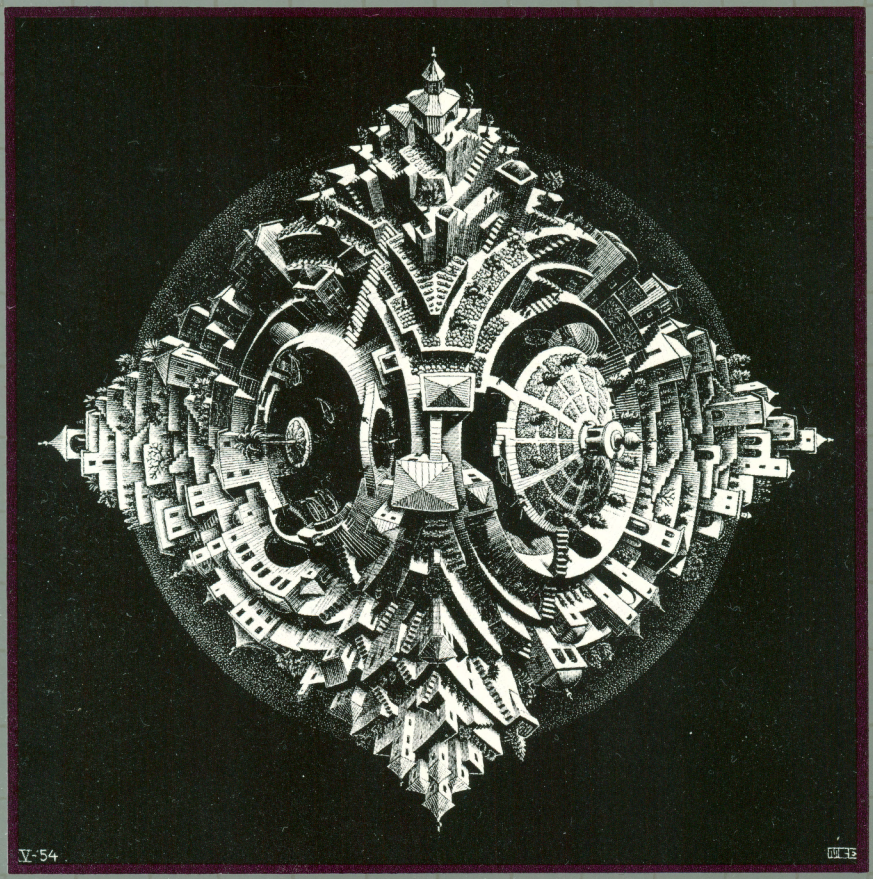
Apple III

COBOL

Language Reference Manual Volume 1



Customer Satisfaction

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable diskettes and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

Limitation on Warranties and Liability

Even though Apple has tested the software described in this manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is", and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

This manual and the software (computer programs) described in it are copyrighted by Apple or by Apple's software suppliers, with all rights reserved. Under the copyright laws, this manual or the programs may not be copied, in whole or part, without the written consent of Apple, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given or loaned to another person. Under the law, copying includes translating into another language.

You may use the software on any computer owned by you but extra copies cannot be made for this purpose. For some products, a multi-use license may be purchased to allow the software to be used on more than one computer owned by the purchaser, including a shared-disk system. (Contact your authorized Apple dealer for information on multi-use licenses.)

Product Revisions

Apple cannot guarantee that you will receive notice of a revision to the software described in this manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple Dealer.

© Micro Focus, Inc. 1978, 1982
1860 Embarcadero Road
Palo Alto, CA 94303

© Apple Computer, Inc. 1982
20525 Mariani Avenue
Cupertino, California 95014

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the U.S.A and Canada.

Reorder Apple Product #A3D0021

Apple III COBOL

Language Reference Manual

Volume 1

ACKNOWLEDGEMENTS

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

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

The authors and copyright holders of the copyrighted material used herein:

FLOW-MATIC (Trademark for 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, DSI27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

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

Contents

Volume 1

Preface 1

2 Manual Organization
4 Notation in This Manual

Chapter 1

Introduction 7

7 What is Apple III COBOL?
9 Program Structure
10 Formats and Rules
10 General Format
10 Syntax Rules
10 General Rules
10 Elements
11 Source Format
11 Sequence Number
11 Indicator Area

Chapter 2

COBOL Concepts 13

13 Language Concepts
13 Character Set
13 Language Structure
14 Separators
15 Character-Strings
15 COBOL Words
15 User-Defined Words
16 Condition-Name
16 Mnemonic-Name
17 Paragraph-Name
17 Section-Name
17 Other User-Defined Names
17 System-Names
17 Reserved Words
18 Key Words
18 Optional Words

18 Connectives
18 Figurative Constants
18 Literals
18 Nonnumeric Literals
19 Numeric Literals
19 Figurative Constant Values
21 PICTURE Character-Strings
21 Comment-Entries
21 Concept of Computer Independent Data Description
22 Concept of Levels
22 Level-Numbers
23 Concept of Classes of Data
24 Selection of Character Representation and Radix
26 Algebraic Signs
26 Standard Alignment Rules
27 Uniqueness of Reference
27 Qualification
28 Subscripting
29 Indexing
30 Identifier
30 Condition-Name
31 Explicit and Implicit Specifications
31 Explicit and Implicit Procedure Division References
31 Explicit and Implicit Transfers of Control
33 Explicit and Implicit Attributes
33 Program Structure
33 The "ANSI Switch" Compiler Directive
34 Identification Division
34 General Description
34 Organization
34 Structure
34 General Format
35 Environment Division
35 General Description
35 Organization
35 Structure
35 General Format
36 Data Division
36 Overall Approach
36 Physical and Logical Aspects of Data Description
36 Data Division Organization
37 General Format
38 Procedure Division
38 General Description
38 Declaratives
38 Procedures
39 Execution
39 General Format
39 Header
39 Body
39 Statements and Sentences
40 Conditional Statement
40 Conditional Sentence
40 Compiler Directing Statement

40 Compiler Directing Sentence
41 Imperative Statement
41 Imperative Sentence
42 Reference Format
42 General Description
42 Reference Format Representation
43 Sequence Numbers
43 Continuation of Lines
43 Blank Lines
44 Division, Section, Paragraph Formats
44 Division Header
44 Section Header
44 Paragraph Header, Paragraph-Name and Paragraph
44 Data Division Entries
45 Declaratives
45 Comment Lines
46 Reserved Words

Chapter 3

The Nucleus 47

47 Function of the Nucleus
47 Overall Language
47 Name Characteristics
47 Figurative Constants
47 Reference Format
48 Identification Division in the Nucleus
48 General Description
48 Organization
49 The PROGRAM-ID Paragraph
49 The DATE-COMPILED Paragraph
50 Environment Division in the Nucleus
50 Configuration Section
50 The SOURCE-COMPUTER Paragraph
50 The OBJECT-COMPUTER Paragraph
51 The SPECIAL-NAMES Paragraph
55 Data Division in the Nucleus
55 Working-Storage Section
55 Noncontiguous Working-Storage
55 Working-Storage Records
55 Initial Values
55 The Data Description - Complete Entry Skeleton
58 The BLANK WHEN ZERO Clause
59 The DATA-NAME or FILLER Clause
60 The JUSTIFIED Clause
61 LEVEL-NUMBER
62 The PICTURE Clause
62 Function
62 General Format
62 Syntax Rules
62 General Rules

62 Alphabetic Data Rules
62 Numeric Data Rules
63 Alphanumeric Data Rules
63 Alphanumeric Edited Data Rules
63 Numeric Edited Data Rules
64 Elementary Item Size
64 Symbols Used
66 Editing Rules
67 Simple Insertion Editing
67 Special Insertion Editing
67 Fixed Insertion Editing
68 Floating Insertion Editing
69 Zero Suppression Editing
69 Precedence Rules
72 The REDEFINES Clause
74 The RENAMES Clause
76 The SIGN Clause
78 The SYNCHRONIZED Clause
80 The USAGE Clause
81 The VALUE Clause
81 Function
81 General Format
81 Syntax Rules
81 General Rules
82 Condition-Name Rules
82 Data Description Entries other than Condition-Names
84 Procedure Division in the Nucleus
84 Arithmetic Expressions
84 Definition of an Arithmetic Expression
84 Arithmetic Operators
84 Formation and Evaluation Rules
86 Conditional Expressions
86 Simple Conditions
86 Relation Condition
87 Comparison of Numeric Operands
87 Comparison of Nonnumeric Operands
88 Class Condition
89 Condition-Name Condition
89 Switch-Status Condition
90 Sign Condition
90 Complex Conditions
91 Negated Simple Conditions
91 Combined and Negated Combined Conditions
92 Abbreviated Combined Relation Conditions
93 Condition Evaluation Rules
95 Common Phrases and General Rules for Statement Formats
95 The ROUNDED Phrase
95 The SIZE ERROR Phrase
95 SIZE ERROR Phrase Not Specified
95 SIZE ERROR Phrase Specified
96 The CORRESPONDING Phrase
96 Arithmetic Statements
97 Overlapping Operands
97 Multiple Results in Arithmetic Statements

97 Incompatible Data
98 CRT Devices
99 The ACCEPT Statement
104 The ADD Statement
106 The ALTER Statement
107 The COMPUTE Statement
108 The DISPLAY Statement
111 The DIVIDE Statement
114 The ENTER Statement
115 The EXIT Statement
116 The GO TO Statement
118 The IF Statement
120 The INSPECT Statement
128 The MOVE Statement
132 The MULTIPLY Statement
134 The PERFORM Statement
142 The STRING Statement
145 The STOP Statement
146 The SUBTRACT Statement
148 The UNSTRING Statement

Chapter 4

Table Handling 153

153 Introduction to the Table Handling Module
153 Data Division in the Table Handling Module
153 The OCCURS Clause
156 The USAGE Clause
157 Procedure Division in the Table Handling Module
157 Relation Condition
157 Comparisons Involving Index-names and/or Index Data Items
157 Overlapping Operands
157 The SEARCH Statement
162 The SET Statement

Chapter 5

Sequential Input and Output 165

165 Introduction to the Sequential I-O Module
165 Language Concepts
165 Organization
165 Access Mode
165 Current Record Pointer
165 I-O Status
165 Status Key 1
166 Status Key 2
167 Valid Combinations of Status Keys 1 and 2
167 The AT END Condition

167 LINAGE-COUNTER
168 Environment Division in the Sequential I-O Module
168 Input-Output Section
168 The FILE-CONTROL Paragraph
168 The FILE-CONTROL Entry
170 The I-O Control Paragraph
172 Data Division in the Sequential I-O Module
172 File Section
172 Record Description Structure
173 The File Description - Complete Entry Skeleton
174 The BLOCK CONTAINS Clause
174 The CODE-SET Clause
174 The DATA RECORDS Clause
175 The LABEL RECORDS Clause
175 The LINAGE Clause
179 The RECORD CONTAINS Clause
179 The VALUE OF Clause
181 Procedure Division in the Sequential I-O Module
181 The CLOSE Statement
182 The OPEN Statement
185 The READ Statement
188 The REWRITE Statement
190 The USE Statement
191 The WRITE Statement

Chapter 6

Relative Input and Output

197

197 Introduction to the Relative I-O Module
197 Language Concepts
197 Organization
197 Access Modes
197 Current Record Pointer
197 I-O Status
198 Status Key 1
198 Status Key 2
199 Valid Combinations of Status Keys 1 and 2
200 The INVALID KEY Condition
200 The AT END Condition
201 Environment Division in the Relative I-O Module
201 Input-Output Section
201 The FILE-CONTROL Paragraph
201 The FILE CONTROL Entry
204 The I-O CONTROL Paragraph
206 Data Division in the Relative I-O Module
206 File Section
206 Record Description Structure
206 The File Description - Complete Entry Skeleton
207 The BLOCK CONTAINS Clause
208 The DATA RECORDS Clause
208 The LABEL RECORDS Clause

209 The RECORD CONTAINS Clause
210 The VALUE OF Clause
211 Procedure Division in the Relative I-O Module
211 The CLOSE Statement
212 The DELETE Statement
213 The OPEN Statement
216 The READ Statement
219 The REWRITE Statement
221 The START Statement
223 The USE Statement
225 The WRITE Statement

Chapter 7

Indexed Input and Output

229

229 Introduction to the Indexed I-O Module
229 Language Concepts
229 Organization
229 Access Modes
230 Current Record Pointer
230 I-O Status
230 Status Key 1
231 Status Key 2
232 Valid Combinations of Status Keys 1 and 2
232 The INVALID KEY Condition
233 The AT END Condition
234 Environment Division in the Indexed I-O Module
234 Input-Output Section
234 The FILE-CONTROL Paragraph
234 The FILE CONTROL Entry
236 The I-O CONTROL Paragraph
239 Data Division in the Indexed I-O Module
239 File Section
239 Record Description Structure
240 The File Description - Complete Entry Skeleton
240 The BLOCK CONTAINS Clause
241 The DATA RECORDS Clause
241 The LABEL RECORDS Clause
242 The RECORD CONTAINS Clause
242 The VALUE OF Clause
244 Procedure Division in the Indexed I-O Module
244 The CLOSE Statement
245 The DELETE Statement
246 The OPEN Statement
249 The READ Statement
253 The REWRITE Statement
255 The START Statement
257 The USE Statement
259 The WRITE Statement

Figures and Tables 263

Index 265

Volume 2

Chapter 8

Sort-Merge 1

1 Introduction to the Sort-Merge Module
1 Relationship with Sequential I-O Module
1 Environment Division in the Sort-Merge Module
1 Input-Output Section
1 The FILE-CONTROL Paragraph
1 The File Control Entry
2 The I-O Control Paragraph
4 Data Division in the Sort-Merge Module
4 File Section
4 The Sort-Merge File Description - Complete Entry Skeleton
4 The DATA RECORDS Clause
5 The RECORD CONTAINS Clause
6 Procedure Division in the Sort-Merge Module
6 The MERGE Statement
10 The RELEASE Statement
11 The RETURN Statement
13 The SORT Statement

