-FILE-OUT.
       C-R-2.     READ NET-FILE-OUT AT END GO TO C-R-FINAL.
             EXHIBIT NAMED EMPL-NO-OUT DEPT-OUT NET-SALES-OUT
                   NAME-ADDR-OUT.
       C-R-3.     GO TO C-R-2.
       C-R-FINAL.
                CLOSE NET-FILE-OUT.  STOP RUN.

01390  SCREEN-DEPT SECTION.
       S-D-1.    OPEN INPUT NET-FILE-IN.
       S-D-2. READ NET-FILE-IN AT END GO TO S-D-FINAL.
             EXHIBIT NAMED EMPL-NO-IN  DEPT-IN  NET-SALES-IN
                   NAME-ADDR-IN.
01420  S-D-3.    IF DEPT-IN = 7 OR 9 GO TO S-D-2
01430               ELSE MOVE NET-CARD-IN  TO SALES-RECORDS,
01440                 RELEASE SALES-RECORDS, GO TO S-D-2.
01450  S-D-FINAL.  CLOSE NET-FILE-IN.
```

Figure 19. COBOL Program Using Sort Feature

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:

Format 1 (within the Configuration Section):

```
SPECIAL-NAMES.[1] COPY library-name.
```

Format 2 (within the Input-Output Section):

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

Format 3 (within the File-Control paragraph):

```
SELECT file-name COPY library-name.
```

Format 4 (within the File Section):

```
{FD file-name       }
{SD sort-file-name[1]} COPY library-name.
```

Format 5 (within the Report Section):[1]

```
RD report-name [WITH CODE mnemonic-name]
    COPY library-name.
```

Format 6 (within a File or Sort Description entry, or within the Working-Storage or Linkage Section):

```
01 data-name COPY library-name.
```

Format 7 (within a Report Description[1] entry):

```
01 [data-name] COPY library-name.
```

Format 8 (within the Working-Storage or Linkage Section):

```
77 data-name COPY library-name.
```

Library-name is the name of a member of a partitioned data set contained in the user's library; it identifies the entries to be copied. Library-name is an external-name; therefore, it must be enclosed within single quotation marks and contain no more than eight alphabetic and numeric characters, the first of which must be alphabetic.

The words preceding COPY conform to margin restrictions for COBOL programs. 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.

A COPY clause with one of the required formats may be written on one or more cards.

When formats 1, 2, 3, or 4 are written, the words COPY library-name are replaced internally 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 formats 5, 6, 7, or 8 are written, the entire entry is replaced by the information identified by library-name, except that data-name (if specified) replaces the corresponding data-name from the library.

---

[1]Implemented for COBOL F only

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 occurs during compilation but is not shown in the listing.

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:

Format 1 (For insertion of a paragraph):

```
paragraph-name.  INCLUDE library-name.
```

Format 2 (For insertion of a section):

```
section-name SECTION.  INCLUDE library-
    name.
```

Library-name is the name of a member of a partitioned data set contained in the user's library. It 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.

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.

F ONLY | EXT | EXTENDED SOURCE PROGRAM LIBRARY FACILITY

A complete program may be included as an entry in the user's library, and may be used as the basis of compilation. Input to the compiler is a BASIS card, followed by any number of INSERT and/or DELETE cards, and followed by any number of debugging packets, if desired. Debugging packets are described in the chapter entitled "COBOL Debugging Language."

The format of the BASIS card is:

```
1      8
BASIS  library-name
```

Library-name is an external name: it names the complete program entry used as a basis for the compilation.

If INSERT or DELETE cards follow the BASIS card, the program entry is temporarily modified prior to being processed by the compiler. The library member itself is not affected and can be modified permanently only with a utility program. See IBM System/360 Operating System: COBOL (F) Programmer's Guide, Form GC28-6380.

The format of the INSERT card is:

```
1      8-72
INSERT sequence-number-field
```

The format of the DELETE card is:

```
1      8-72
DELETE sequence-number-field
```

Each number in the sequence-number-field must refer to a sequence number of the basic library entry.

The sequence-number-field of an INSERT card must be a single number. At least one new source program card must follow the INSERT card, for insertion after the card specified by the sequence-number-field.

The entries comprising sequence-number-field of a DELETE card must be numbers or ranges of numbers. Each entry must be separated from the preceding entry by a comma followed by a space. Ranges of numbers are indicated by separating the two bounding numbers of the range by a hyphen (e.g., 000001-000005). Source program cards may follow a DELETE card, for insertion before the card following the last one deleted.

[EXT] STERLING CURRENCY FEATURE AND INTERNATIONAL CONSIDERATIONS

TERLING CURRENCY FEATURE

System/360 COBOL provides facilities for andling sterling currency items by means f an extension of the PICTURE clause. dditional options and formats, necessiated by the non-decimal nature of stering, and by the conventions by which terling amounts are represented in punched ards, are also available.

System/360 COBOL provides a means to xpress sterling currency in pounds, shilings, and pence, in that order. There are 0 shillings in a pound, and 12 pence in a hilling. Although sterling amounts are ometimes expressed in shillings and pence nly (in which case the number of shillings ay exceed 99), within machine systems hillings will always be expressed as a -digit field. Pence, when in the form of ntegers, likewise will be expressed as a -digit field. However, provision must be ade for pence to be expressed as decimal ractions as well, as in the form 7s.10.237d.

The IBM method for representing sterling mounts in punched cards uses two columns or shillings and one for pence. Tenpence 10d.) is represented by an '11' punch and levenpence (11d.) by a '12' punch. The ritish Standards Institution (B.S.I.) epresentation uses single columns for both illings and pence. The B.S.I. representation for shillings consists of a '12' unch for ten shillings and the alphabetic unches A through I for 11 to 19 shillings, espectively.

ote: The B.S.I. representation for shilings precludes the use of more than 19 illings in a sterling expression; therefore, 22/10 (that is, 22 shillings 10 ence) must be expanded, by the user, to 2/10. Similarly, the guinea -- 21 shil.ngs -- or any multiple thereof, must be panded to pounds and shillings.

The indicated representations may be sed separately or in combination, resultng in four possible conventions.

.. IBM shillings and IBM pence

2. IBM shillings and B.S.I. pence

3. B.S.I. shillings and IBM pence

4. B.S.I. shillings and B.S.I. pence

Any of these conventions may be associated with any number of digits, (or none) in the pound field and any number of decimal places (or none) in the pence field. 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 System/360 COBOL in the PICTURE clause for the representation of sterling amounts; sterling report format (used for editing) and sterling non-report format (used for arithmetic). In COBOL E, neither a sterling report format nor a sterling non-report format can contain more than 15 digits in the pound and pence decimal-fraction fields combined.

In the formats that follow, n stands for a positive integer other than zero. This integer enclosed in parentheses and following the symbols 9, B, etc. indicates the number of consecutive occurrences of the preceding symbol. For example, 9(6) and 999999 are equivalent.

The characters 6 7 8 9 C D * , / B Z V . : s d CR DB + - are the PICTURE characters used to describe sterling items. (The character is the sterling equivalent of the character $.)

Note: The lower case letters "s" and "d" are represented by an 11-0-2 punch and a 12-0-4 punch, respectively.

STERLING NON-REPORT

The format of the PICTURE clause for a sterling non-report data item is:

```
PICTURE IS 9[(n)]D[8]8D{6[6]}
                       {7[7]}

  [[V]9[(n)]] USAGE IS DISPLAY-ST
```

Note: For a sterling non-report picture to be valid, it must contain a pound field, a shilling field, and a pence field.

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.

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 2-column field. 8 indicates B.S.I. single-column shilling representation.

2. The character D indicates the position of an assumed shilling separator.

The representation for pence is

$$\left\{\begin{matrix}6[6]\\7[7]\end{matrix}\right\}[[V]9[(n)]]$$

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 2-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 2-column representation of pence. Consequently, 66 and 77 serve the same purpose and are interchangeable.

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.

4. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.

Example: Assume that a sterling currency data item used in arithmetic expressions is to be represented in IBM shillings and IBM pence, and that this data item will never exceed £ 99/19s/11d. Its picture should be:

```
PICTURE 9(2)D88D6 DISPLAY-ST.
```

Note: The VALUE clause may not be specified for a sterling non-report item.

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 2-column shilling representation, the low-order pence digit in 2-column pence representation, or the least significant decimal position of pence.

The following are examples of sterling currency non-report data items showing sign representation in each of the allowable positions:

```
PICTURE S99D88D6V9(3) DISPLAY-ST

PICTURE 9S9D88D6V9(3) DISPLAY-ST

PICTURE 9(2)DS88D6V9(3) DISPLAY-ST

PICTURE 9(2)D88D6S6V9(3) DISPLAY-ST

PICTURE 9(2)D88D6V99S9 DISPLAY-ST
```

STERLING REPORT

The format for the PICTURE clause for a sterling currency report data item is show in Figure 20.

The sterling currency report data item is composed of four portions: pounds, shillings, pence, and pence decimal fractions.

The delimiter characters C and D primarily serve to indicate the end of the pounds and shillings portions of the picture. In addition, they serve to indicate the type of editing to be applied to separator characters to the right of the low-order digit (of the pounds and shillings integer portions of the item).

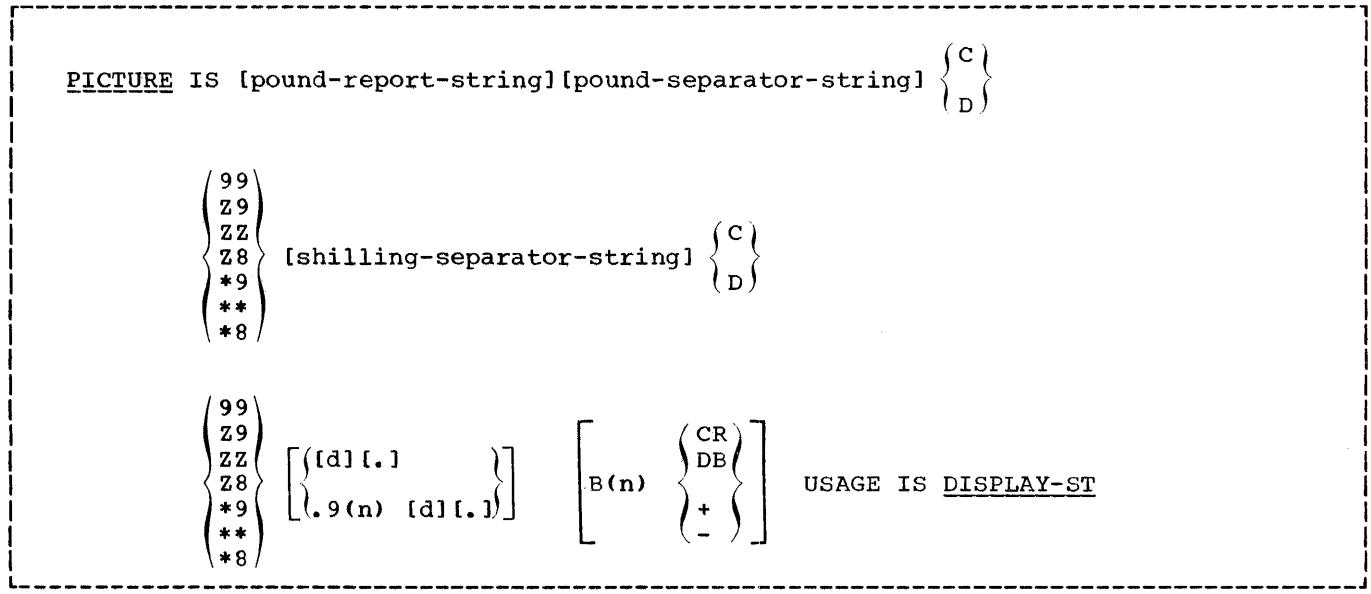


Figure 20. Format of Sterling Report PICTURE Clause

The delimiter character D indicates that separator character(s) to the right of the low-order digit position (of the field delimited) are always to appear; that is, no editing is performed on the separator character(s).

The delimiter character C indicates that if the low-order digit position (of the field delimited) is represented by other than the edit character 9, then editing continues through the separator character(s). For example, a value of zero moved to a sterling report item represented by the picture

```
**/CZ9s/D99d
```

would result in

```
***b0s/00d
```

whereas if the picture were

```
**/DZ9s/D99d
```

the result would be

```
**/b0s/00d.
```

The delimiter C is equivalent to D when the low-order digit position is represented by a 9. That is, the following two pictures are equivalent:

```
ZZ9/CZ9/C99
ZZ9/DZ9/D99
```

The delimiters used for the pounds and shillings portion of the picture need not be the same.

Editing applications are shown in Table 20.

Note: Although the pound-report-string and the pound-separator-string are optional, a delimiter character (either C or D) must be present; thus, when programming for shillings and pence only, the PICTURE clause must begin PICTURE [IS] C (or D)...

The separator characters (those characters required to distinguish one portion of the data item from the next) that may be used in a sterling currency report picture are B : / £ s (for shillings) d (for pence) and a period. Any of these characters may be used in any position in which a separator character is permitted.

The pound-report-string is subject to the same rules as a 'normal' report picture

1. The allowable characters are £ 9 Z * + - 0 (zero) B and a comma.
2. The character £ is the sterling equivalent of $.
3. Termination is explicitly specified by the character C or D.
4. Editing of separator characters to the right of the low order digit varies (depending on the use of C or D as a delimiter).

Table 20. Sterling Currency Editing Applications

Picture	Numeric Value (in pence)	Sterling Equivalent £ s d	Result
£££/D99s/D99d	3068	12 15 08	£12/15s/08d
£££/D99s/D99d	0668	2 15 08	b£2/15s/08d
£££/D99s/D99d	0188	0 15 08	bbb/15s/08d
£££/C99s/D99d	0188	0 15 08	bbbb15s/08d
ZZZ/DZZs/DZZd	0000	0 00 00	bbb/bbs/bbd
ZZZ/CZZs/DZZd	0000	0 00 00	bbbbbbs/bbd
ZZZ/CZZs/CZZd	0000	0 00 00	bbbbbbbbbbd
***/C**D/C**.99d	1040.12	4 06 08.12	**4/*6s/*8.12d
***/C**s/C**.99d	080.12	0 06 08.12	*****6s/*8.12d
***/D**s/D**.99d	00001.23	0 00 01.23	***/**s/*1.23d
£££/D*9s/D**.99d	00961.23	4 00 01.23	b£4/*0s/*1.23d
£**/D*9s/D**.99d	00961.23	4 00 01.23	£*4/*0s/*1.23d
£**/D*9s/D**.99d	00001.23	0 00 01.23	£**/*0s/*1.23d

A sterling report PICTURE may have a BLANK WHEN ZERO clause associated with it specifying that the item described is filled with spaces whenever the value of the item is zero. The VALUE clause may be specified for a sterling report item if the literal is alphanumeric.

The representation of digits positions in both the shillings and pence integers portion of the picture is identical. The edit character 8 is treated as a 9, but if the digits to the left of the edit character 8 are zeros, the 8 is treated as the character that precedes it (either Z or *).

PROCEDURE DIVISION CONSIDERATIONS

[F ONLY] MOVE, DISPLAY, ACCEPT, EXAMINE, and TRANSFORM statements, arithmetic statements, and relation tests may be written containing data-names that specify sterling items.

In COBOL E, the MOVE, ADD, and SUBTRACT statements may be written containing data-names described as sterling items.

Sterling items are not considered as integral items and should not be used where an integer is required.

INTERNATIONAL CONSIDERATIONS

1. [F ONLY] The functions of the period and the comma may be exchanged in the PICTURE character-string and in numeric literals by writing the clause DECIMAL-POINT IS COMMA in the Special-Names paragraph of the Configuration Section of the Environment Division.
2. The PICTURE of report items may terminate with the currency symbol in cases where the graphic $ is supplanted by a particular national currency symbol.

The following debugging statements may appear anywhere in an System/360 COBOL program or in a compile-time debugging packet.

For the TRACE and EXHIBIT statements, the output is written on the system logical output device (SYSOUT). In COBOL E, the logical record size must be specified on the associated SYSOUT DD statement. In COBOL F, a maximum logical record size of 120 characters is assumed. This assumed size is overridden if a logical record size is specified on the associated SYSOUT DD statement. (See the publications IBM System/360 Operating System, COBOL (E) Programmer's Guide, Form GC24-5029 and IBM System/360 Operating System, COBOL (F) Programmer's Guide, Form GC28-6380.)

TRACE

The format of the TRACE statement is:

```
{READY}
{RESET} TRACE
```

After a READY TRACE statement is executed, a message is written each time execution of a paragraph or section begins.

The execution of a RESET TRACE statement terminates the functions of a previous READY TRACE statement.

EXHIBIT

The format of the EXHIBIT statement is:

```
                {NAMED        }
EXHIBIT         {CHANGED NAMED}
                {CHANGED      }

{data-name...          }
{literal[1]...         }
{alphanumeric-literal  }
```

[1]Implemented for COBOL F only

The execution of an EXHIBIT NAMED statement causes a formatted display of the data-names (or non-numeric literals) listed in the statement. The format of the output for each data-name listed in the NAMED or CHANGED NAMED form of an EXHIBIT statement is:

```
original data-name (including qualifiers, if written)
blank
equal sign
blank
value of data-name
blank
```

Literals listed in the statement are preceded by a blank, when displayed.

In COBOL E, if the sum of the operands of an EXHIBIT statement, including the inserted blanks, exceeds the maximum logical record size for the system logical output device (SYSOUT), the action taken is as described for the DISPLAY statement. In COBOL F, the display of the operands is continued as described for the DISPLAY statement.

For COBOL E, 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 (data-name being the same in both cases), 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.

If the list of operands in an EXHIBIT CHANGED statement includes literals, they are printed as remarks and are preceded by a blank.

For COBOL E, only one data-name may be listed in an EXHIBIT CHANGED statement.

|F ONLY|

If there are two or more data-names as operands of EXHIBIT CHANGED, and some but not all are changed from the previous execution of the statement, only the changed values are displayed. The positions reserved for a given operand in the data to be displayed are blank when the value of the operand is not changed. The programmer can thus create a fixed columnar format for the data to be displayed by use of the EXHIBIT CHANGED. The data-name operands of the EXHIBIT CHANGED statement must be less than 256 bytes in length.

If all operands in an EXHIBIT CHANGED statement have not changed value from the previous execution of the statement, a blank line(s) will be printed. However, for EXHIBIT CHANGED NAMED, no blank line(s) will be printed.

Variable length data items are not permitted as operands when the CHANGED option is used.

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.

ON (Count-Conditional Statement)

The ON statement is a conditional statement. It specifies when the statements it contains are to be executed.

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
```

ELSE (or OTHERWISE) NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence. All integers contained in the statement must be positive and less than $2^{31}-1$. When using the ON statement, integral numeric literals cannot exceed $2^{31}-1$, or 2,147,483,647.