Chapter 9

Segmentation 19

19 Introduction to the Segmentation Module
19 General Description of Segmentation
19 Organization
19 Program Segments
19 Fixed Portion
19 Independent Segments
20 Segmentation Classification
20 Segmentation Control
21 Structure of Program Segments
21 Segment-Numbers
21 General Format
21 Syntax Rules
21 General Rules

22 Restrictions on Program Flow
22 The ALTER Statement
22 The PERFORM Statement
22 Extra Intermediate Code Files
23 The MERGE Statement
23 The SORT Statement

Chapter 10

Library 25

25 Introduction to the Library Module
26 The COPY Statement

Chapter 11

Debug and Debugging with the Animation Option 27

27 Introduction
27 Apple III COBOL Animation Option
27 Standard ANSI COBOL Debug
28 Compile-Time Switch
28 COBOL Debug Object Time Switch
28 Environment Division in COBOL Debug
28 The WITH DEBUGGING MODE Clause
29 Procedure Division in COBOL Debug
29 The USE FOR DEBUGGING Statement
32 Debugging Lines

Chapter 12

Interprogram Communication 33

33 Introduction to the Inter-Program Communication Module
33 Data Division in the Inter-Program Communication Module
33 Linkage Section
34 Noncontiguous Linkage Storage
35 Procedure Division in the Inter-Program Communication Module
35 The Procedure Division Header
36 The CALL Statement
38 The CANCEL Statement
39 The EXIT PROGRAM Statement

Chapter 13

Communication 41

41 Introduction to the Communication Module

41 Function
41 Data Division in the Communication Module
41 Communication Section
41 The Communication Description - Complete Entry Skeleton
52 Procedure Division in the Communication Module
52 The ACCEPT MESSAGE COUNT Statement
53 The DISABLE Statement
55 The ENABLE Statement
57 The RECEIVE Statement
60 The SEND Statement

Chapter 14

Programming Techniques, Useful Hints and Program Sizing 65

65 Programming Techniques
65 Useful Hints
66 Sizing
66 General Description
66 Data Dictionary

Appendix A

Reserved Word List 69

Appendix B

Character Sets and Collating Sequence 73

Appendix C

Glossary 75

75 Introduction
75 Definitions

Appendix D

Compile-Time Errors 97

Appendix E

Run-Time Errors 101

Appendix F

Syntax Summary 103

Appendix G

Summary of Extensions to ANSI COBOL 121

121 Screen Formatting and Data Entry
121 The ACCEPT Statement
121 The DISPLAY Statement
122 Disk Files
122 Line Sequential Files
122 Run Time Input of File Names
122 Lower Case Characters
122 Hexadecimal Values
122 Interactive Debugging

Appendix H

System Dependent Language Features 123

123 Mandatory Changes
123 Environment Division
123 Configuration Section
123 Input-Output Section
123 Statements Compiled as Documentation Only
123 Environment Division
124 Data Division
124 Procedure Division

124 Data Division
124 Procedure Division

Appendix I

125

Language Specification

Appendix J

131

IBM Extensions

Appendix K

Table of Possible MOVEs in a COBOL Program 133

Appendix L

135

Table of MOVE Data Categories

Appendix M

Table of Valid PICTURE Clause Sequences 137

Appendix N

139

SET Statement Valid Operations

Appendix O
Permissible I-O Statements 141

Figures and Tables 143

Index 145

Preface

This book describes the Apple III COBOL language for programming the Apple III computer. The Apple III COBOL language is based on the ANSI COBOL standard X3.23 (1974) (see acknowledgement). It also describes the additional APPLE III COBOL features that exploit the capabilities of the Apple III.

Use the book to find information you may need while learning and using the Apple III COBOL programming language. This manual is not a tutorial, and if you are not familiar with COBOL, you should first study a good textbook on the subject. If you are unfamiliar with the Apple III computer, you should first read the Apple III Owner's Guide that is packed with your computer.

As a reference book, this volume together with its companion volume the Apple III COBOL Introduction and Operating System Manual, contains everything you will normally need to write and run COBOL programs.

MANUAL ORGANIZATION

Chapters 1 through 4 of the manual apply to all users and describe basic features of the language. Chapters 5 through 7 describe language features for programming the three file organization formats supported: sequential, relative and indexed.

Chapters 8 through 11 apply to all users and describe additional features and facilities available with the standard language. The appendices supply reference information pertinent to all systems.

The manual contains the following chapters and appendices:

"Chapter 1. Introduction", which gives a general description of the language, including a broad outline of ANSI COBOL features included and omitted and additional features of Apple III COBOL.

"Chapter 2. COBOL Concepts", which describes general concepts of the COBOL language including program structure, and details of statement components and notation.

"Chapter 3. Nucleus", which describes the nucleus of all COBOL programs and the layout of each program division in the nucleus.

"Chapter 4. Table Handling", which describes the handling of data tables in the Data and Procedure divisions of a COBOL program.

"Chapter 5. Sequential Input and Output", which describes the programming of input and output of data in files with sequential format.

"Chapter 6. Relative Input and Output", which describes the programming of input and output of data in files with relative format.

"Chapter 7. Indexed Input and Output", which describes the programming of input and output of data in files with indexed format.

"Chapter 8. Sort-Merge", which describes the facility to order one or more files of records or to combine two or more identically ordered files of records according to a set of user-specified keys contained within each record.

"Chapter 9. Segmentation", which describes the facility for specifying permanent and independent object program segments.

"Chapter 10. Library", which describes the source library maintenance feature of COBOL.

"Chapter 11. Debug and Interactive Debugging", which describes the basic and interactive debugging features available in Apple III COBOL.

"Chapter 12. Interprogram Communication", which describes the ability of Apple III COBOL programs to interface during running and to access common data, enabling modular programming.

"Chapter 13. Communication", which describes the facility to communicate through a Message Control System (MCS) with local and remote devices, and to access, process and create messages or portions thereof.

"Chapter 14. Programming Techniques and Sizing", which describes the means available for Apple III COBOL programmers to estimate object program size and includes programming techniques in Apple III COBOL.

"Appendix A. Reserved Word Table", which lists words reserved for Apple III COBOL functions within a program.

"Appendix B. Character Set and Collating Sequence", which lists all characters available and their collating sequence.

"Appendix C. Glossary", which lists specific terms used in Apple III COBOL.

"Appendix D. Compile - Time Errors", which lists all errors that can be signalled during program compilation.

"Appendix E. Run-Time Errors", which lists all errors that can be signalled during program execution.

"Appendix F. Syntax Summary", which summarizes the syntax used in Apple III COBOL programming.

"Appendix G. Summary of Extensions to ANSI COBOL", which summarizes all extensions to ANSI COBOL provided by Apple III COBOL.

"Appendix H. System Dependent Language Features", which describes the system dependent Apple III COBOL entries for use with microcomputers and those features not included because of hardware requirements.

"Appendix I. Language Specification", which is an overall specification of the Apple III COBOL language.

"Appendix J. IBM Extensions", which describes some Apple III COBOL extensions that are compatable with the IBM 8100 DPPX COBOL implementation.

"Appendix K, Table of Possible MOVEs in a COBOL Program", "Appendix L, Table of MOVE Data Categories", "Appendix M, Table of Valid PICTURE Clause Sequences", "Appendix N, SET Statement Valid Operations", and "Appendix O, Permissible I-O Statements and File OPEN Modes", which are all short tables giving important information about constraints on the COBOL language.

NOTATION IN THIS MANUAL

Throughout this manual, the following notation is used to describe the format of COBOL statements:

1. All words printed in capital letters which are underlined must always be present when the functions of which they are a part are used. An error printout will occur during compilation if the underlined words are absent or incorrectly spelled. The underlining is not necessary when writing a COBOL source program.

2. All words printed in capital letters which are not underlined are used for readability only. They may be written, or not, as the programmer wishes.

3. All words printed in small letters are generic terms representing names which will be devised by the programmer.

4. When material is enclosed in braces { }, a choice must be made from the options within them.

5. When material is enclosed in square brackets [ ], it is an indication that the material is an option which may be included or omitted as required.

6. When material is enclosed in square brackets crossed ǂ ǂ, it is an indication that the material is mandatory when the ANSI switch is set (see Chapter 2) but optional otherwise.

7. Language features that are shaded in the text are language extensions which exceed the ANSI standard.

8. In text, the ellipsis (...) shows the omission of a portion of a source program or a sequence. This meaning becomes apparent in context.

   In the general formats, the ellipsis represents the position at which repetition may occur at the user's option. The portion of the format that may be repeated is determined as follows:

   Given ... in a clause or statement format, scanning right to left, determine the { or [immediately to the left of the ...; continue scanning right to left and determine the logically matching } or]; the ... applies to the words between the determined pair of delimiters.

9. The term identifier means either a data-name or a subscripted data-name. An identifier takes the following form:

   data-name-1 [({ data-name-2 / literal-1 })]

   data-name-2 or literal-1 must be a positive integer in the range 1 to the number of elements in the table.

Headings are presented in this manual in the following order of importance:

CHAPTER N
TITLE
} Chapter Heading

ORDER ONE HEADING
ORDER TWO HEADING
Order Three Heading
Order Four Heading
} Text two lines down

Order Five Heading: Text on same line

Numbers one (1) to nine (9) are written in text as letters, e.g. one.

Numbers ten (10) upwards are written in text as numbers, e.g. 12.

The phrase "For documentation purposes only" in the text of this manual means that the associated coding is accepted syntactically by the Compiler, but is ignored when producing the object program.

Chapter 1
Introduction

WHAT IS APPLE III COBOL?

COBOL (COmmon Business Oriented Language) is today's most frequently used programming language for commercial and administrative data processing. Apple III COBOL is a compact, interactive and standard COBOL Language System.

It is based on the ANSI COBOL as specified in "American National Standard Programming Language COBOL" (ANSI X3.23 1974). The Apple III COBOL implementation has been selected from both levels of ANSI COBOL. The following modules are fully implemented at Level 1:

- Nucleus
- Table Handling
- Sequential Input and Output
- Relative Input and Output
- Indexed Input and Output
- Segmentation
- Library
- Inter-Program Communication
- Debug
- Sort Merge

In addition the following modules are fully implemented at Level 2:

- Nucleus
- Table Handling
- Sequential Input and Output
- Relative Input and Output
- Indexed Input and Output
- Inter-Program Communication
- Sort Merge

The Communications module is accepted by the compiler for syntax checking only. The Run-Time System will not execute this module if included in a program.

This manual is intended as a reference work for Apple III COBOL programmers and material from the ANSI COBOL language standard document is included.

The package has been proved to meet and exceed the COBOL ANSI standard X3.23 and has been certified by the Federal Compiler Testing Center (FCTC) under

the direction of the General Services Administration (GSA) as validated at Federal High Intermediate Level. The GSA Validation Summary Report is available under the reference FCTC-81/131.

Along with the ANSI implementation Apple III COBOL also contains several language extensions specifically oriented to the small computer environment and for compatibility with some larger mainframe applications. These enable a Apple III COBOL program to format CRT screens for data input and output (DISPLAY and ACCEPT), READ and WRITE text files efficiently and define external file names at run time.

The programmer wishing to transport an existing COBOL program to run under Apple III COBOL must check that the individual language features he has used are supported by Apple III COBOL. The COBOL SECTION statements in the Segmentation feature can be performed using the PERFORM statement.

A compile time FLAG directive can be set that flags all Apple III COBOL extension features, together with ANSI COBOL features at any of the levels specified by the Federal Compiler Testing Center under the direction of the General Services Administration (GSA). (See Chapter 2).

The Apple III COBOL compiler is designed to enable programs to be developed in a 128K machine. The Compiler supports sequential, relative and indexed sequential files, as well as interactive communications via the ACCEPT and DISPLAY verbs.

The Apple III COBOL System also contains a powerful utility called FORMS-2 that enables the Operator to define screen layouts from a screen "module" and produce automatically the data description for direct inclusion in a Apple III COBOL program. This is described in the FORMS-2 Operating Guide

Apple III COBOL programs are created using a conventional text editor. The Compiler compiles the programs and the Run-Time system links with the compiled output to form a running user program. A listing of the Apple III COBOL program is provided by the Compiler during compilation. Error messages are inserted in the listing. Interactive Debugging facilities are provided for run-time use, and these are described in the Apple III COBOL Operating Guide.

Apple III COBOL is designed to be interfaced easily to any microprocessor operating system. Detailed operating characteristics are dependent on the particular host operating system used and are defined in the appropriate Operating Guide.

PROGRAM STRUCTURE

A COBOL program consists of four divisions:

1. IDENTIFICATION DIVISION - An identification of the program

2. ENVIRONMENT DIVISION - A description of the equipment to be used to compile and run the program

3. DATA DIVISION - A description of the data to be processed

4. PROCEDURE DIVISION - A set of procedures to specify the operations to be performed on the data

Each division is divided into sections which are further divided into paragraphs which in turn are made up of sentences.

Within these subdivisions of a COBOL program, further subdivisions exist as clauses and statements. A clause is an ordered set of COBOL elements that specify an attribute of an entry, and a statement is a combination of elements in the Procedure Division that include a COBOL verb and constitute a program instruction.

FORMATS AND RULES

GENERAL FORMAT

A general format is the specific arrangement of the elements of a clause or a statement. Throughout this document a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the general format is separated into numbered formats. Clauses must be written in the sequence given in the general formats. (Clauses that are optional must appear in the sequence shown if they are used). In certain cases, stated explicitly in the rules associated with a given format, the clauses may appear in sequences other than that shown. Applications, requirements or restrictions are shown as rules.

SYNTAX RULES

Syntax rules are those rules that define or clarify the order in which words or elements are arranged to form larger elements such as phrases, clauses, or statements. Syntax rules also impose restrictions on individual words or elements.

These rules are used to define or clarify how the statement must be written, i.e., the order of the elements of the statement and restrictions on what each element may represent.

GENERAL RULES

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

ELEMENTS

Elements which make up a clause or a statement consist of uppercase words, lowercase words, level-numbers, brackets, braces, connectives and special characters (see Chapter 2).

SOURCE FORMAT

The COBOL source format divides each COBOL source record into 72 columns. These columns are used in the following way:

Columns 1 - 6	Sequence number
Column 7	Indicator area
Column 8 - 11	Area A
Columns 12 - 72	Area B

SEQUENCE NUMBER

A sequence number of six digits may be used to identify each source program line.

INDICATOR AREA

An asterisk * in this area marks the line as documentary comment only. Such a comment line can appear anywhere in the program after the Identification Division header. Any characters from the ASCII character set can be included in Area A and Area B of the line.

A stroke /, in the indicator area acts as a comment line above but causes the page to eject before printing the comment.

A "D" in the indicator area represents a debugging line. Areas A and B may contain any valid COBOL sentence.

A "-" in the indicator area represents the continuation of a non-numeric literal. The first non-blank character in Area B of the continuation line must be a quotation mark. The literal continues with the first character after the quotation mark. All spaces at the end of the continued line are significant.

Section names and paragraph names begin in Area A and are followed by a period and a space. Level indications FD, 01 and 66, 77 and 88 begin in Area A and are followed in Area B by the appropriate file and record description.

Program sentences may commence anywhere in Area A and Area B. More than one sentence is permitted in each source record.

Figure 1-1 shows the source format of a typical program.

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. STOCK-FILE-SET-UP.
000030 AUTHOR. MICRO FOCUS LTD.
000040 ENVIRONMENT DIVISION.
000050 CONFIGURATION SECTION.
000060 SOURCE-COMPUTER. APPLE-III.
000070 OBJECT-COMPUTER. APPLE-III.
000075 SPECIAL-NAMES. CONSOLE IS CRT.
000080 INPUT-OUTPUT SECTION.
000090 FILE-CONTROL.
000100     SELECT STOCK-FILE ASSIGN "STOCK.IT"
000110     ORGANIZATION INDEXED
000120     ACCESS DYNAMIC
000130     RECORD KEY STOCK-CODE.
000140 DATA DIVISION.
000150 FILE SECTION.
000160 FD  STOCK-FILE; RECORD 32.
000170 01  STOCK-ITEM.
000180     02  STOCK-CODE PIC X(4).
000190     02  PRODUCT-DESC PIC X(24).
000200     02  UNIT-SIZE PIC 9(4).
000210 WORKING-STORAGE SECTION.
000220 01  SCREEN-HEADINGS.
000230     02  ASK-CODE PIC X(21) VALUE "STOCK CODE      <    >".
000240     02  FILLER PIC X(59).
000250     02  ASK-DESC PIC X(16) VALUE "DESCRIPTION    <".
000260     02  SI-DESC PIC X(25) VALUE "                         >".
000270     02  FILLER PIC X(39).
000280     02  ASK-SIZE PIC X(21) VALUE "UNIT SIZE       <    >".
000290  01  ENTER-IT REDEFINES SCREEN-HEADINGS.
000300     02  FILLER PIC X(16).
000310     02  CRT-STOCK-CODE PIC X(4).
000320     02  FILLER PIC X(76).
000330     02  CRT-PROD-DESC PIC X(24).
000340     02  FILLER PIC X(56).
000350     02  CRT-UNIT-SIZE PIC 9(4).
000360     02  FILLER PIC X.
000370 PROCEDURE DIVISION.
000380 SR1.
000390     DISPLAY SPACE.
000400     OPEN I-O STOCK-FILE.
000410     DISPLAY SCREEN-HEADINGS.
000420 NORMAL-INPUT.
000430     MOVE SPACE TO ENTER-IT.
000440     DISPLAY ENTER-IT.
000450 CORRECT-ERROR.
000460     ACCEPT ENTER-IT.
000470     IF CRT-STOCK-CODE = SPACE GO TO END-IT.
000480     IF CRT-UNIT-SIZE NOT NUMERIC GO TO CORRECT-ERROR.
000490     MOVE CRT-PROD-DESC TO PRODUCT-DESC.
000500     MOVE CRT-UNIT-SIZE TO UNIT-SIZE.
000510     MOVE CRT-STOCK-CODE TO STOCK-CODE.
000520     WRITE STOCK-ITEM; INVALID GO TO CORRECT-ERROR.
000530     GO TO NORMAL-INPUT.
000540 END-IT.
000550     CLOSE STOCK-FILE.
000560     DISPLAY SPACE.
000570     DISPLAY "END OF PROGRAM".
000580     STOP RUN.
* Apple /// COBOL V1.0                                    URN AA/0000/HA
* Compiler  (C) 1982 Apple Computer Inc.
*
* ERRORS=00000 DATA=01024 CODE=00512 DICT=00476:61598/62074 GSA FLAGS =  OFF
```

Cols. 1-6 Sequence Number

Col 7 Indicator Area

Cols 8-11 Area A

Cols. 12-72 Area B

Inserted by Compiler

Figure 1-1. Sample Program Listing Showing Source Format.

Chapter 2

COBOL Concepts

LANGUAGE CONCEPTS

CHARACTER SET

The most basic and indivisible unit of the language is the character. The set of characters used to form Apple III COBOL character-strings and separators includes the letters of the alphabet, digits and special characters. The character set consists of the characters defined below:

```
0 to 9
A to Z
a to z (Reserved and User Word Characters
        read as:  A to Z)
Space
+     Plus sign
-     Minus sign or hyphen
*     Asterisk
/     Oblique Stroke/Slash
=     Equal sign
$     Dollar sign
.     Full stop or decimal point
,     Comma or decimal point
;     Semicolon
"     Quotation mark
(     Left Parenthesis
)     Right Parenthesis
>     Greater than symbol
<     Less than symbol
```

The Apple III COBOL language is restricted to the above character set, but the content of non-numeric literals, comment lines and data may include any of the characters from the ASCII character set. See Appendix B.

LANGUAGE STRUCTURE

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

Separators

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

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

2. The punctuation characters comma, semicolon and period, when immediately followed by a space, are separators. These separators may appear in a COBOL source program only where explicitly permitted by the general formats, by format punctuation rules (see FORMATS AND RULES in Chapter 1), by statement and sentence structure definitions (see STATEMENTS AND SENTENCES in this Chapter), or reference format rules (see REFERENCE FORMAT in this Chapter).

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

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

   Quotation marks may appear only in balanced pairs delimiting nonnumeric literals except when the literal is continued. (See CONTINUATION OF LINES in this Chapter).

5. The separator space may optionally immediately precede all separators except the following:

   a. As specified by reference format rules see REFERENCE FORMAT in this Chapter.

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

6. The separator space is optional and can immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

Any punctuation character which appears as part of the specification of a PICTURE character-string (see Chapter 3) or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals, comment-entries, or comment lines.

Character-Strings

A character-string is a character or a sequence of contiguous characters which forms an Apple III COBOL word, a literal, a PICTURE character-string, or a comment-entry. A character-string is delimited by separators.

COBOL Words

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

User-Defined Words: A user-defined word is a COBOL word that must be supplied by the user to satisfy the format of a clause or statement. Each character of a user-defined word is selected from the set of characters 'A', 'B', 'C', ... 'Z', 'a', 'b', 'c', ...'z', '0', ...'9', and '-', except that the '-' may not appear as the first or last character. The exception to this rule is a text-name which must be a normal alphanumeric literal.

User-defined word types which are implemented are as follows:

alphabet-name
cd-name
condition-name
data-name
file-name
index-name
level-number
mnemonic-name
paragraph-name
program-name
record-name
routine-name
section-name
segment-number
text-name

Within a given source program, 11 of these 15 types of user-defined words are grouped into nine disjoint sets. The disjoint sets are:

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

All user-defined words, except segment-numbers and level-numbers, can belong to one and only one of these disjoint sets. Further, all user-defined words within a given disjoint set must be unique, because no other user-defined word in the same source program has identical spelling or punctuation. (See UNIQUENESS OF REFERENCE in this Section.)

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

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

Condition-names may be defined in the Data Division or in the SPECIAL-NAMES paragraph within the Environment Division, where a condition-name must be assigned to the ON STATUS or OFF STATUS, or both, of the run time switches.

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

Mnemonic-Name: A mnemonic-name assigns a user-defined word to an implementor-name. These associations are established in the SPECIAL-NAMES paragraph of the Environment Division. (See SPECIAL-NAMES in Chapter 3.)

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

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

Other User-Defined Names: See the glossary in Appendix C for definitions of all other types of user-defined words.

System-Names: A system-name is a COBOL word which is used to communicate with the operating environment. Each character used in the formation of a system-name must be selected from the set of characters 'A', 'B', 'C', ... 'Z', 'a', 'b' ... 'z', '0', ... '9' and '-', except that the '-' may not appear as the first or last character.

There are three types of system-names:

1. computer-name
2. implementor-name
3. language-name

Within a given implementation these three types of system-names form disjoint sets; a given system-name may belong to one and only one of them.

The system-names listed above are individually defined in the glossary in Appendix C.

Reserved Words: A reserved word is a COBOL word that is one of a specified list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names. Reserved words can only be used as specified in the general formats. (See Appendix A.)

There are six types of reserved words:

1. Key words
2. Optional words
3. Connectives
4. Special registers
5. Figurative constants
6. Special-character words

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

Key words are of three types:

1. Verbs such as ADD, READ, and ENTER.
2. Required words, which appear in statement and entry formats.
3. Words which have a specific functional meaning such as NEGATIVE, SECTION, etc.

Optional Words: Within each format, uppercase words that are not underlined are called optional words and may appear at the user's option. The presence or absence of an optional word does not alter the semantics of the COBOL program in which it appears.

Connectives: 1. Qualifier connectives are used to associate a data-name, condition-name, text-name, or paragraph-name with its qualifier: OF, IN; 2. series connectives link two or more consecutive operands: , (separator comma) or ; (separator semicolon); 3.logical connectives are used in the formation of conditions: AND, OR.

Figurative Constants: Certain reserved words are used to name and reference specific constant values. These reserved words are specified under Figurative Constant Values in this chapter.

Literals

A literal is a character-string whose value is implied by an ordered set of characters of which the literal is composed or by specification of a reserved word which references a figurative constant. Every literal belongs to one of two types, nonnumeric or numeric.

Nonnumeric Literals: A nonnumeric literal is a character-string delimited on both ends by quotation marks and consisting of any allowable character in the computer's character set. Allowed are nonnumeric literals of 1 through 128 characters in length. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used. The value of a nonnumeric literal in the object program is the string of characters itself, except:

1. The delimiting quotation marks are excluded, and

2. Each embedded pair of contiguous quotation marks represents a single quotation mark character.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals are category alphanumeric. (See The PICTURE Clause in Chapter 3). In addition, hexadecimal binary values can be attributed to non-numeric literals by expressing literals as: X"nn", where n is a hexadecimal character in the set 0-9 A-F; nn may be repeated up to 128 times, but the number of hex digits must be even.

Numeric Literals: A numeric literal is a character-string whose characters are selected from the digits '0' through '9', the plus sign, the minus sign, and/or the decimal point. The implementation allows for numeric literals of 1 through 18 digits in length. The rules for the formation of numeric literals are as follows:

1. A literal must contain at least one digit.

2. A literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.

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

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

4. The value of a numeric literal is the algebraic quality represented by the characters in the numeric literal. Every numeric literal is category numeric. (See THE PICTURE CLAUSE in Chapter 3). The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

Figurative Constant Values

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

The figurative constant values and the reserved words used to reference them are shown in Table 2-1.

Table 2-1. Figurative Constants and their Reserved Words

CONSTANT	REPRESENTATION
ZERO	Represents the value '0', or one or more of the character '0' depending on context.
ZEROS ZEROES	
SPACE SPACES	Represents one or more of the character space from the computer's character set.
HIGH-VALUE HIGH-VALUES	Represents one or more of the character that has the highest ordinal position in the program collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more of the character that has the lowest ordinal position in the program collating sequence.
QUOTE QUOTES	Represents one or more of the character '"'. The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTE ABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD".
ALL literal	Represents one or more characters of the string of characters comprising the literal. The literal must be either a nonnumeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

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

1. When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character on the right until the size of the resultant string is equal to the size in characters of the associated data item. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

2. When a figurative constant is not associated with another data item, when the figurative constant appears in a DISPLAY, STRING, STOP or UNSTRING statement, the length of the string is one character.

DISPLAY SPACE is, of course, an exception.

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

When the figurative constants HIGH-VALUE(S) or LOW-VALUE(S) are used in the source program, the actual character associated with each figurative constant depends upon the program collating sequence specified. (See THE OBJECT-COMPUTER PARAGRAPH, and THE SPECIAL-NAMES PARAGRAPH in Chapter 3).

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

PICTURE Character-Strings

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. See THE PICTURE CLAUSE in chapter 3 for the PICTURE character-string and for the rules that govern their use.

Any punctuation character which appears as part of the specification of a PICTURE character-string is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string.

Comment-Entries

A comment-entry is an entry in the Identification Division that may be any combination of characters from the computer's character set.

CONCEPT OF COMPUTER INDEPENDENT DATA DESCRIPTION

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

Concept of Levels

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

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

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

Level-Numbers

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49. A maximum of 49 levels in a record is allowed. There are special level-numbers, 66, 77 and 88 which are exceptions to this rule (see below). Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item. Note that group items must not exceed 8191 bytes in length.

Three types of entries exist for which there is no true concept of level. These are:

1. Entries that specify elementary items or groups introduced by a RENAMES clause

2. Entries that specify noncontiguous working storage and linkage data items

3. Entries that specify condition-names.

Entries describing items by means of RENAMES clauses for the purpose of regrouping data items have been assigned the special level-number 66.

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

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

Concept of Classes of Data

The five categories of data items (see THE PICTURE CLAUSE in Chapter 3) are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric (without editing). Every elementary item except for an index data item belongs to one of the classes and further to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items subordinate to that group item. Table 2-2 depicts the relationship of the class and categories of data items.

Table 2-2. Data Levels, Classes and Categories.

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Non-Elementary Group	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

Selection of Character Representation and Radix

The value of a numeric item may be represented in either binary or decimal form, depending on the equipment. In addition, there are several ways of expressing decimal. Since these representations are actually combinations of bits, they are commonly called binary-coded decimal forms. The four standard formats used for storing numeric data in Apple III COBOL are as follows:

1. As alphanumeric characters stored one per byte in ASCII representation.

2. As numeric characters defined by USAGE IS DISPLAY (See The USAGE Clause in Chapter 3) one per byte in ASCII representation. If they are signed and the sign is specified as INCLUDED, bit 6 of the leading or trailing byte of the field is set for negative, depending on the field definition. If a SEPARATE sign is specified as a one byte ASCII + or -, a sign is added as the leading or trailing byte. If no SIGN clause is specified, bit 6 of the trailing digit is set to indicate negative by default.

3. As numeric characters defined by USAGE IS COMP or COMPUTATIONAL in pure binary form. If the field is signed the number is held in its twos-compliment form. Storage is then dependent on the number of 9's in the PICTURE clause (see The PICTURE Clause in Chapter 3) and on whether the field is SIGNed or not (see The SIGN Clause in Chapter 3).

   Table 2-3 shows the storage requirements for each COMP(UTATIONAL) PICTURE Clause.

   Table 2-3. Numeric Data Storage for the COMP(UTATIONAL) PICTURE Clause.

Bytes Required	Number of Characters	
	Signed	Unsigned
1	1-2	1-2
2	3-4	3-4
3	5-6	5-7
4	7-9	8-9
5	10-11	10-11
6	12-14	12-14
7	15-16	15-16
8	17-18	17-18

4. As numeric characters defined by USAGE IS COMPUTATIONAL-3 or USAGE IS COMP-3 in packed internal decimal form. Storage is dependent on the number of 9's in the PICTURE clause. The decimal numbers are stored as signed strings of variable length of 1 through 18 digits. The sign of the packed decimal number is always stored in place of the least significant quartet of the low order byte. Each byte contains two decimal positions (four bits per digit) and the digits (0 - 9) are encoded as BCD numbers (0000 - 1001). Numbers are represented in the field as right-justified values with a + or - sign as shown in the example below. The maximum number of digits permitted in arithmetic operands is 18.

EXAMPLE:

a. For COMPUTATIONAL-3 and PICTURE 9999, the number +1234 would be stored as follows:

```
... 0     1     2     3     4     F
    0000  0001  0010  0011  0100  1111
                          ----  ----
                          1 byte
```

where F represents the non-printing plus sign.

b. For COMPUTATIONAL-3 and PICTURE S9999, the number +1234 would be stored as follows:

Storage would be as in a above except that the least significant digit would be replaced by C (1100) representing the plus sign.

c. For COMPUTATIONAL-3 and PICTURE S9999, the number -1234 would be stored as follows:

Storage would be as in a above except that the least significant byte would be replaced by D (1101) representing the minus sign.

Table 2-4 shows the storage requirements for each COMP-3 clause.

Bytes Required	Number of Digits (Signed or Unsigned)
1	1
2	2-3
3	4-5
4	6-7
5	8-9
6	10-11
7	12-13
8	14-15
9	16-17
10	18

Table 2-4. Numeric Data Storage for the COMPUTATION-3 PICTURE Clause.

Algebraic Signs

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

The SIGN Clause permits the programmer to state explicitly, the location of the operational sign. The Clause is optional; if it is not used operational signs will be represented as defined by setting bit 6 of the trailing digit for ASCII numbers. (see above).

Editing signs are inserted into a data item through the use of the sign control symbols of THE PICTURE CLAUSE.

Standard Alignment Rules

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:

   a. The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.

   b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character and is aligned as in paragraph a. above.

2. If the receiving data item is a numeric edited data item, the data moved to the edited item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required.

If the JUSTIFIED Clause is specified for the receiving item, these standard rules are modified as described in THE JUSTIFIED CLAUSE in Chapter 3.

Uniqueness of Reference

Qualification

Every user-specified name that defines an element in a COBOL source program must be unique, either because no other name has the identical spelling and hyphenation, or because the name exists within a hierarchy of names such that references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and this process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. Within the Data Division, all data-names used for qualification must be associated with a level indicator or a level-number. Therefore, two identical data-names must not appear as entries subordinate to a group item unless they are capable of being made unique through qualification. In the Procedure Division two identical paragraph-names must not appear in the same section.

In the hierarchy of qualification, names associated with a level indicator are the most significant, then those names associated with level-number 01, then names associated with level-number 02, ... , 49. A section-name is the highest (and the only) qualifier available for a paragraph-name. Thus, the most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as procedure-names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and procedure-name.

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

The general formats for qualification are:

Format 1

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

Format 2

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

Format 3

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

The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

2. The same name must not appear at two levels in a hierarchy.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except in the REDEFINES clause where qualification is unnecessary and must not be used).

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same section.

5. A data-name cannot be subscripted when it is being used as a qualifier.

6. A name can be qualified even though it does not need qualifications; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used. The complete set of qualifiers for a data-name must not be the same as any partial set of qualifiers for another data-name.

   Qualified data-names may have any number of qualifiers up to an implementor-defined limit. This limit must be at least five.

7. If more than one COBOL library is available to the compiler during compilation, text-name must be qualified each time it is referenced.

Subscripting

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names (see THE OCCURS CLAUSE in Chapter 4).

The subscript can be represented either by a numeric literal that is an integer or by a data-name. The data-name must be a numeric elementary item that represents an integer.

The subscript may be signed and, if signed, it must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ... The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization.

The format is:

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

Indexing

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY phrase in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name must be initialized before it is used as a table reference. An index-name can be given an initial value by a SET statement.

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the operator + or -, followed by an unsigned integer numeric literal all delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (when the operator - is used), by the value of the literal, the occurrence number represented by the value of the index. When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

At the time of execution of a statement which refers to an indexed table element, the value contained in the index referenced by the index-name associated with the table element must neither correspond to a value less than one nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resulting from relative indexing.

The general format for indexing is:

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

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

Identifier

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

The general formats for identifiers are:

Format 1:

```
[ { OF } data-name-2 ] ...  [ (subscript-1,  subscript-2  [, subscript-3]] ) ]
  { IN }
```

Format 2:

```
[ { OF } data-name-2 ] ...  [ ( { index-name-1 } [ { + } literal-2 ]
  { IN }                       { literal-1    } [ { - }           ]

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

Restrictions on subscripting and indexing are:

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

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

3. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data in a form specified by the implementor. Such data items are called index data items.

4. Literal-1, literal-3, literal-5, in the above format must be positive numeric integers. Literal-2, literal-4, literal-6 must be unsigned numeric integers.

Condition-Name

Each condition-name must be unique, or be made unique through qualification and/or indexing, or subscripting. If qualification is used to make a condition-name unique, the associated conditional variable may be used as the first qualifier. If qualification is used, the hierarchy of names associated with the conditional variable or the conditional variable itself must be used to make the condition-name unique.

If references to a conditional variable require indexing or subscripting, then references to any of its condition-names also require the same combination of indexing or subscripting.

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names is exactly that of 'identifier' except that data-name-1 is replaced by condition-name-1.

In the general formats, 'condition-name' refers to a condition-name qualified, indexed or subscripted, as necessary.

EXPLICIT AND IMPLICIT SPECIFICATIONS

There are three types of explicit and implicit specifications that occur in COBOL source programs:

1. Explicit and implicit Procedure Division references

2. Explicit and implicit transfers of control

3. Explicit and implicit attributes.

Explicit and Implicit Procedure Division References

A COBOL source program can reference data items either explicitly or implicitly in Procedure Division statements. An explicit reference occurs when the name of the referenced item is written in a Procedure Division statement or when the name of the referenced item is copied into the Procedure Division by the processing of a COPY statement. An implicit reference occurs when the item is referenced by a Procedure Division statement without the name of the referenced item being written in the source statement. An implicit reference also occurs, during the execution of a PERFORM statement, when the index or data item referenced by the index-name or identifier specified in the VARYING, AFTER or UNTIL phrase is initialized, modified, or evaluated by the control mechanism associated with that PERFORM statement. Such an implicit reference occurs if and only if the data irem contributes to the execution of the statement.

Explicit and Implicit Transfers of Control

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

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the execution of a procedure branching statement. COBOL provides the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

1. If a paragraph is being executed under control of another COBOL statement (for example, PERFORM, USE, SORT and MERGE) and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. Further, if a paragraph is being executed under the control of a PERFORM statement which causes iterative execution and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and the first statement in that paragraph for each iterative execution of the paragraph.

2. When a SORT or MERGE statement is executed, an implicit transfer of control occurs to any associated input or output procedures.

3. When any COBOL statement is executed which results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Note that another implicit transfer of control occurs after execution of the declarative section, as described in (1) above.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. (See STATEMENTS AND SENTENCES later in this Chapter.) An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The execution of the procedure branching statement ALTER does not in itself constitute an explicit transfer of control, but affects the explicit transfer of control that occurs when the associated GO TO statement is executed. The procedure branching statement EXIT PROGRAM causes an explicit transfer of control when the statement is executed in a called program.

In this document, the term 'next executable statement' is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules associated with each language element in the Procedure Division.

There is no next executable statement following:

1. The last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other COBOL statement.

2. The last statement in a program when the paragraph in which it appears is not being executed under the control of some other COBOL statement.

Explicit and Implicit Attributes

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

For example, the usage of a data item need not be specified, in which case a data item's usage is DISPLAY.

PROGRAM STRUCTURE

An Apple III COBOL program consists of four divisions:

1. IDENTIFICATION DIVISION - An identification of the program.

2. ENVIRONMENT DIVISION - A description of the equipment to be used to compile and run the program.

3. DATA DIVISION - A description of the data to be processed.

4. PROCEDURE DIVISION - A set of procedures to specify the operations to be performed on the data.

Each division is divided into sections which are further divided into paragraphs, which in turn are made up of sentences.

THE "ANSI SWITCH" COMPILER DIRECTIVE

Some of the 'red-tape' statements required by a strict ANSI interpretation are optional under Apple III COBOL. It is possible to force the compiler to insist on a strict ANSI interpretation by using the "FLAG" directive. In the remainder of this Chapter these statements are marked ǂ ǂ. Elsewhere in this manual a reference is made to the ANSI switch when this applies.

If the operator enters the FLAG directive at compile time, all Apple III COBOL extensions are flagged on the compiler listing, together with ANSI COBOL requirements depending on their level as specified by the Federal Compiler Testing Center under the direction of the General Services Administration (GSA). See the description of the Compiler FLAG directive in the Apple III COBOL Introduction and Operating System Manual.

IDENTIFICATION DIVISION

GENERAL DESCRIPTION

The Identification Division must be included in every ANSI COBOL source program. This division identifies both the source program and the resultant output listing. In addition, the user may include the date the program is written, the date the compilation of the source program is accomplished and such other information as desired under the paragraphs in the general format shown below.

ORGANIZATION

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in order of presentation shown by the format below.

STRUCTURE

The following is the general format of the paragraphs in the Identification Division and it defines the order of presentation in the source program.

General Format

```
┼IDENTIFICATION DIVISION.┼

┼PROGRAM-ID.   program-name.┼

[AUTHOR.  [comment-entry]   ...]

[INSTALLATION.     [comment-entry]     ...]

[DATE-WRITTEN.     [comment-entry]     ...]

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

[SECURITY.      [comment-entry]     ...]
```

ENVIRONMENT DIVISION

GENERAL DESCRIPTION

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the compiling computer and the object computer. In addition, information relating to input-output control, special hardware characteristics and control techniques can be given.

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

ORGANIZATION

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

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

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

STRUCTURE

The following is the general format of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program.

General Format

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

DATA DIVISION

OVERALL APPROACH

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

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

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

3. Constants which are defined by the user.

PHYSICAL AND LOGICAL ASPECTS OF DATA DESCRIPTION

Data Division Organization

The DATA DIVISION which is one of the required divisions in a program, is subdivided into sections. These are the File, Working-Storage and Linkage sections.

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions, or by a file description entry and one or more report description entries. Record descriptions are written immediately following the file description entry. The WORKING-STORAGE SECTION describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values are assigned in the source program and do not change during the execution of the object program. The LINKAGE SECTION appears in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the WORKING-STORAGE SECTION.

General Format

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

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

PROCEDURE DIVISION

GENERAL DESCRIPTION

The Procedure Division must be included in every COBOL source program. This division may contain declarative procedures.

Declaratives

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES. (See descriptions of the USE statement in Chapters 5, 6 and 7 and the Debug Chapter 10).

Procedures

A procedure is composed of a paragraph, or group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name (which may be qualified), or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear.

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

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

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term 'identifier' is defined as the word or words necessary to make unique reference to a data item.

Execution

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

General Format

Procedure Division Header

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

```
PROCEDURE DIVISION   [USING data-name-1   [, data-name-2]   ...   ].
```

Procedure Division Body

The body of the Procedure Division must conform to one of the following formats:

Format 1:

```
DECLARATIVES.

{section-name SECTION  [segment-number].   declarative-sentence

 [paragraph-name.   [sentence]   ...]   ... } ...

END DECLARATIVES.

{section-name SECTION   [segment-number] .

 [paragraph-name.   [sentence]   ...]   ... } ...
```

Format 2:

```
{paragraph-name.   [sentence]   ... } ...
```

STATEMENTS AND SENTENCES

There are three types of statements:

1. Conditional statements,
2. Compiler directing statements,
3. Imperative statements.

There are three types of sentences:

1. Conditional sentences,
2. Compiler directing sentences,
3. Imperative sentences.

Conditional Statement

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

A conditional statement is one of the following:

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

Conditional Sentence

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

Compiler Directing Statement

A compiler directing statement consists of a compiler directing verb and its operands. The compiler directing verbs are COPY, ENTER and USE (see THE COPY STATEMENT in Chapter 9, THE ENTER STATEMENT in Chapter 3, and THE USE STATEMENT in Chapters 5, 6 and 7). A compiler directing statement causes the compiler to take a specified action during compilation.