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 m is any positive integer, if the value of the counter is equal to integer-1 + (m * integer-2), but is less than integer-3 (if specified), then 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, but integer-3 is 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. In COBOL E, compile-time debugging packets are grouped together and are placed immediately preceding the first card in the source program. See the publication IBM System/360 Operating System: COBOL E Programmer's Guide, Form GC24-5029, for the proper DEBUG JCL construction. In COBOL F, compile-time debugging packets are grouped together and are placed immediately following the last card of 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 [,][TRY]1
```

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 may not be a paragraph-name within the DEBUG packet itself.

Note: Location can start anywhere within Margin A.

[F ONLY] If the TRY option is used, immediate loading and execution of the object program will be allowed even if an error appears within a debug packet. If TRY is not specified, a significant source program error, when encountered in the procedural text of a debug packet, will prevent loading and execution of the object program. The debug packet may refer to paragraph and section-names brought into the program as the result of one or more INCLUDE statements.

A debug packet may consist of any procedural statements conforming to the requirements of System/360 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. However, before an explicit branch to a debug packet is executed, the path of program flow must reach the location specified in the DEBUG statement.

1Implemented for COBOL F only.

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

BEFORE
BLANK
BLOCK
BY

CALL
CF
CH
CHANGED
CHARACTERS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORRESPONDING

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DIRECT
DIRECT-ACCESS
DISPLAY
DISPLAY-ST
DIVIDE
DIVISION

ELSE
END
ENTER
ENTRY
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXHIBIT
EXIT

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILLER
FINAL
FIRST
FOOTING
FOR
FORM-OVERFLOW
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD

IBM-360
ID
IDENTIFICATION
IF
IN
INCLUDE
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS

JUSTIFIED

KEY

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

MODE
MOVE
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOT
NOTE
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OMITTED
ON
OPEN
OR
ORGANIZATION
OTHERWISE
OUTPUT

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PICTURE
PLUS
POSITIVE
PRINT-SWITCH
PROCEDURE
PROCEED
PROCESS
PROGRAM-ID

QUOTE
QUOTES

RANDOM
RD
READ
READY
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMARKS
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTRICTED
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SELECT
SENTENCE
SEQUENTIAL
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STOP
SUBTRACT
SUM
SYMBOLIC
SYSIN
SYSOUT
SYSPUNCH

TALLY
TALLYING
TERMINATE
THAN
THEN
THRU
TIMES
TO
TRACE
TRACK-AREA
TRACKS
TRANSFORM
TRY
TYPE

UNIT
UNIT-RECORD
UNITS
UNTIL
UPON
USAGE
USE
USING
UTILITY

VALUE
VARYING

WHEN
WITH
WORKING-STORAGE
WRITE
WRITE-ONLY
ZERO
ZEROES
ZEROS

In IBM System/360, storage is organized into bytes. Four bytes comprise a word of storage. Two bytes comprise a halfword; eight bytes comprise a doubleword. Certain types of processing operations require that data be aligned on a certain type of boundary -- halfword, fullword, or doubleword. In order to ensure 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.

INTRA-RECORD SLACK BYTES

An item described with the COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 option of the USAGE clause must be aligned on the proper type of word boundary. When necessary, slack bytes are added to ensure this alignment.

These slack bytes are added on the assumption that every level 01 data item begins on a double-word boundary. In the following cases, responsibility for arranging such alignment rests with the user:

1. If the item is the argument of a CALL statement, and corresponds to a dataname with a level 01 description in the linkage section of the subprogram.

2. If the record-name of the item is associated with a file containing blocked records.

The user can insure double-word boundary alignment in one of two ways:

1. By moving the item to a level 01 description in the Working-Storage Section.

2. By the addition of inter-record slack bytes to force proper alignment of succeeding records in the block (see below, "Inter-record Slack Bytes").

The following method is used by the compiler to determine whether intra-record slack bytes are required:

The total number of bytes contained in all elementary data items preceding the computational item under discussion are added together, including any slack bytes previously added. This sum is divided by m, where:

```
m = 2 for COMPUTATIONAL items of 4-
    digit length or less
m = 4 for COMPUTATIONAL items of 5-
    digit length or more
m = 4 for COMPUTATIONAL-1 items
m = 8 for COMPUTATIONAL-2 items
```

If the remainder (r) of this division is equal to zero, no slack bytes are required. If the remainder is not equal to zero, the number of slack bytes that must be added is m - r.

These slack bytes are added following the elementary data item immediately preceding the computational item under discussion. They are defined with a level-number equal to that of the data item following the elementary item. For example:

```
01 A.
  02 B    PICTURE IS X(5).
  02 C.
    03 D    PICTURE IS X(2).
    03 SLACK-BYTES    PICTURE IS X.
    03  E USAGE COMPUTATIONAL
        PICTURE IS S9(6).

01 A.
  02 V    PICTURE IS X(5).
  02 C    PICTURE IS X(2).
  02  SLACK-BYTES    PICTURE IS X.
  02 D.
    03 E  USAGE COMPUTATIONAL
       PICTURE IS S9(6).
```

Slack bytes may also be added by the compiler when a group item is defined with an OCCURS clause and contains within it a data item with USAGE defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2.

To determine whether slack bytes are required, calculate the size of the group including all the necessary intra-record slack bytes. Divide this sum by the largest m required by any elementary item within the group.

If r is equal to zero, no slack bytes are required. If r is not equal to zero, m - r slack bytes must be added.

The slack bytes are added at the end of the group item containing the OCCURS clause, with a level-number equal to one

(1) greater than the level-number of the group item. For example:

```
01 A.
  02 B     PICTURE IS X(7).
  02 C     OCCURS 10 TIMES.
    03 D.
      04 E     PICTURE IS X.
      04 F     USAGE COMPUTATIONAL-2.
    03 G.
      04 H     PICTURE IS XX.
    03  SLACK-BYTES     PICTURE IS X(5).
  02 I     PICTURE IS X.
```

Where data items defined as COMPUTATIONAL, COMPUTATIONAL-1 or COMPUTATIONAL-2 follow a field containing an OCCURS clause with the DEPENDING ON option, slack bytes are added on the basis of the field occurring the maximum number of times. If the length of this field is divisible by the m required for the computational data, all values of the data-name specified in the OCCURS clause are valid. If it is not evenly divisible, the only values of the data-name in the OCCURS clause that will give proper alignment of the computational fields are those where the length of the data item times the number of occurrences plus the slack bytes that have been calculated based on the maximum number of occurrences is evenly divisible by m. For example:

```
01 A.
  02 B     PICTURE 9(2).
  02 C     PICTURE X OCCURS 99 TIMES DEPEND-
           ING ON B.
  02  SLACK-BYTE     PICTURE X.
  02 D     USAGE COMPUTATIONAL PICTURE IS
           S9(2).
```

In this example when references to D are required, B is restricted to odd values only.

```
01 A.
  02 B     PICTURE 9(3).
  02 C     PICTURE X (2) OCCURS 99 TIMES
           DEPENDING ON B.
  02  SLACK-BYTE     PICTURE X.
  02 D     USAGE COMPUTATIONAL     PICTURE IS
           S9(2).
```

In this example, all values of B give proper references to D.

INTER-RECORD SLACK BYTES

If a file contains blocked logical records that are to be processed in a buffer, the user must add any inter-record slack bytes needed for proper alignment if any of the records contain entries defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2.

When intra-record slack bytes are added in order to assure proper alignment of a data item described with the COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 option of the USAGE clause, it is assumed that every level 01 data item starts on a double-word boundary. This alignment is automatic only in the case of level 01 data items in the Working-Storage Section, and of input/output buffers (excluding any control bytes required by Data Management).

The following method should be used to determine whether any inter-record slack bytes are required:

Add the lengths of all the elementary data items in the record, including all intra-record slack bytes. If RECORDING MODE IS V has been specified for the file, it is necessary to add four bytes for the count field. Divide the total by the highest value of m for any one of the elementary items in the record.

If r (the remainder) is equal to zero, no inter-record slack bytes are required. If r is not equal to zero, m - r slack bytes are required. These slack bytes may be specified by writing an 02 FILLER at the end of the record.

If format U records are being read backwards, double-word boundary alignment of the I-O buffer will be obtained only if the lengths of the logical records are divisible by eight.

SUMMARY OF DATA DIVISION REQUIREMENTS

┌-------┐
|F ONLY|
└-------┘ REPORT SECTION: The compiler adds all necessary slack bytes to insure proper alignment of computational fields.

WORKING-STORAGE SECTION: The compiler adds all necessary slack bytes to insure proper alignment of computational fields.

┌----┐
|EXT|
└----┘ LINKAGE SECTION: All level 01 items are assumed to begin on double-word boundaries. It is the responsibility of the user to insure that the arguments in a CALL statement are properly aligned.

FILE SECTION:

1. In the case of input files, it is the responsibility of the user to make sure that the logical records contain any necessary intra-record slack bytes.

If an input file contains blocked records, and processing is done in a buffer, the required inter-record slack bytes must have been inserted when the file was created.

2. In the case of output files, the compiler adds the necessary intra-record slack bytes to the logical records.

   The user is responsible for inserting necessary inter-record slack bytes to an output file if they will be required when it functions as an input file.

APPENDIX C: INTERMEDIATE RESULTS

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 such cases, the compiler treats the 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       yielding ir1
ADD A            TO ir1     yielding ir2
DIVIDE E         INTO D     yielding ir3
SUBTRACT ir3     FROM ir2   yielding ir4
** F             BY G       yielding ir5
ADD ir4          TO ir5     yielding Y
```

The conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results is discussed in this appendix.

When a statement applies to only one of the compilers, it will be so noted. If no restriction is noted, statements apply to both COBOL E and COBOL F.

The following abbreviations will be used:

i--number of integer places carried for an intermediate result.

d--number of decimal places carried for an intermediate result.

dmax--maximum number of decimal places defined for any operand (except for exponents or divisors) in a particular statement.

op1--first operand in a generated arithmetic statement.

op2--second operand in a generated arithmetic statement.

d1,d2--number of decimal places defined for op1 or op2, respectively.
umber 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--E COMPILER

In COBOL E, the number of integer and decimal places contained in an ir is calculated as shown in Table 21.

INTERMEDIATE RESULTS--F COMPILER

In COBOL F, the number of integer places contained in an ir is calculated as follows:

The compiler first determines the maximum value that the ir can contain by performing the arithmetic statement in which the ir occurs.

1. If an operand in this statement is a data-name, the value used for the data-name is equal to the numeric value of the PICTURE for the data-name (e.g., PICTURE 9V99 has the value 9.99).
2. If an operand is a literal, the actual value of the literal is used.
3. If an operand is an intermediate result, the value determined for the intermediate result in a previous arithmetic operation is used.
4. If the operation is division:
   a. If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE for the data-name (e.g., PICTURE 9V99 has the value 0.01).

b. If op2 is an intermediate result, the intermediate result is treated as if it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.