Compiler Directing Sentence

A compiler directing sentence is a single compiler directing statement terminated by a period followed by a space.

Imperative Statement

An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement, nor a compiler directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator.

ACCEPT	EXIT	SET
ADD[1]	GO	SORT
ALTER	INSPECT	START[2]
CALL[3]	MERGE	STOP
CANCEL	MOVE	STRING
CLOSE	MULTIPLY[1]	SUBTRACT[1]
COMPUTE	OPEN	UNSTRING
DELETE[2]	PERFORM	WRITE[5]
DISPLAY	READ[4]	
DIVIDE[1]	REWRITE[2]	

1 - Without the optional SIZE ERROR phrase.
2 - Without the optional INVALID KEY phrase.
3 - Without the optional ON OVERFLOW phrase.
4 - Without the optional AT END phrase or INVALID KEY phrase.
5 - Without the optional INVALID KEY phrase or END-OF-PAGE phrase.

When 'imperative-statement' appears in the general format of statements, 'imperative-statement' refers to that sequence of consecutive imperative statements that must be ended by a period or an ELSE phrase associated with a previous IF statement.

Imperative Sentence

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

REFERENCE FORMAT

GENERAL DESCRIPTION

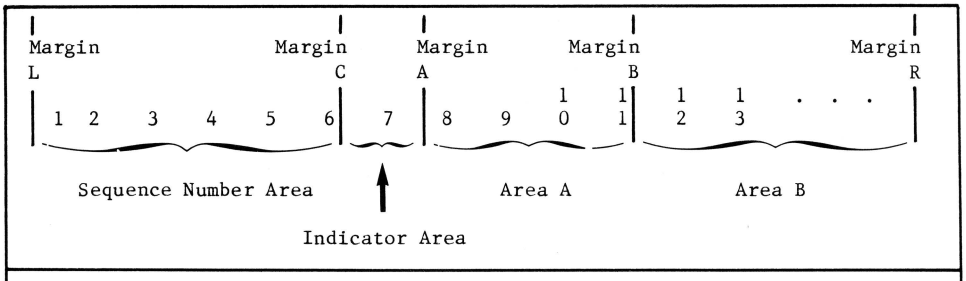
The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions in a line on an input-output medium. The Apple III COBOL compiler accepts source programs written in reference format and produces an output listing of the source program input in reference format.

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

The divisions of a source program must be ordered as follows: the Identification Division, then the Environment Division, then the Data Division, then the Procedure Division. Each division must be written according to the rules for the reference format.

REFERENCE FORMAT REPRESENTATION

The reference format for a line is represented as in Figure 2-1.



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

Margin C is between the 6th and 7th character positions of a line.

Margin A is between the 7th and 8th character positions of a line.

Margin B is between the 11th and 12th character positions of a line.

Margin R is immediately to the right of the rightmost character position of a line.

Figure 2-1. Reference Format for a COBOL Source Line.

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

The indicator area is the 7th character position of a line.

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

Area B occupies character positions 12 through 72 inclusive; it begins immediately to the right of Margin B and terminates immediately to the left of Margin R.

Sequence Numbers

A sequence number, consisting of six digits in the sequence area, may be used to label a source program line.

Continuation of Lines

Whenever a sentence, entry, phrase, or clause requires more than one line, it may be continued by starting subsequent line(s) in area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line.

A hyphen in the indicator area of a line indicates that the first nonblank character in area B of the current line is the successor of the last nonblank character of the preceding line without any intervening space. However, if the continued line contains a nonnumeric literal without closing quotation mark, the first nonblank character in area B on the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of a continuation line must be blank.

If there is no hyphen in the indicator area of a line, it is assumed that the last character in the preceding line is followed by a space.

Blank Lines

A blank line is one that is blank from margin C to margin R, inclusive. A blank line can appear anywhere in the source program, except immediately preceding a continuation line. (See Figure 2-1.)

DIVISION, SECTION, PARAGRAPH FORMATS

Division Header

The division header must start in area A. (See Figure 2-1.)

Section Header

The section header must start in area A. (See Figure 2-1.)

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

Paragraph Header, Paragraph-Name and Paragraph

A paragraph consists of a paragraph-name followed by a period and a space and by zero, one, or more sentences, or a paragraph header followed by one or more entries. Comment entries may be included within a paragraph. The paragraph header or paragraph-name starts in area A of any line following the first line of a division or a section.

The first sentence or entry in a paragraph begins either on the same line as the paragraph header or paragraph-name or in area B of the next nonblank line that is not a comment line. Successive sentences or entries either begin in area B of the same line as the preceding sentence or entry or in area B of the next nonblank line that is not a comment line.

When the sentences or entries of a paragraph require more than one line they may be continued as described in CONTINUATION OF LINES in this Chapter.

DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or a level-number, followed by a space, followed by its associated name (except in the Report Section), followed by a sequence of independent descriptive clauses. Each clause, except the last clause of an entry, may be terminated by either the separator semicolon or the separator comma. The last clause is always terminated by a period followed by a space.

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

A level indicator is any of the following: FD (see THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON in Chapters 5, 6 and 7), SD (see the SORT MERGE FILE DESCRIPTION - COMPLETE ENTRY SKELETON in Chapter 8).

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

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

A level-number has a value taken from the set of values 1 through 49, 66, 77 and 88. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. At least one space must separate a level-number from the word following the level-number.

In those data description entries that begin with level-number 01 or 66, 77 and 88, the level-number begins in area A followed by a space and followed in area B by its associated record-name or item-name and appropriate descriptive information.

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

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

DECLARATIVES

The key word DECLARATIVES and the key words END DECLARATIVES that precede and follow, respectively, the declaratives portion of the Procedure Division must each appear on a line by themselves. Each must begin in area A and be followed by a period and a space (see Figure 2-1.)

COMMENT LINES

A comment line is any line with an asterisk in the continuation indicator area of the line. A comment line can appear as any line in a source program after the Identification Division header. Any combination of characters from the computer's character set may be included in area A and area B of that line (see Figure 2-1). The asterisk and the characters in area A and area B will be produced on the listing but serve as documentation only. A special form of comment line represented by a stroke in the indicator area of the line causes page ejection prior to printing the comment.

Successive comment lines are allowed. Continuation of comment lines is permitted, except that each continuation line must contain an '*' in the indicator area.

RESERVED WORDS

A full list of reserved words is given in Appendix A.

Chapter 3

The Nucleus

FUNCTION OF THE NUCLEUS

The Nucleus provides a basic language capability for the internal processing of data within the basic structure of the four divisions of a program.

OVERALL LANGUAGE

NAME CHARACTERISTICS

Apple III COBOL data-names need not begin with an alphabetic character; the alphabetic characters may be positioned anywhere within the data-name. Qualification is permitted and all data-names, condition-names, paragraph-names, and text-names need not be unique.

FIGURATIVE CONSTANTS

All the following figurative constants may be used: ZERO, ZEROS, ZEROES, SPACE, SPACES HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, QUOTE, QUOTES, and ALL literal.

REFERENCE FORMAT

A word or numeric literal can be broken in such a way that part of it appears on a continuation line.

IDENTIFICATION DIVISION IN THE NUCLEUS

GENERAL DESCRIPTION

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

ORGANIZATION

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included in this division at the user's choice, in the order of presentation shown by the general format below.

Structure

The general format of the paragraphs in the Identification Division is given below. Paragraphs can be in any order.

General Format

```
IDENTIFICATION DIVISION

PROGRAM-ID. program-name.

  AUTHOR.  [comment-entry]  ...

  INSTALLATION.  [comment-entry]  ...

  DATE-WRITTEN.  [comment-entry]  ...

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

  SECURITY.  [comment-entry]  ...
```

Syntax Rules

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

2. The comment-entry may be any combination of the characters from the computer's character set and may be written in Area B on one or more lines. The continuation of the comment-entry by the use of the hyphen in the indicator area is not permitted.

THE PROGRAM-ID PARAGRAPH

Function

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

General Format

PROGRAM-ID. program-name.

Syntax Rules

1. The program-name must conform to the rules for formation of a user-defined word.

General Rules

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

2. The program-name identifies the source program and all listings pertaining to a particular program.

THE DATE-COMPILED PARAGRAPH

Function

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

General Format

DATE-COMPILED. comment-entry ...

Syntax Rule

The comment-entry may be any combination of the characters from the computer's character set. The continuation of the comment entry by use of the hyphen is not permitted; however, the comment entry may be contained on one or more lines.

General Rule

The paragraph-name DATE-COMPILED causes a date entry string to be inserted during program compilation. If a DATE-COMPILED paragraph is present, the comment-entry is replaced in its entirety by the date string. See the Apple III COBOL Introduction and Operating System Manual for details of the derivation of the comment-entry replacement string for your COBOL implementation at compile-time.

ENVIRONMENT DIVISION IN THE NUCLEUS

CONFIGURATION SECTION

The SOURCE-COMPUTER Paragraph

Function

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

General Format

```
SOURCE COMPUTER.    computer-name.
```

Syntax Rule

Computer-name must be one COBOL word defined by the user.

General Rules

The computer-name provides a means for identifying equipment configuration, in which case the computer-name and its implied configuration are specified by the user.

The OBJECT-COMPUTER Paragraph

Function

The OBJECT-COMPUTER Paragraph identifies the computer on which the program is to be executed.

General Format

```
OBJECT-COMPUTER.    computer-name , MEMORY SIZE integer { WORDS      }
                                                        { CHARACTERS }
                                                        { MODULES    }
    [,PROGRAM COLLATING SEQUENCE IS alphabet-name]
```

Syntax Rule

1. Computer-name must be one COBOL word defined by the user.

General Rules

1. The computer-name provides a means for identifying equipment configuration, in which case the computer-name and its implied configurations are specified by the user. The configuration definition contains specific information concerning the memory size.

2. If the PROGRAM COLLATING SEQUENCE Clause is specified, the collating sequence associated with alphabet-name is used to determine the truth value of any nonnumeric comparisons:

   a. Explicitly specified in relation conditions (see Relation Condition later in this Chapter).

   b. Explicitly specified in condition-name conditions; see Condition Name Condition (Conditional Variable).

3. If the PROGRAM COLLATING SEQUENCE Clause is not specified, the native collating sequence is used. Appendix B lists the full ASCII collating sequence (native) and those characters used in COBOL.

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

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

The SPECIAL-NAMES Paragraph

Function

The SPECIAL-NAMES paragraph provides a means of relating implementor-names to user-specified mnemonic-names and of relating alphabet-names to character sets and/or collating sequences.

General Format

```
SPECIAL-NAMES.

[function-name-1 IS mnemonic-name-1]
     [function-name-2 IS mnemonic-name-2] ...

[ SWITCH {0 .  :  7} IS mnemonic-name [,ON STATUS IS condition-name-1
                                       [,OFF STATUS IS condition-name-2]] ]

, {SYSIN  } IS mnemonic-name-1
  {SYSOUT }

, TAB IS mnemonic-name-2

[ , alphabet-name IS { {STANDARD-1} ...
                     { {NATIVE   }
                     {                {THROUGH} literal-2
                     {  literal-1     {THRU   }
                     {              ALSO literal-3 [, ALSO literal-4] ...
                     { [                                                ]
                     { [ literal-5 [ {THROUGH} literal-6                ]...
                     { [           {THRU   }
                     { [           ALSO literal-7 [, ALSO literal-8] ...]
                     { [                                                ]
                     { [                                                ]
  [, CURRENCY SIGN IS literal-9]

  [, DECIMAL-POINT IS COMMA]

  [, CONSOLE IS CRT]

  [, CURSOR is data-name-1]
```

Syntax Rules

1. Mnemonic-names can be any COBOL user-defined word and at least one constituent character must be alphabetic.

2. The literals specified in the literal phrase of the alphabet-name clause:

   a. If numeric, must be unsigned integers and must have a value within the range of one (1) through the maximum number of characters in the native character set.

   b. If nonnumeric and associated with a THROUGH or ALSO phrase, must each be one character in length.

3. If the literal phrase of the alphabet-name clause is specified, a given character must not be specified more than once in an alphabet-name clause.

4. The words THRU and THROUGH are equivalent.

General Rules

1. Function-name-1 specifies system devices or functions used by the compiler. The programmer can associate any user-defined COBOL word with a function-name. Mnemonic-name-1, -2, etc can be used in the ACCEPT, DISPLAY or WRITE statements. Function-name-1, -2, etc. can be one of the following:

   SYSIN - System logical input unit: the CRT keyboard
   SYSOUT - System logical output unit: the CRT screen
   FORMFEED - Usually skip to head of form (WRITE ADVANCING)
   TAB - Skip to next vertical paper movement stop.

   Note that the FORM and TAB functions correspond to the ASCII characters ØC and ØD as sent to the printer. On most printers ØC moves the paper to the Top of Form position and the effect of ØB depends on the individual make of printer. Usually a specified number of lines are skipped.

2. The SWITCH clause enables a mnemonic name to be set to one of two condition-names depending on the setting of a switch at run time by the operator.

3. The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause (see THE OBJECT-COMPUTER PARAGRAPH in this Chapter) or the COLLATING SEQUENCE phrase of a SORT or MERGE statement (see THE SORT STATEMENT or THE MERGE STATEMENT in Chapter 8), the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry (see The File Description - Complete Entry Skeleton in Chapter 5), the alphabet-name clause specifies a character code set.
   a. If the STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in American Standard Code for Information Interchange, X3.4-1968. Appendix B defines the correspondence between the characters of the standard character set and the characters of the native character set.
   b. If the NATIVE phrase is specified, the native character code set or native collating sequence is used. The native collating sequence is as in ANSI publication X3.4-1968 (see Appendix B).

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

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

6. The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters.

   * digits 0 thru 9;

   * alphabetic characters A, B, C, D, L, P, R, S, V, X, Z, or the space;

   * special characters '*', '+', '-', ',', '.', ';', '(', ')', '"', '/'or '='.

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

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

8. The clause CONSOLE IS changes the defaults in the ACCEPT and DISPLAY statements to the Apple III COBOL interactive extension that enables data to be accepted or displayed at any specified point on the screen. See THE ACCEPT STATEMENT and the DISPLAY STATEMENT in this Chapter.

9. The clause CURSOR IS specifies the data-name to contain the CRT cursor address as used by the ACCEPT statement. If CURSOR IS is not specified the default cursor position on executing an ACCEPT statement is the 'Home' position at top left of the CRT screen. The CURSOR IS clause enables a program to retain the position at the end of execution of the last ACCEPT statement or to specify the initial position at the start of any ACCEPT statement. This is a useful facility when programming menu-type operator prompts. The operator need then only move the cursor to the selected option prompt and press RETURN or just press RETURN for the default option.

   Data-name contains the name of a PIC 9999 field in which the most significant 99 represents a line count in the range one to the maximum number of lines on the user screen, and the least significant 99 represents a character position in the range one to the maximum positions allowed by the width of the user screen. If data-name is zero, the effect is as if the CURSOR IS clause was not used, i.e., initial cursor position is top left of screen. (See also THE ACCEPT STATEMENT later in this Chapter).

DATA DIVISION IN THE NUCLEUS

WORKING STORAGE SECTION

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

Noncontiguous Working-Storage

Items and constants in Working-Storage which bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined in a separate data description entry which begins with the special level-number, 77.

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

* Level-number 77
* Data-name
* The PICTURE clause or the USAGE IS INDEX clause

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

Working-Storage Records

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

Initial Values

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

THE DATA DESCRIPTION - COMPLETE ENTRY SKELETON

Function

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

General Format

Format 1:

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

    [; REDEFINES data-name-2]
    [; { PICTURE } IS character-string]
       { PIC     }

    [; [USAGE IS] { COMPUTATIONAL   }]
                  { COMP            }
                  { COMPUTATIONAL-3 }
                  { COMP-3          }
                  { DISPLAY         }

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

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

    [; BLANK WHEN ZERO]

    [; VALUE IS literal]
```

Format 2:

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

Format 3:

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

Syntax Rules

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

2. The clauses may be written in any order with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

3. The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

4. The words THRU and THROUGH are equivalent.

General Rules

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

2. Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:

   a. Another condition-name.

   b. A level 66 item.

   c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

   d. An index data item (see The USAGE IS INDEX Clause in Chapter 4).

THE BLANK WHEN ZERO CLAUSE

Function

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

General Format

```
BLANK WHEN ZERO
```

Syntax Rule

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited. (See THE PICTURE CLAUSE later in this Chapter.)

General Rules

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

2. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

THE DATA-NAME OR FILLER CLAUSE

Function

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

General Format

```
{data-name}
{FILLER   }
```

Syntax Rule

In the File, Working-Storage, Communication and Linkage Sections, a data-name or the key word FILLER must be the first word following the level-number in each data description entry.

General Rule

The key word FILLER may be used to name an elementary item or group in a record. Under no circumstances can a FILLER item be referred to explicitly. However, the key word FILLER may be used as a conditional variable because such use does not require explicit reference to the FILLER item itself, but only to its value.

THE JUSTIFIED CLAUSE

Function

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

General Format

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

Syntax Rules

1. The JUSTIFIED clause can be specified only at the elementary item level.

2. JUST is an abbreviation for JUSTIFIED.

3. The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.

General Rules

1. When a receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill for the leftmost character positions.

2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. (See Standard Alignment Rules.)

LEVEL NUMBER

Function

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working storage items, linkage items, condition-names, and the RENAMES clause.

General Format

level-number

Syntax Rules

1. A level-number is required as the first element in each data description entry.

2. Data description entries subordinate to an FD, CD, or SD entry must have level-numbers with the values 01-49, 66 or 88. (See THE FILE DESCRIPTION in Chapter 5).

3. Data description entries in the Working-Storage Section and Linkage Section must have level-numbers with the values 01-49, 66, 77, or 88.

General Rules

1. The level-number 01 identifies the first entry in each record description or a report group.

2. Special level numbers have been assigned to certain entries where there is no real concept of level:

   a. The level-number 77 is assigned to identify noncontiguous working storage data items, noncontiguous linkage data items, and can be used only as described by Format 1 of the data description skeleton. (See THE DATA DESCRIPTION - COMPLETE ENTRY SKELETON in this Chapter).

   b. Level number 66 is assigned to identify RENAMES entries and can be used only as described in Format 2 of the data description skeleton earlier in this Chapter.

   c. Level number 88 is assigned to entries which define condition-names associated with a conditional variable and can be used only as described in Format 3 of the data description skeleton earlier in this Chapter.

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

THE PICTURE CLAUSE

Function

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

General Format

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

Syntax Rules

1. A PICTURE clause can be specified only at the elementary item level.

2. A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.

3. The maximum number of characters allowed in the character-string is 30.

4. The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

5. PIC is an abbreviation for PICTURE.

6. The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.

General Rules

There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited. General rules within these categories are given below:

Alphabetic Data Rules

1. Its PICTURE character-string can only contain the symbols 'A', 'B'; and

2. Its contents when represented in standard data format must be any combination of the twenty-six (26) letters of the Roman alphabet and the space from the COBOL character set.

Numeric Data Rules

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

2. If unsigned, the data in standard data format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign. (see THE SIGN CLAUSE later in this Chapter).

Alphanumeric Data Rules

1. The PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', '9', and the item is treated as if the character-string contained all X's. A PICTURE character-string which contains all A's or all 9's does not define an alphanumeric item; and

2. The contents when represented in standard data format can consist of any characters in the computer's character set.

Alphanumeric Edited Data Rules

1. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0', and '/' as follows:

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

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

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

Numeric Edited Data Rules

1. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', '/', 'P', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules as follows:

   a. The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive.

   b. The character-string must contain at least one '0', 'B', '/', 'Z', '*', '+', ',', '.', '-', 'CR', 'DB', or currency symbol.

2. The contents of the character positions of these symbols that are allowed to represent a digit in standard data format, must be one of the numerals.

Elementary Item Size

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

Symbols Used

The functions of the symbols used to describe an elementary item are explained as follows:

A - Each 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

B - Each 'B' in the character-string represents a character position into which the space character will be inserted.

P - Each 'P' indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items. The scaling position character 'P' can appear only to the left or right as a continuous string of 'P's within a PICTURE description; since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE characters and to the right if 'P's are rightmost PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost character within such a PICTURE description. The character 'P' and the insertion character '.' (period) cannot both occur in the same PICTURE character-string. If, in any operation involving conversion of data from one form of internal representation to another, the data item being converted is described with the PICTURE character 'P', each digit position described by a 'P' is considered to contain the value zero, and the size of the data item is considered to include the digit positions so described.

S - The letter 'S' is used in a character-string to indicate the presence, but neither the representation nor, necessarily, the position of an operational sign; it must be written as the leftmost character in the PICTURE. The 'S' is not counted in determining the size (in terms of standard data format characters) of the elementary item unless the entry is subject to a SIGN clause which specifies the optional SEPARATE CHARACTER phrase. (See the SIGN Clause in this Chapter.)

V - The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and therefore is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string the 'V' is redundant.

X - Each 'X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set.

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

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

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

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

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

. - When the character '.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause. The insertion character '.' must not be the last character in the PICTURE character-string.

+, -, CR, DB - These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

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

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

Editing Rules

There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available. They are:

* Simple insertion
* Special insertion
* Fixed insertion
* Floating insertion

There are two types of suppression and replacement editing:

* Zero suppression and replacement with spaces
* Zero suppression and replacement with asterisks

The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. Table 3-1 specifies which type of editing may be performed upon a given category.

Table 3-1. Editing Types for Data Categories

CATEGORY	TYPE OF EDITING
Alphabetic	Simple insertion 'B' only
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion '0', 'B' and '/'
Numeric Edited	All, but see NOTE below

NOTE:

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

Simple Insertion Editing

Simple Insertion Editing. The ',' (comma), 'B' (space), '0' (zero), and '/' (stroke) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

Special Insertion Editing

Special Insertion Editing. The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

Fixed Insertion Editing

Fixed Insertion Editing. The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item. The symbol '+' or '-', when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character

Table 3-2. Editing Symbols in PICTURE Character-Strings.

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE OR ZERO	DATA ITEM NEGATIVE
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Floating Insertion Editing

The currency symbol and editing sign control symbols '+' or '-' are the floating insertion characters and as such are mutually exclusive in a given PICTURE character-string.

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

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data item.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Non-zero numeric data may replace all the characters at or to the right of this limit.

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

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character will be placed into the character position immediately preceding either the decimal point or the first non-zero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of non-floating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

Zero Suppression Editing

The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used, the replacement character will be the space and if the asterisk is used, the replacement character will be '*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

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

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is 'Z', the entire data item will be spaces. If the value is zero and the suppression symbol is '*', the data item will be all '*' except for the actual decimal point.

The symbols '+', '-', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Precedence Rules

Table 3-3 shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol 'cs'.

At least one of the symbols 'A', 'X', 'Z', '9' or '*', or at least two of the symbols '+', '-' or 'cs' must be present in a PICTURE string.

Table 3-3. PICTURE Character Precedence Chart.

First Symbol / Second Symbol		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ −}	{+ −}	{CR DB}	cs	{Z *}	{Z *}	{+ −}	{+ −}	cs	cs	9	A X	S	V	P	P
Non-Floating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	.	x	x	x	x		x			x	x		x		x		x					
	{+ −}																					
	{+ −}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	cs						x															
Floating Insertion Symbols	{Z *}	x	x	x	x		x			x	x											
	{Z *}	x	x	x	x	x	x			x	x	x								x		x
	{+ −}	x	x	x	x					x			x									
	{+ −}	x	x	x	x	x				x			x	x						x		x
	cs	x	x	x	x		x								x							
	cs	x	x	x	x	x	x								x	x				x		x
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x
	A X	x	x	x													x	x				
	S																					
	V	x	x	x	x		x			x	x		x		x		x		x		x	
	P	x	x	x	x		x			x	x		x		x		x		x		x	
	P						x			x									x	x		x

In Table 3-3, non-floating insertion symbols '+' and '-', floating insertion symbols 'Z', '*', '+', '-', and 'cs', and other symbol 'P' appear twice in the PICTURE character precedence chart. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of symbol in the row and column represents its use to the right of the decimal point position.

THE REDEFINES CLAUSE

Function

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

General Format

```
level-number data-name-1; REDEFINES data-name-2
```

NOTE: Level-number, data-name-1 and the semi-colon are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the REDEFINES clause.

Syntax Rules

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

2. The level-numbers of data-name-1 and data-name-2 must be identical but must not be 66 or 88.

3. This clause must not be used in level 01 entries in the File Section. (See General Rule 2 of THE DATA RECORDS CLAUSE in Chapter 5.

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

5. Data-name-2 may be subordinate to an entry which contains a REDEFINES clause. However data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause may not be subscripted or indexed. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS clause. (See THE OCCURS CLAUSE in Chapter 4).

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

General Rules

1. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

2. When the level-number of data-name-1 is other than 01, it must specify the same number of character positions that the data item referenced by data-name-2 contains. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

3. Multiple redefinitions of the same character positions are permitted. The entries giving the new descriptions of the character positions must follow the entries defining the area being redefined, without intervening entries that define new character positions. Multiple redefinitions of the same character positions must all use the data-name of the entry that originally defined the area.

4. The entries giving the new description of the character positions must not contain any VALUE clauses except in condition-name entries.

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

THE RENAMES CLAUSE

Function

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

General Format

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

NOTE: Level-number 66, data-name-1 and the semicolon are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the RENAMES clause.

Syntax Rules

1. All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.

2. Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 entry.

3. Data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the associated level 01, FD, CD or SD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry. (See THE OCCURS CLAUSE in Chapter 4.)

4. The beginning of the area described by data-name-3 must not be to the left of the beginning of the area desribed by data-name-2. The end of the area described by data-name-3 must be the right of the end of the area described by data-name-2. Data-name-3, therefore, cannot be subordinate to data-name-2.

5. Data-name-2 and data-name-3 may be qualified.

6. The words THRU and THROUGH are equivalent.

7. None of the items within the range, including data-name-2 and data-name-3, if specified, can be an item whose size is variable as defined in THE OCCURS CLAUSE in Chapter 4.

General Rules

1. One or more RENAMES entries can be written for a logical record.

2. When data-name-3 is specified, data-name-1 is a group item which includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

3. When data-name-3 is not specified, data-name-2 can be either a group or an elementary item; when data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

THE SIGN CLAUSE

Function

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

General Format

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

Syntax Rules

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

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

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

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

General Rules

1. The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 'S'; the 'S' indicates the presence of, but neither the representation nor, necessarily, the position of the operational sign.

2. A numeric data description entry whose picture contains the character 'S', but to which no optional SIGN clause applies, has an operational sign, but neither the representation nor, necessarily, the position of the operational sign is specified by the character 'S'. In this (default) case, general rules 3 through 5 do not apply to such signed numeric data items.

3. If the optional SEPARATE CHARACTER phrase is not present, then:

   a. The operational sign will be presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item.

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

4. If the optional SEPARATE CHARACTER phrase is present, then:

   a. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

   b. The letter 'S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

   c. The operational signs for positive and negative are the standard data format characters '+' and '-', respectively.

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

THE SYNCHRONIZED CLAUSE

Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory (see ITEM ALIGNMENT FOR INCREASED OBJECT-CODE EFFICIENCY in Chapter 2).

General Format

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

Syntax Rules

1. This clause may only appear with an elementary item.

2. SYNC is an abbreviation for SYNCHRONIZED.

General Rules

1. The SYNCHRONIZED clause is accepted for documentation purposes only.

2. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries delimiting this data item. If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions (or portions thereof) must not be used for any other data item. Such unused character positions, however, are included in:

   a. The size of any group item(s) to which the elementary item belongs; and

   b. The character positions redefined when this data item is the object of a REDEFINES clause.

3. SYNCHRONIZED not followed by either RIGHT or LEFT specifies that the elementary item is to be positioned between natural boundaries in such a way as to effect efficient utilization of the elementary data item.

4. SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left character position of the natural boundary in which the elementary item is placed.

5. SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right character position of the natural boundary in which the elementary item is placed.

6. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining any action that depends on size, such as justification, truncation or overflow.

7. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

8. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:

   a. Each occurrence of the data item is SYNCHRONIZED.

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

9. This clause is hardware dependent.

THE USAGE CLAUSE

Function

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

General Format