Operation	Decimal Places
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	d1 - d2 or dmax, whichever is greater
**	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

In COBOL F, the number of decimal places contained in an ir is calculated as follows:

Compiler Treatment of Intermediate Results

Table 22 indicates the action of the compiler when handling intermediate results.

Table 21. Calculating Intermediate Results Using the E Compiler.

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 1 in all + or - operations if either field is binary or packed.

Table 22. Compiler Action on Intermediate Result

Compiler	Value of i + d	Value of d	Value of i + dmax	Action taken
F	<30	Any value	Any value	i integer and d decimal places are carried for ir
	=30			
	>30	<dmax	Any value	30 - d integer and d decimal places are carried for ir
		=dmax		
		>dmax	<30	i integer and 30 - i decimal places are carried for ir
			=30	
			>30	30 - dmax integer and dmax decimal places are carried for ir
E	<30	Any value	Not applicable	i integer and d decimal places are carried for ir. (If operation is / or **, i + d never exceeds 30).
	=30			
	>30	Any value	Not applicable	30 - df integer and df decimal places are carried

Note: If ROUNDED is specified, the value of df is df + 1.

This appendix contains two sample COBOL rograms. Figure 21 is a calling program; he other, Figure 22, is a subprogram which is linked by the calling program. The linkage subprogram illustrated need not be a COBOL program.

ALLING PROGRAM

```
1       8   12
001001 IDENTIFICATION DIVISION.
001002 PROGRAM-ID. 'CALLPROG'.
001003 REMARKS. EXAMPLE OF A CALLING PROGRAM.
 .  .       .
 .  .       .
 .  .       .
001008 DATA DIVISION.
 .  .       .
 .  .       .
001014 WORKING-STORAGE SECTION.
001015 01 RECORD1.
001016   02 JONES-J.
001017      03 SALARY PICTURE IS 9(5)V99.
001018      03 RATE PICTURE IS 9V99.
001019      03 HOURS PICTURE IS 99V9.
 .  .       .
001021 PROCEDURE DIVISION.
 .  .       .
 .  .       .
001025      ENTER LINKAGE.
001026      CALL 'PAYMASTR' USING JONES-J.
001027      ENTER COBOL.
 .  .       .
 .  .       .
 .  .       .
```

Figure 21. Example of a Calling Program

CALLED PROGRAM

```
1       8   12
002001 IDENTIFICATION DIVISION.
002002 PROGRAM-ID. 'SUBPROG'.
 .  .       .
 .  .       .
 .  .       .
002005 DATA DIVISION.
 .  .       .
 .  .       .
 .  .       .
 .  .       .
002012 LINKAGE SECTION.
 .  .       .
002015 01 PAYOFF.
002016   02 PAY PICTURE IS 9(5)V99.
002017   02 RATEX PICTURE IS 9V99.
002018   02 HOURS PICTURE IS 99V9.
 .  .  .
 .  .  .
002025 PROCEDURE DIVISION.
 .  .       .
 .  .       .
 .  .       .
002040     ENTER LINKAGE.
002041     ENTRY 'PAYMASTR' USING PAYOFF.
002042     ENTER COBOL.
 .  .       .
 .  .       .
002050     ENTER LINKAGE.
002051     RETURN.
002052     ENTER COBOL.
```

Figure 22. Example of a Called Program

COBOL F ONLY FEATURES

The following COBOL 1965 features, described in this publication, are implemented only for COBOL F:

1. The CORRESPONDING option of the ADD, SUBTRACT, and MOVE statements
2. The Report Writer feature (part of this feature is an extension to COBOL, see below)
3. The Sort feature
4. Implied subjects and relational operators in compound conditions
5. The SPECIAL-NAMES paragraph
6. The mnemonic-name option of the DISPLAY statement
7. The mnemonic-name option of the ACCEPT statement
8. The REEL/UNIT option in conjunction with the NO REWIND/LOCK option of the CLOSE statement (both options may be used in a statement)

(See "COBOL Extensions" below for other COBOL F only features)

COBOL EXTENSIONS

The following features, described in this publication, are IBM extensions to COBOL 1965 for IBM System/360 Operating System. Some are extensions to COBOL features, while others are completely new for COBOL E and COBOL F.

1. The ORGANIZATION clause
2. Internal and external floating-point items and floating-point literals
3. The MOVE 1 TO PRINT-SWITCH statement in the Report Writer feature (this feature is implemented for COBOL F only)
4. The Linkage Section of the Data Division
5. The overflow-name test-condition
6. The REWRITE statement
7. The TRANSFORM statement
8. The debugging language (the TRY option of the debugging packet is implemented for COBOL F only; the EXHIBIT statement is available to COBOL E in restricted form only)
9. The Sterling Currency Feature (this feature is available to COBOL E in restricted form only)
10. The RECORD KEY clause
11. The TRACK-AREA clause (this feature is available to COBOL E in restricted form only)
12. The Extended Source Program Library Facility (this feature is implemented for COBOL F only)

(Where more than one page reference is given, the major reference is given first.)

Special characters
. (decimal point) 50
in sterling PICTURE clause 131-134
< (less than) 63
+ (plus)
in PICTURE clause 52
in sterling PICTURE clause
$ (dollar sign) 52
£ (pound symbol) 131-134
* (check protection) 51
in sterling PICTURE clause 131-134
- (Minus)
in PICTURE clause 52
/ (slash)
in sterling PICTURE clause 131-134
, (comma) 9-10
in PICTURE clause 51
in sterling PICTURE clause 131-134
> (greater than) 63
= (equal sign) 63

ACCEPT 78-79
sterling items 134
ACCESS clause 17-19,27,72-75
Access methods
(see Sequential access, Direct access)
ACTUAL KEY clause 71-74,16,19,28,55
not permitted under QSAM 17
ADD 87,145
ADD CORRESPONDING 87,150
Addressing
relative record 16
relative track 16
AFTER ADVANCING 78
(see also WRITE, Line spacing)
Algebraic comparisons 63
ALL 81-82
ALL 'figurative constant' 36
Allocation of space 15
Alpha-form option of PICTURE clause 50
ALPHABETIC 64
Alphabetic items 38
format 45
Alphanumeric items 38
format 45
ALTER 89,90,137
(see also TRANSFORM)
An-form option of PICTURE clause 50
AND 67
APPLY clause 30-31
(see also APPLY FORM-OVERFLOW option, APPLY WRITE-ONLY option, APPLY RESTRICTED SEARCH option)
APPLY FORM-OVERFLOW option 30
APPLY RESTRICTED SEARCH option 30,20
APPLY WRITE-ONLY option 30-31,43
Arithmetic expressions 9,67
Arithmetic operators 68
rules regarding 10
Arithmetic statements 85
Arithmetic verbs 69
(see also COMPUTE, ADD, SUBTRACT, MULTIPLY, DIVIDE)
ASCENDING 122
ASSIGN clause 26,27,123
AT END 59,61,71-2,124
AUTHOR 23
Automatic end-of-volume 76

B
in PICTURE clause 51
in sterling PICTURE clause 131-134
BASIS 130
BDAM 16,20
with relative record addressing 21
with relative track addressing 20
Binary items 36
format 46
BISAM 16,18,19
BLANK WHEN ZERO 51-53,134
BLOCK CONTAINS clause 43-44
Blocked records 17
Blocks, definition of 40
Boundary alignment 94
(see also Slack bytes)
BSAM 16,19,20
with relative record addressing 20
with relative track addressing 19
BSI (British Standards Institution) 131
Buffer, definition of 40
BY 90-91
Byte alignment (see slack bytes)

C (character in sterling PICTURE clause) 131-134
CALL 93,94,123,141
CALLED program 150
Calling Program 149
Capacity records (see Dummy records)
Carriage control (see Line spacing)
CF (see CONTROL FOOTING)
CH (see CONTROL HEADING)
CHARACTERS option 43-44
Check protection 51
Checkpoint records (see RERUN clause)
Checkpoint/Restart (see RERUN clause)
Class tests 62-64
Clauses
(see individidual clause name, i.e., LABEL RECORDS clause, LINE clause)
CLOSE 69-72,75-76
COBOL character set 9
COBOL program sheet, use of 11
COBOL programs, examples of 149-150
COBOL reserved words 139
COBOL subprograms (see subprograms)

CODE clause 99
Column clause 103
Comma, rules regarding 9-10
Comparisons
  algebraic 63
  of non-numeric items 64
  of numeric items 63
Compile-time debugging packets 136-139
Compiler-directing statements 59,68-69
  (see also ENTER, EXIT, NOTE)
Compound conditions 66
  relational operators in 150
  (see also AND, OR, NOT)
COMPUTATIONAL 49,78,94,141-142
  rounding of 85
  (see also Binary items)
COMPUTATIONAL-1 49,78,94,141-142
  rounding of 86
COMPUTATIONAL-2 49,78,94,141-142
  rounding of 86
COMPUTATIONAL-3 78
  rounding of 85
COMPUTE 86,145
Condition-name tests 37,62,66
Condition-names 10,34
  subscripting of 37
Conditional statements 59-60
  evaluation of 60
Configuration Section, format of 25
CONSOLE 25-26,77-79
Continuation indicator 11
Continuation of non-numeric literals 11-12
Control breaks 99
CONTROL clause 99
CONTROL FOOTING 97-100,102
CONTROL HEADING 98,100,102
Controls 99,100,105
  hierarchy of 100
Conversion 80
COPY 129
CORRESPONDING option 85,87,150
Count control field 40,42-44
Count-condition 59,136-137
CR (credit symbol) 51
  in sterling PICTURE clause 131-134

D (shilling indicator) 131-134
Data control blocks (DCB's) 15
Data Division 33
Data items, maximum length of 37
Data maninipulation verbs 69
  (see also MOVE, EXAMINE, TRANSFORM)
Data organization 15
  direct 16
  (see also BDAM, BSAM)
  indexed 15
  (see also QISAM, BISAM)
  relative 16
  (see also BSAM, BDAM)
  standard sequential 15
  (see also QSAM)
DATA RECORDS clause 44
Data set label 15
Data sets
  (see Files)
Data-names 10
  definition 34
  maximum size in arithmetic statements 85
  qualification of 11,35
DATE-COMPILED 23
DATE-WRITTEN 23
DB (debit symbol) 51
  in sterling PICTURE clause 131-134
DD card 15,42,44,71,75-7,135
DE (see DETAIL)
Debugging language 135-137,150
Debugging packets 130,135
Decimal point alignment 62,80,85
DECIMAL-POINT IS COMMA clause 25-26,50
  international considerations 134
Declarative sections 68-69,106
  format 68
Declaratives 68,106
DELETE 11,130
Delete codes 36
Deletion of records 17
DEPENDING ON option 90,192
DECENDING 122
DETAIL 97,102
Detail line
  (see DETAIL)
Device classes 27
Device dependence 15-17
Direct access 16
Direct data organization 16
Direct file processing techniques 19
  (see also BSAM, BDAM)
DISP parameter 76-77
DISPCK option 78
Displaced records 18
Display items, rounding of 85
DISPLAY option 49
DISPLAY statement 77-78,150
  sterling items 134
DISPLAY-ST 131-134
DIVIDE 88
Dummy records 18-20,36

E (symbol in floating-point literal) 35-36,39-40,53
EBCDIC 9,95
Editing 79-80
  sterling items 133
8 in sterling PICTURE clause 131-134
88 (see Condition-names)
Elementary items 34,37-38
  format 45
ELSE 60-61,136
END DECLARATIVES 68
End-of-file 71-72
End-of-page 97
End-of-volume 72-73
  automatic 76
  forced 76
ENTER 93-94,59
ENTRY statement 93-94,33,57
Environment Division 25-31
  sample coding 31
Equal sign 63
EQUAL TO 63
Error checking (see Error-processing)
Error-processing 68-69,72,74

valuation of conditional statements 60
vent-conditions 60
XAMINE 81-82
sterling items 134
XHIBIT 135,151
XHIBIT CHANGED 135-136
XHIBIT CHANGED NAMED 136
XHIBIT NAMED 135
XIT 59,91,94-95,125
xponentiation 68
xponents 35-36,53
xpressions, arithmetic 9
xtended Binary Coded Decimal Interchange Code (see EBCDIC)
xtended Source Program Library Facility 131,151
xtensions to COBOL 151
xternal-decimal items 38
format 46
xternal-names 10

Only features 151
record format (see Fixed format records)
D 26,39,42,44,71
format 41
igurative constants 85
definition 36
(see also ZERO, SPACE, QUOTE, HIGH-VALUE, LOW-VALUE)
ile description (see FD)
ile processing 16
(see also QSAM, QISAM, BSAM, BISAM, BDAM)
File Section 33,40
FILE-CONTROL 26,119
FILE-LIMIT clause 29
File-names 10
qualification 11
Files 5
creation 15
definition 40
I-O 17-18,20-21
input 17-18,20-21,69
multi-volume 75
output 17-20
FILLER 34-35,81
FINAL 99-100
FIRST 81-83
FIRST DETAIL 100
Fixed format records 17,19-20,40,42-43,120
Fixed length records
(see Fixed format records)
Fixed-point items 38
(see also Binary items, External-decimal items, Internal-decimal items)
Floating strings 51-52
Floating-point items 39
(see also External floating-point items, Internal floating-point items)
Floating-point literals 35
maximum value 36
FOOTING 100
Forced-end-of-volume 76
FORM-OVERFLOW option 17,30
Format F (see Fixed format records)
Format notation 12
Format U (see Unspecified format records)
Format V (see Variable length records)
FP-form option of PICTURE clause 52
FROM option 73,78,83,85,88,90-91

GENERATE 97,105
GIVING option 85,87-88,119-122
GO TO 69,89,91,137
GREATER THAN 63
GROUP INDICATE 103
Group items
definition 37
format 45
HEADING 100
Hierarchy of operations 68
HIGH-VALUE 17-18,20,36,73

I-O files 17-18,20-21
I-O option 17,71
I-O-CONTROL Paragraph 21
IBM pence 131
IBM shillings 131
Identification Division 23
IF statement 23-59-60,195
format 60
nested 61
Imperative statements 59
Implicit redefinition 41
Implied operators 67
Implied subjects 67,150
IN (see Qualification)
INCLUDE 130,137
Indexed data organization 15
Indexed file processing techniques 17
(see also QISAM, BISAM)
INITIATE 97,105
INPUT option 71
Input files 17-18,20-21,43-44,69
INPUT PROCEDURE 121-124
control of 124
Input-Output Section 26
Input/Output verbs
(see OPEN, READ, WRITE, REWRITE, CLOSE, ACCEPT, DISPLAY)
INSERT 11,130
INSTALLATION 23
Inter-record slack bytes
(see Slack bytes)
Intermediate results 85,145-147
Internal decimal items 38,46
Internal floating-point items 40,150
format 47
Internal representation of numeric items (chart) 39
International considerations 134
INTO 71-72
Intra-record slack bytes (see Slack bytes)
INVALID KEY 59,61,71-74

JUSTIFIED RIGHT Clause 56,80

Key words 12
Keys
actual 16,71-75
record 18,28,72-73,151
symbolic 16,18,21,71-75

LABEL RECORDS clause 42,45
Labels
non-standard 42
omitted 42
standard 42
LAST DETAIL 100
LEADING 81-82
LESS THAN 63
Level numbers 12,34
Level 88
(see also Condition-names)
LINE clause 102
Line spacing 73-74
LINE-COUNTER 104-105,107
Linkage Section 33,57,94,151
Literals
definition 35
floating-point 35-36
maximum size in arithmetic statements 85
non-numeric 11-12,35
numeric 35
Logical operators 66
Long-precision format
(see Internal floating-point items)
LOW-VALUE 36

Mantissas 35-36,53
Margin restrictions 12,23,33,41,59-60,68,95
Mnemonic-names 25-26
(see also CONSOLE, SYSOUT, SYSPUNCH, SYSIN)
MOVE 79-80
sterling items 134
MOVE CORRESPONDING 80-81,151
MOVE 1 TO PRINT-SWITCH
(see Print-switch)
Multi-volume files 75
MULTIPLY 88

Names
qualification of 10-11
types of 10
NEGATIVE 64
Nested IF statements 61
NEXT GROUP 103
NEXT PAGE 103
NEXT SENTENCE 61,136
9 (digit position) 50
in sterling PICTURE clause 131-134
NO REWIND option 71,75-77,151
Non-numeric items, comparison of 64
Non-numeric literals 35-36
continuation of 11-12
Non-standard labels 42
NOT 62
NOTE 59,95
NUMERIC 64
Numeric form option of PICTURE clause 50
Numeric items, comparison of 63
Numeric literals 35

OCCURS clause 54-55,81
(see also Subscripting)
OCCURS DEPENDING ON clause 28,40,43-44,54-55,120
OF (see Qualification)
OMITTED 42
ON 59
count-conditional 136-137
OPEN 69-73,15,76
Operators
arithmetic 10,68
implied 67
logical 66
relational 62,67
unary 10,68
Optional words, definition of 12
Options
CORRESPONDING 85
DISPCK 78
FORM-OVERFLOW 17
I-O 17
NO REWIND 71
REEL 70
REVERSED 71,15,69,76
TIMES 90
USE AFTER STANDARD ERROR 6.
USE BEFORE REPORTING 68-69,106
WITH LOCK 75-76
OR 67
ORGANIZATION clause 15-21,27,72-75,151
OTHERWISE (see ELSE)
OUTPUT 71
Output files 15-21,43
OUTPUT PROCEDURE 121-124
control of 124
Overflow areas 19
Overflow tests 62,66
Overflow-name test 151
Overflow-names 10
Overprinting 78

P (assumed numeric digit position) 50
Packed decimal format (see COMPUTATIONAL-3)
Page break 102
PAGE FOOTING 102,97
PAGE HEADING 102,97
PAGE LIMIT clause 100-101
PAGE-COUNTER 104-105
Paragraph-names 60,10
definition 60
qualified 60
Parentheses, rules regarding 9
Pence(British currency) 131-134
PERFORM 90-91,69,95,137,145
Period 9-10
PH (see PAGE HEADING)
PICTURE clause 38-39,50,52
format 49
maximum size 53
sterling representation 131-134
PLUS 103
Pocket selection 73-74
POSITIVE 64
Pounds (British currency) 131-134
Procedure branching statements 68-69,89
use of section names in 60
(see also STOP, GO TO, ALTER, PERFORM)
Procedure-names 10
qualification 11
Procedure Division 59-95

Program identification code 11
PROGRAM-ID 23
Punctuation 9,12

QISAM 16-18,44
  different from QSAM 17
QSAM 16-17,20,43-44
  different from QISAM 17
Qualification 56-57,10-11,35,80
  hierarchy 11
  of data-names 35
  of paragraph-names 60
  of sort-keys 121
  (see also Subscripting)
Qualifiers, length in COBOL E 11
  (see also Qualification)
QUOTE 36

RD 99,45,101
READ 70-76,18,59
READY TRACE 135
RECORD CONTAINS Clause 43-44
Record deletion 17
Record description entries 33,40,57
  format 45
Record displacement 18
Record formats 17,40
RECORD KEY clause 17-18,28,55,72-73,151
Record key field 18,28
RECORDING MODE clause 40-43,120
RECORDS option 43-44
REDEFINES clause 47,35,55,81
  altering usage in 47-48
Redefinition, implicit 41
REEL option 75-77,40,70-71,151
Relation operators 62,67
  in compound conditions 151
Relation tests 62,9
Relative data organization 16
Relative file processing techniques 20-21
  (see also BDAM, BSAM)
Relative record addressing 16
  (see also BDAM, BSAM)
Relative record number 72,74
Relative track addressing 16
  (see also BDAM, BSAM)
Relative track number 72-75
RELEASE 123
REMARKS 23
REPLACING 81-83
REPORT clause 98,44-45
Report description entries
  (see RD)
REPORT FOOTING 102,72,97
Report group description entries 101
Report groups, format of 101
REPORT HEADING 102,97
Report Items 38
  format 46
Report Section 97-105,33
Report Writer coding example 108-117
Report Writer feature 96-117,151
Report-form option of PICTURE clause 50
RERUN clause 29-30
RESERVE clause 17-18,27,29
Reserved words
  list 139
  definition 10,12
RESET 104
RESET TRACE 135
RETURN 59,94,124
REVERSED Option 71,15,17,76
REWRITE 74-75,18,59,69,71,151
RF (see REPORT FOOTING)
RH (see REPORT HEADING)