```
[USAGE IS]  { COMPUTATIONAL   }
            { COMP            }
            { DISPLAY         }
            { COMPUTATIONAL-3 }
            { COMP-3          }
```

Syntax Rules

1. The PICTURE character-string of a COMPUTATIONAL or COMPUTATIONAL-3 item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', one or more 'P's. (See THE PICTURE CLAUSE earlier in this Chapter).

2. COMP is an abbreviation for COMPUTATIONAL.

General Rules

1. The USAGE clause can be written at any level. If the USAGE clause is written at group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

2. This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

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

4. The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.

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

6. Space requirements for the various USAGE storage options are given under Selection of Character Representation and Radix in Chapter 2.

THE VALUE CLAUSE

Function

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

General Format

Format 1:

```
VALUE is literal
```

Format 2:

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

Syntax Rules

1. The VALUE clause cannot be stated for any items whose size is variable. (See THE OCCURS CLAUSE in Chapter 4).

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

3. All numeric literals in a VALUE clause of an item must have values which are within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

4. The words THRU and THROUGH are equivalent.

General Rules

1. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:

   a. If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working storage item, the literal is aligned in the data item according to the standard alignment rules. (See Standard Alignment Rules in Chapter 2).

b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. (See STANDARD ALIGNMENT RULES in Chapter 2). Editing characters in the PICTURE clause are included in determining the size of the data item (see THE PICTURE CLAUSE earlier in this Chapter) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.

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

2. A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

Condition-name Rules

1. In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

2. Format 2 can be used only in connection with condition-names. Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

Data Description Entries other than Condition-Names

Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

1. In the File Section, the VALUE clause may be used only in condition-name entries.

2. In the Working-Storage Section, the VALUE clause must be used in condition-name entries. The VALUE clause may also be used to specify the initial value of a data item; in which case the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item's description, the initial value is undefined.

3. In the Linkage Section, the VALUE clause may be used only in condition-name entries.

4. The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, but not in an entry that is subordinate to an entry containing an OCCURS clause. (See THE OCCURS CLAUSE in Chapter 4).

5. The VALUE clause may be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause.

6. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

7. The VALUE clause must not be written for a group containing items with descriptions, including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

PROCEDURE DIVISION IN THE NUCLEUS

ARITHMETIC EXPRESSIONS

Definition of an Arithmetic Expression

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operator and parentheses are given in Table 3-4, Combination of Symbols in Arithmetic Expressions.

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

Arithmetic Operators

There are five binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that may be preceded by a space and followed by a space.

Binary Arithmetic Operators	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operators	Meaning
+	The effect of multiplication by numeric literal +1
-	The effect of multiplication by numeric literal -1.

Formation and Evaluation Rules

1. Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When

parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

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

2. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in Table 3-4, where:

   a. The letter 'P' indicates a permissible pair of symbols.

   b. The character '-' indicates an invalid pair.

   c. 'Variable' indicates an identifier or literal.

FIRST SYMBOL	SECOND SYMBOL			
	Variable	* / ** - +	Unary + or -	( )
Variable	-	P	-	- P
* / ** + -	P	-	P	P -
Unary + or -	P	-	-	P -
(	P	-	P	P -
)	-	P	-	- P

Table 3-4. Combination of Symbols in Arithmetic Expressions

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

5. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items. See, for example, syntax rule 3 of THE ADD STATEMENT in this Chapter.

CONDITIONAL EXPRESSIONS

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

Simple Conditions

The simple conditions are the relation, class, condition-name, switch-status, and sign conditions. A simple condition has a truth value of 'true' or 'false'. The inclusion in parentheses of simple conditions does not change the simple truth value.

Relation Condition

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

The general format of a relation condition is as follows:

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

NOTE: The required relational characters '<', '>', and '=' are not underlined to avoid confusion with other symbols such as '≥' (Greater than or equal to)

The first operand (identifier-1, literal-1 or arithmetic-expression-1) is called the subject of the condition; the second operand (identifier-2 or literal-2 or arithmetic-expression-2) is called the object of the condition. The relation condition must contain at least one reference to a variable.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, 'NOT' and the next key word or relation character are one relational operator that defines the comparison to be executed for truth value; e.g., 'NOT EQUAL' is a truth test for an 'unequal'.

Comparison 'NOT GREATER' is a truth test for an 'equal' or 'less' comparison. The meaning of the relational operators is as shown in Table 3-5.

Table 3-5. Relational Operators.

Meaning	Relational Operator
Greater than or not greater than	IS [NOT] GREATER THAN IS [NOT] >
Less than or not less than	IS [NOT] LESS THAN IS [NOT] <
Equal to or not equal to	IS [NOT] EQUAL TO IS [NOT] =
The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).	

Comparison of Numeric Operands: For operands whose class is numeric a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic expression operands, in terms of number of digits represented, is not significant. Zero is considered a unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

Comparison of Nonnumeric Operands: For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters (see The OBJECT-COMPUTER Paragraph in this Chapter). If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data

item (in terms of standard data format characters), and the contents of this alphanumeric data item were then compared to the nonnumeric operand. (See THE MOVE STATEMENT in this Chapter, and the PICTURE Character 'P' under the heading Symbols Used earlier in this Chapter.)

2. If the non-numeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item were then compared to the nonnumeric operand. (See THE MOVE STATEMENT in this Chapter, and the PICTURE character 'P' under the heading Symbols Used under THE PICTURE CLAUSE earlier in this Chapter.)

3. A non-integer numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same.

There are two cases to consider:

1. Operands of equal size - If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

   The first encountered pair of unequal characters is compared to determine their relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

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

Class Condition

The class condition determines whether the operand is numeric, that is, consists entirely of the characters '0', '1', '2', '3', ..., '9', with or without the operational sign; or alphabetic, that is, consists entirely of the characters 'A', 'B', 'C', ..., 'Z', space. The general format for the class condition is as follows:

identifier IS [NOT] { NUMERIC / ALPHABETIC }

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

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present. Valid operational signs for data items described with the SIGN IS SEPARATE clause are the standard data format characters, '+' and '-'.

The ALAPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space.

Condition-Name Condition (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name. The general format for the condition-name condition is as follows:

condition-name

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

Switch-Status Condition

A switch-status condition determines the 'on' or 'off' status of an implementor-defined switch. The implementor-name and the 'on' or 'off' value associated with the condition must be named in the SPECIAL-NAMES paragraph of the Environment Division. The general format for the switch-status condition is as follows:

condition-name

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

Sign Condition

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

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

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

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions and/or complex conditions with logical connectors (logical operators 'AND' and 'OR') or negating these conditions with logical negation (the logical operator 'NOT'). The truth value of a complex condition, whether parenthesized or not, is that truth value which results from the interaction of all the stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically connected or logically negated.

The logical operators and their meanings are:

Logical Operator	Meaning
AND	Logical conjunction; the truth value is 'true' if both of the conjoined conditions are true; 'false' if one or both of the conjoined conditions is false.
OR	Logical inclusive OR; the truth value is 'true' if one or both of the included conditions is true; 'false' if both included conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is 'true' if the condition is false; 'false' if the condition is true.

The logical operators must be preceded by a space and followed by a space.

Negated Simple Conditions: A simple condition is negated through the use of the logical operator 'NOT'. The negated simple condition effects the opposite truth value for a simple condition. Thus the truth value of a negated simple condition is 'true' if and only if the truth value of the simple condition is 'false'; the truth value of a negated simple condition is 'false' if and only if the truth value of the simple condition is 'true'. The inclusion in parentheses of a negated simple condition does not change the truth value.

The general format for a negated simple condition is:

NOT simple-condition

Combined and Negated Combined Conditions: A combined condition results from connecting conditions with one of the logical operators 'AND' or 'OR'. The general format of a combined condition is:

condition { { AND / OR } condition } ...

where 'condition' may be:

a. A simple condition, or

b. A negated simple condition, or

c. A combined condition,

d. A negated combined condition; i.e., the 'NOT' logical operator followed by a combined condition enclosed within parentheses, or

e. Combinations of the above, specified according to the rules summarized in Table 3-6, Combinations of Conditions, Logical Operators, and Parentheses, located on the next page.

Although parentheses need never be used when either 'AND' or 'OR' (but not both) is used exclusively in a combined condition, parentheses may be used to effect a final truth value when a mixture of 'AND', 'OR' and 'NOT' is used. (See Table 3-6, Combinations of Conditions, Logical Operators, and Parentheses below, and Condition Evaluation Rules earlier in this Chapter.)

Table 3-6 on the next page indicates the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is matched by a corresponding right parenthesis. The table assumes a left-to-right sequence of elements.

Table 3-6. Combinations of Conditions, Logical Operators, and Parentheses.

Element	Permitted location in conditional expression	Element can be preceded by only:	Element can be followed by only:
simple-condition	Any	OR, NOT, AND, (	OR, AND,)
OR, or AND	Not first or last	simple-condition,)	simple-condition, NOT, (
NOT	Not last	OR, AND, (	simple-condition, (
(	Not last	OR, NOT, AND, (	simple-condition, NOT, (
)	Not first	simple-condition,)	OR, AND,)

Thus, the element pair 'OR NOT' is permissible while the pair 'NOT OR' is not permissible; 'NOT (' is permissible while 'NOT NOT' is not permissible.

Abbreviated Combined Relation Conditions

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by:

1. The omission of the subject of the relation condition, or

2. The omission of the subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:

relation-condition $\left\{\begin{Bmatrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{Bmatrix}\right.$ [$\underline{\text{NOT}}$] [relational-operator] object $\left.\right\}$...

Within a sequence of relation conditions both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must

comply with the rules of Table 3-6, Combinations of Conditions, Logical Operators, and Parentheses. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The interpretation applied to the use of the word 'NOT' in an abbreviated combined relation condition is as follows:

1. If the word immediately following 'NOT' is 'GREATER', '>', 'LESS', '<', 'EQUAL','=', then the 'NOT' participates as part of the relational operator; otherwise

2. The 'NOT' is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Some examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

Abbreviated Combined Relation Condition	Expanded Equivalent
a > b AND NOT < c OR d	((a > b) AND (a NOT < c)) OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b)) OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT ((((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d))))

Condition Evaluation Rules

Parentheses may be used to specify the order in which individual conditions of complex conditions are to be evaluated when it is necessary to depart from the implied evaluation precedence. Conditions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. When parentheses are not used, or parenthesized conditions are at the same level of inclusiveness, the following hierarchical order of logical evaluation is implied until the final truth value is determined:

1. Values are established for arithmetic expressions. (See Formation and Evaluation Rules under ARITHMETIC EXPRESSIONS in this Chapter.)

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

   relation (following the expansion of any abbreviated relation condition)
   class
   condition-name
   switch-status
   sign

3. Truth values for negated conditions are established.

4. Truth values for combined conditions are established:

   'AND' logical operators, followed by
   'OR' logical operators.

5. Truth values for negated combined conditions are established.

6. When the sequence of evaluation is not completely specified by parentheses, the order of evaluation of consecutive operations of the same hierarchical level is from left to right.

COMMON PHRASES AND GENERAL RULES FOR STATEMENT FORMATS

In the statement descriptions that follow, several phrases appear frequently: the ROUNDED phrase, the SIZE ERROR phrase and the CORRESPONDING phrase.

These are described below; a resultant-identifier is that identifier associated with the result of an arithmetic operation.

The ROUNDED Phrase

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one whenever the most significant digit of the the excess is greater than or equal to five.

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

The SIZE ERROR Phrase

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results, except in MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR phrase is specified, as follows:

SIZE ERROR Phrase Not Specified

When a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

SIZE ERROR Phrase Specified

When a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are

unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

For the ADD statement with the CORRESPONDING phrase and the SUBTRACT statement with the CORRESPONDING phrase, if any of the individual operations produces a size error condition, the imperative statement in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

The CORRESPONDING Phrase

In the text that follows d_1 and d_2 must each be identifiers that refer

to group items. A pair of data items (one from d_1 and one from $d_{2)}$ correspond if the following conditions exist:

1. A data item in d_1 and a data item in d_2 are not designated by the key

   word FILLER and have the same data-name and the same qualifiers up to, but not including, d_1 and d_2.

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

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

4. A data item that is subordinate to d_1 or d_2 and contains a REDEFINES,

   RENAMES, OCCURS or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, d_1 and d_2 may have

   REDEFINES or OCCURS clauses or be subordinate to data items with REDFINES or OCCURS clauses.

Arithmetic Statements

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. Common features are as follows:

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment are supplied throughout the calculation.

2. The maximum size of each operand is 18 decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points (see THE ADD STATEMENT, THE DIVIDE STATEMENT, THE MULTIPLY STATEMENT and THE SUBTRACT STATEMENT later in this Chapter) must not contain more than 18 decimal digits.

Overlapping Operands

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

Multiple Results in Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written in the following way:

1. A statement which performs all arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence that the multiple results are listed in.

   The result of the statement

   ```
   ADD a, b, c TO c, d (c), e
   ```

   is equivalent to

   ```
   ADD a, b, c GIVING temp
   ADD temp TO c
   ADD temp TO d (c)
   ADD temp TO e
   ```

   where 'temp' is an intermediate result item provided by the compiler.

Incompatible Data

Except for the class condition (see Class Condition in this Chapter), when the contents of a data item are referenced in the Procedure Division and the contents of that data item are not compatible with the class specified for that data item by its PICTURE clause, then the result of such a reference is undefined.

CRT Devices

The CRT is driven directly by the run time system via a buffer. The COBOL programmer moves data into and out of this buffer by means of ACCEPT and DISPLAY statements. Each ACCEPT or DISPLAY action is relative to the start of the CRT buffer unless POSITION is specified. The syntax is limited to inputting to or outputting from a single data name. The data name may be a group item and several such group items may redefine the same area of storage.

The use of FILLER data items in record descriptions used for input or output to a CRT device is subject to special rules. On output, any FILLER item in a record results in suppression of output for the character positions it defines. On input, any FILLER item suppresses operator keying into the character positions it defines.

THE ACCEPT STATEMENT

Function

The ACCEPT statement causes data keyed at the CRT console to be made available to the program in a specified data item

General Formats

Format 1