S (shilling indicator) 131-134
S (sign bit) 38,50
SAME clause 29
Scientific decimal items
  (see External floating-point items)
SD 19-128,34,42
  format 120
Section headers, definition of 60
Section-names
  in procedure branching statements 60
  in SORT statement 122-123
  qualification of 11
Sections
  definition 60
  declarative 68
  (see also PERFROM statement, Environment Division, Data Division, Procedure Division)
SECURITY 23
SELECT 26,119-23
Semicolon 9-10
Sentences, definition of 60
Sequence checking 11
Sequence errors 11
Sequence numbers 11,130
Sequential access 15-19
  (see also BISAM, BSAM, QISAM, QSAM)
7 in sterling PICTURE clause 131-134
Shillings (British currency) 131-134
Short-precision format
  (see Internal floating-point items)
Sign position 50
Sign tests 62,64
6 in sterling PICTURE clause 131-134
SIZE ERROR option 86-88,38,59,61
Slack bytes 141-143,43,99,94
Sort description entry
  (see SD)
Sort feature 119-128
SORT statement 121-123
  coding examples 125,127
  format 122
Sort-file-description-names 120
Sort-keys 119-122
SORTIN 120
SORTOUT 120
SOURCE clause 104
Source Program Library Facility u29-130
SPACE 36
Space allocation 15
SPECIAL-NAMES 25,151
Standard labels 42
Standard sequential data organization 15-19
Standard sequential file processing
 (see QSAM)

Statements
   compiler-directing 59
   conditional 59
   definition 59
   imperative 59
   procedure-branching 68
Sterling Currency feature 131-134,151
Sterling items
   editing of 133
   non-report 131-132
   report 132
   sign representation 132
STOP 93
STOP RUN 93,124
Subjects, implied 67
Subprogram linkage 93-94,33
Subscripting 55-56,81
   not permitted of qualifiers 11
   of condition-names 37
   (see also OCCURS clause)
SUBTRACT 88,145
SUBTRACT CORRESPONDING 88,151
SUM clause 104
Symbol pairs 67
SYMBOLIC KEY clause 71-75,16-21 27-28,55
SYSIN 25-26,78-79
SYSOUT 25-26,77-78
SYSPUNCH 25-26,77-78

TALLY 81-82
TALLYING
   (see TALLY)
TERMINATE 105-106,72,97
Test-conditions 60-62
Tests
   class 62,64
   condition-name 62,66
   overflow 62,66,151
   relation 62,9
   sign 62
   truth 66
THEN 10
THRU Option 90,94
TIMES Option 90
TO 83,85
TRACE 135
Track addressing, relative 16
Track allocation 16
TRACK-AREA clause 29,55,151
TRANSFORM 83,85,151
   sterling items 134
Truncation 80,86
Truth table 67
Truth tests 66

TRY option 137,151
TYPE clause 101

U format records
   (see Unspecified format records)
Unary operators 10,68
Unblocked records 17
UNIT (see REEL)
Unspecified format records 40-43,17,71,102
UNTIL FIRST 81-83
UNTIL option 90-91
UPON 104,77
USAGE clause 48-49,80,82
   format 49
USE
   AFTER STANDARD ERROR 68-69
   BEFORE REPORTING 68-69,105-106
User error routines 68-69,17-21
USING lists 94
USING option 94,11-12,33,57

V (assumed decimal point) 50
V record format
   (see Variable length records)
VALUE clause 53-54,57
Variable length records 40-44,17,71,120,142
VARYING option 90
Verbs
   arithmetic 69
   compiler-directing 69
   data manipulation 69
   input/output 69
Volume, definition of 40
Volume positioning 75
Volume switching 75-77

WITH LOCK option 75-76,151
Words 9
   definition 10
   key 12
   optional 12
   reserved 10,12
   rules regarding 9
Working-Storage Section 57,33,48
WRITE 72-74,59,69,76-78.
   in sterling PICTURE clause 131-134

Z (zero suppression) 50
0 used in PICTURE clause 51
ZERO 36,64,86

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Each report-name listed in the FD entry must have an RD (Report Description) entry in the Report Section of the Data Division. (Complete details concerning the Report Writer feature are contained in the chapter "Report Writer Feature.")

The REPORT clause may be specified only for files whose organization is standard sequential.

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} [redefines-clause]
             {FILLER   }

    [PICTURE-clause]    [BLANK-clause]

    [OCCURS-clause]     [VALUE-clause]

    [JUSTIFIED RIGHT]   [USAGE-clause].
```

When this format is applied to 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 not shown in a format are forbidden. Clauses that are mandatory in the description of certain data items are written without brackets.

Group Item Format

The format of the Record Description entry for a group item is:

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

    [OCCURS-clause] [USAGE-clause].
```

Sample coding for a group item and its associated subordinate (elementary) items is:

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

Note: A group item, by definition, must have items (FIELD-B and FIELD-C in the above example) that are subordinate to it. An item is subordinate to another by having a level number that is higher than the immediately preceding item. FIELD-B and FIELD-C have 02 level numbers, whereas GROUP-ITEM has an 01 level number.

Formats for Elementary Items

An elementary item is one having no items subordinate to it.

Alphabetic Item:

The format of the Record Description entry for an alphabetic item is:

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

    [OCCURS-clause] PICTURE IS alpha-form

    [USAGE IS DISPLAY]

    [VALUE IS alphabetic-literal]

    [JUSTIFIED RIGHT].
```

Sample coding for an elementary alphabetic item is:

```
02 EMPLOYEE-NAME PICTURE A(20).
```

Alphanumeric Item:

The format of the Record Description entry for an alphanumeric item is:

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

    [OCCURS-clause] PICTURE IS an-form

    [USAGE IS DISPLAY]

    [VALUE IS alphanumeric-literal]

    [JUSTIFIED RIGHT].
```

Sample coding for elementary alphanumeric items is:

```
02 MISC-1 PICTURE X(53).
02 MISC-2 PICTURE XXXX VALUE '25 A'.
```

Report Item:

The format of the Record Description entry for a report item is:

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

[OCCURS-clause]

PICTURE IS {numeric-form  BLANK WHEN ZERO}
           {report-form [BLANK WHEN ZERO]}

[USAGE IS DISPLAY].
```

Sample coding for elementary report items is:

```
02 TOTAL PICTURE $999,999.99-.
02 SUBTOTAL PICTURE S9999V99
     BLANK WHEN ZERO.
```

External-Decimal Item:

The format of the record Description entry for an external-decimal item is:

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

    [OCCURS-clause]

    [USAGE IS DISPLAY]

    PICTURE IS numeric-form

    [VALUE IS numeric-literal].
```

Sample coding for elementary external-decimal items is:

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

02 HOURS-SCHEDULED PICTURE 99V9.

02 ACCOUNTING-CODE PICTURE 99V9 DIS-
   PLAY VALUE 25.4.
```

Internal-Decimal Item:

The format of the Record Description entry for an internal-decimal item is:

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

   [OCCURS-clause]

   PICTURE IS numeric-form

   USAGE IS COMPUTATIONAL-3.

   [VALUE IS numeric-literal].
```

Sample coding for an elementary internal-decimal item is:

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

Binary Item:

The format of the Record Description entry for a binary item is:

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

 [OCCURS-clause] PICTURE IS numeric-form

 USAGE IS COMPUTATIONAL

 [VALUE IS numeric-literal].
```

Sample coding for an elementary binary item is:

```
03 SUBSCRIPT PICTURE S999
   COMPUTATIONAL.

03 TABLE-INDEX PICTURE S99
   VALUE 01.
```

External Floating-Point Item:

The format of the Record Description entry for an external floating-point item is:

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

    [OCCURS-clause] PICTURE IS fp-form

    [USAGE IS DISPLAY].
```

Sample coding for an elementary external floating-point item is:

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

Internal Floating-Point Item:

The format of the Record Description entry for an internal floating-point item is:

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

   [OCCURS-clause]

   USAGE IS {COMPUTATIONAL-1}
            {COMPUTATIONAL-2}

   [VALUE IS floating-point-literal].
```

Sample coding for an elementary internal floating-point item is:

```
02 DEVIATION COMPUTATIONAL-1.
```

REDEFINES Clause

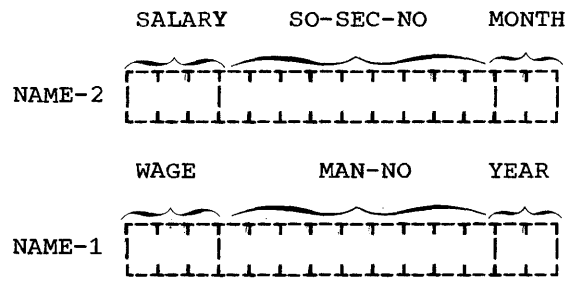
The REDEFINES clause specifies that the same area is to contain different data items, or provides an alternative grouping or description of the same data. The format of the REDEFINES clause is:

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

Data-name-2 is the name associated with the previous data description entry of equal level number. Data-name-1 is an alternate name for the same area. When written, the REDEFINES clause must be the first clause following data-name-1.

The REDEFINES clause provides the capability of using the same storage location for different types of data. Data items within an area can be redefined (in order to change their names) without changing their lengths as follows:

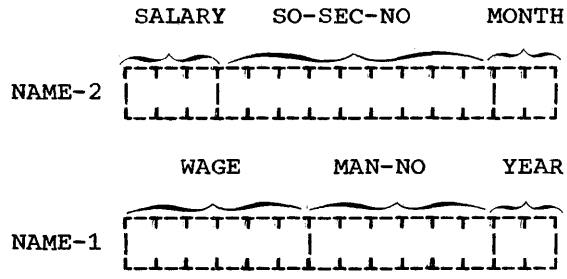
```
02  NAME-2 USAGE DISPLAY.
      03  SALARY PICTURE XXX.
      03  SO-SEC-NO PICTURE X(9).
      03  MONTH PICTURE XX.
02  NAME-1 REDEFINES NAME-2.
      03  WAGE PICTURE XXX.
      03  MAN-NO PICTURE X(9).
      03  YEAR PICTURE XX.
```



Data items can also be rearranged within an area, as follows:

```
02  NAME-2 USAGE DISPLAY.
      03  SALARY PICTURE XXX.
      03  SO-SEC-NO PICTURE X(9).
      03  MONTH PICTURE XX.

02  NAME-1 REDEFINES NAME-2 USAGE
          DISPLAY.
      03  WAGE PICTURE 999V999.
      03  MAN-NO PICTURE X(6).
      03  YEAR PICTURE XX.
```



In addition, the usage of data items within an area can be redefined.

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. It should be noted, however, that if both of the above statements are executed successively in the order specified, the value Y will overlay the value X. However, redefinition itself does not cause any data to be erased and does not supersede a previous description.

Altering the USAGE of an area through redefinition does not cause any change in existing data. Consider the example:

```
02  B  PICTURE 99 USAGE DISPLAY
       VALUE IS 8.
02  C  REDEFINES B PICTURE 99
       USAGE COMPUTATIONAL-3.
```

The bit configuration of the value 8, when used as a display item, is 1111 0000 1111 1000. Redefining B does not change its appearance in storage. Therefore, a great difference results from the two statements, ADD B TO A and ADD C TO A. In the former case, the value 8 is added to A because B is a display item. In the latter case, the bit configuration does not represent a valid internal-decimal (COMPUTATIONAL-3) number and the results of the addition are invalid.

Moving a data item to a second data item that redefines the first one (for example, MOVE B TO C when C redefines B), may produce results that are not those expected by the programmer. The reverse (MOVE B TO C when B redefines C) is also true.

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. However, the REDEFINES clause may appear in 01 levels in the Working-Storage Section. The level number of data-name-2 must be identical to that of the item containing the REDEFINES clause.

The entries giving the new description of the area must immediately follow the entries describing the area being redefined. The description of an area can mean a group item and all associated elementary items. However, in the case where more than one Record Description entry is redefining the same entry, these additional entries may intervene. For example both of the following are valid uses of the REDEFINES clause:

```
02  ARRAY-1 DISPLAY.
    03  A PICTURE X(2).
    03  B PICTURE X(2).

02  ARRAY-2 REDEFINES ARRAY-1
    USAGE COMPUTATIONAL-1.

02  A PICTURE 9999.
02  B REDEFINES A PICTURE 9V999.
02  C REDEFINES A PICTURE 99V99.
```

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 which are themselves redefined. The following would therefore be a valid use of the REDEFINES clause:

```
02  REGULAR-EMPLOYEE DISPLAY.
    03  LOCATION PICTURE A(8).
    03  STATUS PICTURE X(4).
    03  SEMI-MONTHLY-PAY
        PICTURE 9999V99.
    03  WEEKLY-PAY REDEFINES SEMI-
        MONTHLY-PAY PICTURE 999V999.

02  TEMPORARY-EMPLOYEE REDEFINES
    REGULAR-EMPLOYEE DISPLAY.
    03  LOCATION PICTURE A(8).
    03  FILLER PICTURE X(6).
    03  HOURLY-PAY PICTURE 99V99.
```

REDEFINES clauses may also be specified for items subordinate to items containing REDEFINES clauses. For example:

```
02  REGULAR-EMPLOYEE DISPLAY.
    03  LOCATION PICTURE A(8).
    03  STATUS PICTURE X(4).
    03  SEMI-MONTHLY-PAY
        PICTURE 9999V99.
    03  WEEKLY-PAY REDEFINES SEMI-
        MONTHLY-PAY PICTURE 999V999.

02  TEMPORARY-EMPLOYEE REDEFINES
    REGULAR-EMPLOYEE.
    03  LOCATION PICTURE A(8).
    03  FILLER PICTURE X(6).
    03  HOURLY-PAY PICTURE 99V99.
    03  PAY-CODE REDEFINES HOURLY-
        PAY PICTURE 9999.
```

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

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

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.

[EXT] LINKAGE SECTION

The Linkage Section describes data passed from another program.

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 doubleword boundary.

The Linkage Section is required in any program in which an ENTRY statement with a USING option appears. A complete discussion of the ENTRY statement is contained in the chapter entitled "Procedure Division."

The Procedure Division of a source program specifies those actions 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 with the words PROCEDURE DIVISION in margin A followed by a period. This heading must appear on a line by itself. The end of the Procedure Division (and the physical end of the program) is that physical position in a COBOL source program after which no further procedures appear.

SYNTAX

The discussion that follows describes the units of expression that constitute the Procedure Division and the way in which they may be combined. The smallest unit of expression in the Procedure Division is the statement. Sentences, paragraphs, and sections are the larger units of 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.

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 (ENTER, EXIT, NOTE) and its operands. Compiler-directing statements (except for NOTE) 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.

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 RETURN statement in the Sort Feature[1]
3. A WRITE statement with the INVALID KEY option
4. A REWRITE statement[2] with the INVALID KEY option
5. An arithmetic statement with the SIZE ERROR option
6. An ON statement[2]
7. 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 or RETURN statement
2. INVALID KEY in a WRITE or REWRITE statement
3. SIZE ERROR in an arithmetic statement
4. The count-condition in an ON statement
5. One of five tests in an IF statement

---

[1]Implemented for COBOL F only
[2]Extension

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 with their respective statements. The types of conditions evaluated in an IF statement are discussed below, under "Test-Conditions."

SENTENCES

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

PARAGRAPHS

A paragraph is a logical entity consisting of one or more sentences. Each paragraph must begin with a paragraph-name.

Paragraph-names are procedure-names and as such follow the rules for word formation (see "Word Formation" in the chapter "Basic Facts"). A paragraph-name must begin in Margin A but need not be on a line by itself.

A paragraph-name must not be duplicated within the same section. When used as operands in Procedure Division statements, non-unique 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 declarative sections, a paragraph ends at the next paragraph-name, section-name, or at END DECLARATIVES.

A paragraph-name need not be qualified when referred to from within the section in which it is contained.

SECTIONS

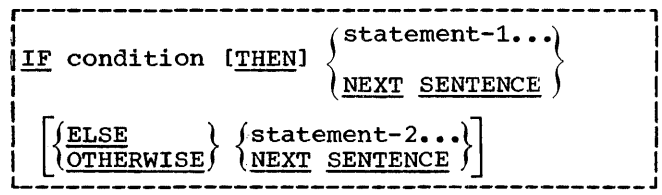
A section is composed of one or more successive paragraphs, and must begin with a section header. A section header consists of a section-name conforming to the rules for procedure-name formation, followed by the word SECTION and a period. A section header must begin in Margin A and appear on a line by 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 the chapter "Source Program library facility." A section-name need not immediately follow the words PROCEDURE DIVISION or END DECLARATIVES.

A section ends at the next section-name, at the end of the Procedure Division, or, in the case of declarative sections, at END DECLARATIVES.

Note: Section-names may be used like paragraph-names as operands in procedure-branching statements. In these statements. the section-name only is to be used, not the word SECTION. For example, the statement PERFORM UPDATE SECTION is incorrect. The correct statement is PERFORM UPDATE.

IF STATEMENT

The format of the IF statement is:



ELSE (or OTHERWISE) NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence. The following are examples of the IF statement:

```
IF SALES ARE NOT EQUAL TO SALES-QUOTA COM-
  PUTE STANDARD-RATE = SALES * BASE.

IF AMOUNT IS LESS THAN 2000 MOVE 'INVENTORY
  COUNT' TO PRINTER-AREA ELSE NEXT
  SENTENCE.

IF MONTH IS EQUAL TO 10 GO TO CALC-1 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 and the conditional is an IF or ON statement, the statements following ELSE or OTHERWISE (or the next sentence if the ELSE/OTHERWISE option is omitted) 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).

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

1. A period

2. An ELSE or OTHERWISE associated with a previous IF or ON

In a series of imperative statements executed when a condition is true, only the last statement may be an unconditional GO TO statement or a STOP RUN statement; otherwise, the series 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 7 is a flowchart showing how an IF or ON conditional statement is evaluated. Figure 8 is a flowchart showing how a conditional statement other than IF or ON is evaluated.

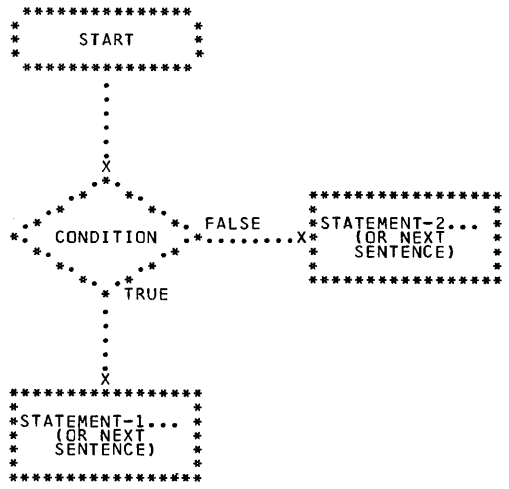


Figure 7. Evaluation of IF or ON Conditional Statement

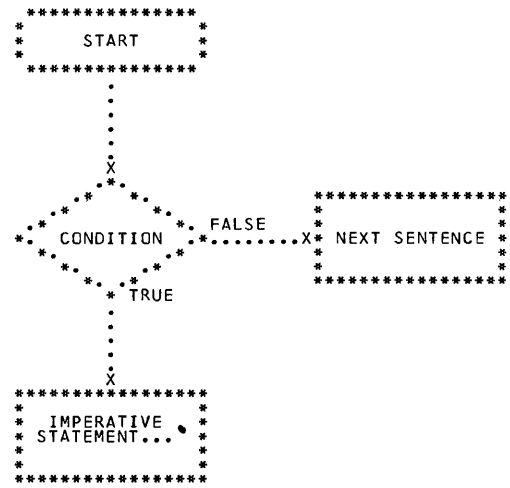


Figure 8. Evaluation of Conditional Statement other than IF or ON.

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 9, 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 10 is a flowchart indicating the logical flow of the conditional statement in Figure 9.

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 which when preceded by the word IF, constitute one of the five types of tests: relation test, sign test, class test, condition-name test, and 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.

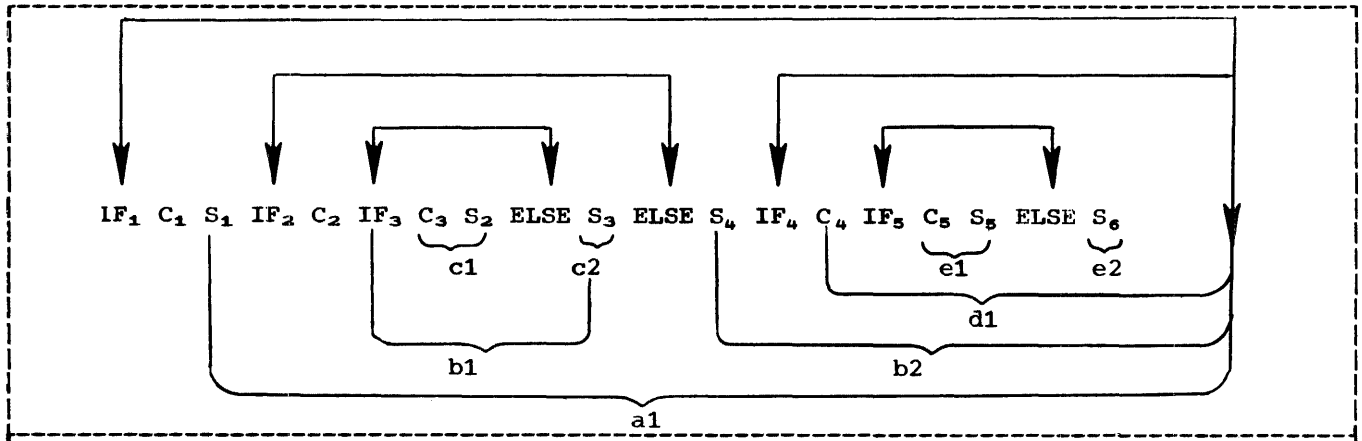
Relation Test

A relation test involves the comparison of two operands, either of which can be a data-name, a literal, or an arithmetic expression. Neither the comparison of two literals nor the comparison of an arithmetic expression to a non-numeric data-name is 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:

```
{data-name-1          }
{arithmetic-expression-1} IS [NOT]
{figurative-constant-1}
{literal-1            }

{>             }     {data-name-2          }
{<             }     {arithmetic-expression-2}
{=             }     {figurative-constant-2}
{GREATER       }     {literal-2            }
{LESS THAN     }
{EQUAL TO      }
```



a1--Statement-1 for IF_1 (If C_1 is false, the next sentence is executed, since there is no ELSE for it.)
b1--Statement-1 for IF_2
b2--Statement-2 for IF_2
c1--Statement-1 for IF_3
c2--Statement-2 for IF_3
d1--Statement-1 for IF_4 (If C_4 is false, the next sentence is executed, since there is no ELSE for it.)
e1--Statement-1 for IF_5
e2--Statement-2 for IF_5

Figure 9. Conditional Statements with Nested IF Statements

alphanumeric field called FIELD is being tested and contains the hexadecimal configuration C1, both of the following will be true because hexadecimal C1 could be interpreted either as an A or as a +1:

```
IF FIELD IS ALPHABETIC MOVE 'A' TO CODE-A.

IF FIELD IS NUMERIC MOVE 'N' TO CODE-N.
```

- Table 5. Permissible Comparisons

	Second Operand											
First Operand	GR	AL	AN	ED	ID	BI	EF	IF	RP	FC	SN[5]	SR[5]
Group Item (GR)	NN	NN	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					NN	NN	NN	NN
External Decimal Item (ED)	NN		NN	NU	NU	NU	NU	NU		NU[3] [6]	NU	
Internal Decimal Item (ID)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
Binary Item (BI)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
External Floating-point Item (EF)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
Internal Floating-point Item (IF)	NN			NU	NU	NU	NU	NU		NU[2]	NU	
Report Item (RP)	NN		NN						NN	NN[4]		
Figurative Constant (FC)	NN	NN[1]	NN	NU[3] [6]	NU[2]	NU[2]	NU[2]	NU[2]	NN[4]		NN[3]	NN[4]
Sterling Nonreport Item (SN)[5]	NN		NN	NU	NU	NU	NU	NU		NN[3]	NU	
Sterling Report Item (SR)[5]	NN		NN							NN[4]		NN

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]F only
[6]A non-numeric comparison is generated for the figurative constant ALL 'character' in COBOL F only

Table 6. Valid Forms of Class Test

Type of Item	Only Valid Forms of Class Test	
Alphabetic	ALPHABETIC	NOT ALPHABETIC
Alphanumeric	ALPHABETIC NUMERIC	NOT ALPHABETIC NOT NUMERIC
Internal or External Decimal	NUMERIC	NOT NUMERIC

Condition-Name Test

The format for a 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 1.
   88 MARRIED VALUE 2.
   88 DIVORCED VALUE 3.
```

Then, instead of writing

```
   IF MARITAL-STATUS = 2 SUBTRACT
   MARRIED-DEDUCTION FROM GROSS,
```

the programmer would write

```
   IF MARRIED SUBTRACT
   MARRIED-DEDUCTION FROM GROSS
```

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

EXT Overflow Test

This is a test for form-overflow of a printer where overflow-name is the condition-name specified in the Option 2 APPLY clause for a file. A form-overflow condition exists when an end-of-page is sensed by an on-line printer.

Use of the overflow test causes the form-overflow indicator to be turned off.

The format for the overflow test is:

```
[NOT] overflow-name
```

The overflow test is true if a form-overflow condition exists. Overflow-name follows the rules for data-name formation.

The following statement could be written (with a programmer-supplied overflow-name):

```
IF OVERFLOW-NAME-ONE WRITE X AFTER ADVANC-
  ING 0 LINES ELSE WRITE X AFTER ADVANCING
  2 LINES.
```

Form overflow condition should not be tested unless the associated file has been opened.

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 or 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 Table 7, where A and B represent simple test-conditions.

Table 7. Truth Table

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

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 Table 8. The letter C stands for conditional expression. P means that the combination is permissible. A dash means that the combination is not permissible.

Table 8. Formation of Symbol Pairs

		Second Symbol					
		C	OR	AND	NOT	(	)
First Symbol	C	-	P	P	-	-	P
	OR	P	-	-	P	P	-
	AND	P	-	-	P	P	-
	NOT	P	-	-	-	P	-
	(	P	-	-	P	P	-
	)	-	P	P	-	-	P

F ONLY

IMPLIED SUBJECTS AND OPERATORS

Simple relation test test-conditions may have implied first operands (subjects) when combined to form compound conditions. The following is the format for a series of relation tests forming a compound condition with implied first operands. The relational operators are GREATER THAN, LESS THAN, >, etc.

```
operand-1 IS [NOT]
   relational-operator operand-2
   {AND} [NOT]
   {OR }

   relational-operator operand-3...
```

Thus, the following statement could be made:

```
IF ACCOUNT-NUMBER IS GREATER THAN COUNT-A
  AND NOT LESS THAN COUNT-B OR = COUNT-C GO
  TO Z.
```

A relational operator can be implied only when a first operand is also implied. The following is the format for a series of relation tests forming a compound condition with implied first operand and relational operators.

```
operand-1 IS [NOT] relational-operator
   operand-2
   {AND} operand-3...
   {OR }
```

Thus, the following statement could be made:

```
IF ACCOUNT-NUMBER GREATER THAN COUNT-A AND
  COUNT-B OR COUNT-C GO TO Z.
```

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: unary plus or minus, then exponentiation, then multiplication and division, and finally addition and subtraction. Thus, the expression A + B / C + D ** E * F - G is read as 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, consecutive additions or subtractions, or consecutive exponentiation) 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.

The plus and minus signs are the only allowable unary operators. A unary operator is an operator having only one operand. The unary plus or minus sign must be the first character of an arithmetic expression, or must be immediately preceded by a left parenthesis.

COMPILER-DIRECTING DECLARATIVE SECTIONS

Declarative sections are identified by compiler-directing statements that specify the circumstances under which a procedure is to be executed in the object program.

A declarative section consists 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 each be followed by a period. A declarative section is terminated by the occurrence of another section or the words END DECLARATIVES.

Although declarative sections are located at the beginning of the Procedure Division, execution of the object program starts with the first procedure following the termination of the declarative sections.

The general form for a declarative section is:

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

The procedure-branching statements ALTER, GO TO, PERFORM, STOP RUN, and STOP literal can refer to a declarative section or to paragraph names within it or can appear within declaratives. Restrictions on the appearance of these statements are given in Table 18.

┌──────┐
│F ONLY│
└──────┘ The SORT statement should not appear within a declarative section.

USE Sentence

The USE sentence identifies the type of declarative.

There are two options of the USE sentence. Each is associated with one of the following types of procedures:

1. Report-writing procedures
2. Input/output error-checking procedures

┌──────┐
│F ONLY│
└──────┘ Option 1

Option 1 of the USE declarative is used to designate procedures that are to be executed by the Report Writer before the report group specified by data-name is produced; data-name may be the name of any type of report group except DETAIL. The