```
ACCEPT identifier [ [FROM { mnemonic-name }] ]
                  [       { CONSOLE     }]
```

Format 2

```
ACCEPT data-name-1 [ AT { data-name-2 } ]   FROM CRT
                       { literal-1    }
```

Format 3

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

General Rules

1. Format 1 is the standard ANSI ACCEPT statement

   Format 2 is the extended ACCEPT format.

   The two formats are distinguished by their FROM phrases and the default assumes FROM CONSOLE. A user-defined mnemonic-name can be used if this is associated to a system device in the SPECIAL-NAMES paragraph (see The SPECIAL-NAMES Paragraph earlier in this Chapter). The default can, however, be changed by specifying CONSOLE IS CRT in the SPECIAL-NAMES clause so that FROM CRT becomes the default. This changed default is not shown in the syntax above.

Format 1

2. The ACCEPT statement reads one line of input data from the system consol device. This input data replaces the contents of the data item named by the identifier.

3. The line of input is line-edited according to the operating system rules for line-editing (see Operating Systems User Guide). The line is terminated by pressing the CR (Carriage Return) key or by exceeding 120 characters in length.

4. If the input line is of the same size as the receiving data item, the transferred data is stored in the receiving data item.

5. If the input line is not of the same size as the receiving data item, then:

   a. If the size of the receiving data item exceeds the size of the input line, the transferred data is stored aligned to the left in the receiving data item and the data item is filled with trailing spaces.

   b. If the size of the transferred data exceeds 120 bytes, only the first 120 characters of the input line are stored in the receiving data item. The remaining characters of the input line which do not fit into the receiving data item are ignored.

Format 2

6. The ACCEPT statement causes the transfer of data from the CRT to data-name-1. The contents of data-name-1 are replaced by this data.

7. data-name-1 is taken as a definition of the screen area in which elementary data items correspond to areas on the screen into which the operator can key. FILLER fields correspond to areas on the screen which are inaccessible to the operator.

8. Elementary data items within data-name-1 may be alphanumeric, integer numeric, numeric or edited. Numeric items are treated as two separate integer numeric fields, and edited fields are treated as alphanumeric fields except as described in rule 16.

9. AT data-name-2 or literal-1 defines the position on the screen of the leftmost character of the data. Either form must refer to a PIC 9999 field. The most significant 99 is taken as a line count in the range one to the maximum lines on the user screen. The least significant 99 is taken as a character position in the range one to the maximum positions allowed by the screen width of the user CRT.

10. data-name-1 may refer to a record, group or elementary item, but it may not be subscripted. REDEFINES may be used within data-name-1, in which case the first description of the data is used and subsequent descriptions are ignored. OCCURS and nested OCCURS may also be used with the effect that the repeated data-item is expanded into the full number of times it occurs and one definition is thus automatically repeated for many fields.

11. Immediately upon execution of the ACCEPT statement a cursor is displayed in the CRT location corresponding to the leftmost non-FILLER character position in data-name-1. Alternatively, when CURSOR is specified in the SPECIAL-NAMES paragraph, the cursor displays at the position held in the CURSOR data-name. The CURSOR position is stored

in CURSOR data-name in the same format as the screen position is held in data-name-2. If the data-name-2 has the value SPACE or ZERO, the effect is as if CURSOR was not specified; if a valid screen position is specified that is not within a non-FILLER item, the cursor is positioned to the next non-FILLER character position. The CURSOR data-name holds the last cursor position at the end of execution of an ACCEPT statement.

12. If FROM CRT is not specified, the default is FROM CONSOLE (see rule 1 above).

13. As the operator keys characters, the cursor moves to the right one character position at a time in locations corresponding to data fields. The operator always keys into the current cursor position. At the end of a line the cursor moves down one line and to the leftmost non-FILLER character position.

14. If the data item is integer numeric, only numeric characters (0 - 9) will be accepted into that item. Keying the decimal point character (. or , as specified in the DECIMAL POINT phrase) when accepting a numeric item causes the item to be right-justified and zero-filled from the left.

15. When the cursor location reaches a position corresponding to a FILLER item in a data-name, it immediately skips to the next non-FILLER character position, or if there is no such position remaining in the portion of the CRT specified by the data-name, it remains in its current position.

16. The operator can terminate input by pressing the CR (carriage return) key, at which time control is passed to the next statement after ACCEPT. Before control is passed to the next statement the following takes place:

    a. The numeric value of each numeric-edited data-field is formed internally from only the keyed characters 0 to 9, +, -, . or , and then moved back to the numeric-edited field with the ANSI PICTURE editing applied. The field may thus be different from that shown on the CRT just before the Carriage Return key was pressed.

    b. When CURSOR IS is specified in the SPECIAL-NAMES paragraph, the cursor position when the Carriage Return key is pressed is returned in the data-name specified by the CURSOR IS clause, except when its value at the start of the ACCEPT function causes it to be treated as unspecified.

17. Before keying CR, the operator can reposition the cursor to overwrite data already keyed or to skip character positions by use of the character positioning keys, as shown in Table 3 - 7.

NOTE: The actual key identification and functions shown in this table vary according to the CRT used and the way it is configured (see the Apple III COBOL Introduction and Operating System Manual).

Table 3-7. Cursor Repositioning Keys.

KEY:	FUNCTION:
←	Backs up the cursor one position.
↑	Backs up the cursor to the start of the non-FILLER field prior to the current cursor position.
↓	Moves the cursor on to the start of the next non-FILLER field in advance of the current cursor position.
→	Moves the cursor on one position without overwriting existing contents.
↖	Moves the cursor back to the start of the first non-FILLER field in the CRT area corresponding to data-name-1.

Format 3

18. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier according to the rules of the MOVE statement. DATE, DAY, and TIME are conceptual data items and, therefore, are not described in the COBOL program.

19. DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes shall be, from high to low order (left to right), year of century, month of year, and day of month. DATE, when accessed by a COBOL program, behaves as if it had been described in the COBOL program as an unsigned elementary numeric integer data item six digits in length.

20. DAY is composed of the data elements year of century and day of year. The sequence of the data element codes shall be from high order to low order (left to right) year of century, day of year. Therefore, July 1, 1968 would be expressed as 68183. DAY, when accessed by a COBOL program, behaves as it it had been described in a COBOL program as an unsigned elementary numeric integer data item five digits in length.

21. TIME is composed of the data elements hours, minutes, seconds and hundredths of a second. TIME is based on elapsed time after midnight on a 24-hour clock basis -- thus, 2:41 p.m. would be expressed as 14410000. TIME, when accessed by a COBOL program behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item eight digits in length. The minimum value of TIME is 00000000; the maximum value of TIME is 23595999. If the hardware does not have the facility to provide fractional parts of TIME, the value is converted to the closest decimal approximation.

THE ADD STATEMENT

Function

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

General Format

Format 1

```
ADD {identifier-1} [, identifier-2] ... TO identifier-m [ROUNDED]
    {literal-1   } [, literal-2   ]

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

Format 2

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

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

Format 3

```
ADD {CORRESPONDING} identifier-1 TO identifier-2 [ROUNDED]
    {CORR         }
    [; ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item. In Format 3, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits (see The Arithmetic Statements in this Chapter).

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

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

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

General Rules

1. See THE ROUNDED PHRASE, THE SIZE ERROR PHRASE, THE CORRESPONDING PHRASE, Arithmetic Statements, Overlapping Operands and Multiple Results in Arithmetic Statements in this Chapter.

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

3. If Format 2 is used, the value of the operands preceding the word GIVING are added together, then the sum is stored as the new value of each identifier-m, identifier-n, ..., the resultant identifiers.

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

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

THE ALTER STATEMENT

Function

The ALTER statement modifies a predetermined sequence of operations.

General Format

```
ALTER procedure-name-1 TO  PROCEED TO   procedure-name-2

    [, procedure-name-3 TO  [PROCEED TO] procedure-name-4] ...
```

Syntax Rules

1. Each procedure-name-1, procedure-name-3, ..., is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.

2. Each procedure-name-2, procedure-name-4, ..., is the name of a paragraph or section in the Procedure Division.

General Rules

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ..., so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2, procedure-name-4, ..., respectively. Modified GO TO statements in independent segments may, under some circumstances, be returned to their initial states (see Independent Segments in Chapter 8).

2. A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

   All other uses of the ALTER statement are valid and are performed even if procedure-name-1, procedure-name-3 is in an overlayable fixed segment.

THE COMPUTE STATEMENT

Function

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

General Format

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

Syntax Rule

Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric edited item.

General Rules

1. See The ROUNDED Phrase, The SIZE ERROR Phrase, The Arithmetic Statements, Overlapping Operands and Multiple Results in Arithmetic Statements.

2. An arithmetic expression consisting of a single identifier or literal provides a method of setting the values of identifier-1, identifier-2, etc., equal to the value of the single identifier or literal.

3. If more than one identifier is specified for the result of the operation, that is preceding =, the value of the arithmetic expression is computed, and then this value is stored as the new value of each of identifier-1, identifier-2, etc., in turn.

4. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

THE DISPLAY STATEMENT

Function

The DISPLAY statement causes data to be transferred from specified data items to the CRT screen.

General Formats

Format 1

```
DISPLAY  { identifier-1 }  [ , { identifier-2 } ] ...  { [UPON mnemonic-name }
         {              }  [   {              }  ]     {      CONSOLE      }
         { literal-1     }  [   { literal-2     }  ]
```

Format 2

```
DISPLAY  { data-name-1 }  [ AT { data-name-2 } ]  UPON { CRT       }
         { literal-3   }  [    { literal-4   } ]       { CRT-UNDER }
```

Syntax Rules

1. Each literal may be any figurative constant, except ALL.
2. If the literal is numeric, it must be an unsigned integer.
3. literal-3 must be alphanumeric. literal-4 must be numeric.
4. data-name-1 may refer to a record, group or elementary item, but it must not be subscripted.

General Rules

1. Format 1 is the standard ANSI DISPLAY statement.

   Format 2 is the extended DISPLAY format.

   The two formats are distinguished by their UPON phrases and the default assumes UPON CONSOLE. A user-defined mnemonic-name can be used if this is associated with a system device in the SPECIAL-NAMES paragraph. (See THE SPECIAL-NAMES paragraph in this Chapter). The default can, however, be changed by specifying CONSOLE IS CRT in the SPECIAL-NAMES clause so that UPON CRT becomes the default. This changed default is not shown in the syntax above.

Format 1

2. The DISPLAY statement causes the contents of each operand to be transferred to the CRT in the order listed as one line of output data.

3. The size of the data transfer can be up to 132 bytes.

4. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

5. If the CRT is capable of displaying data of the same size as the data item being output, the data item is transferred.

6. If the CRT is not capable of displaying data of the same size as the data item being transferred, one of the following applies.

   (a) If the size of the data item being displayed exceeds the size of the data that the CRT is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving CRT.

   (b) If the size of the data item that the CRT is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving CRT.

7. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered.

Format 2

8. The DISPLAY statement is used to output data to the CRT in the screen positions specified.

9. data-name-1 is taken as a definition of the screen area into which data items that correspond to areas on the screen are moved. FILLER fields correspond to areas on the screen into which data is not moved.

10. Elementary data items within data-name-1 may be alphanumeric, integer numeric, numeric or edited.

11. AT data-name-2 or literal-4 defines the position on the screen of the leftmost character of the data. Either form must refer to a PIC 9999 field. The most significant 99 is taken as a line count in the range one to the maximum number of lines on the user screen. The least significant 99 is taken as a character position in the range one to the maximum of characters per line on the user screen.

12. data-name-1 may refer to a record, group or elementary item, but it may not be subscripted. REDEFINES may be used, in which case the first description of the data is used and subsequent descriptions are ignored. OCCURS and nested OCCURS may also be used with the effect that the repeated data-item is expanded into the full number of times it occurs and one definition is thus automatically repeated for many fields.

13. DISPLAY SPACE has the effect of clearing the screen at run time (i.e. filling the whole screen with spaces). DISPLAY " " (one space character), however, displays only one space character.

14. The CRT-UNDER phrase causes the elementary items moved to the CRT to be displayed with the underline feature present. This feature is dependent on the CRT hardware functions and is not available on all makes of CRT (see the Apple III COBOL Introduction and Operating System Manual).

THE DIVIDE STATEMENT

Function

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

General Format

Format 1

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

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

Format 2

```
DIVIDE  {identifier 1}  INTO    {identifier-2}
        {literal-1   }          {literal-2   }

    GIVING    identifier-3  [ROUNDED]  [,identifier-4  [ROUNDED]] ...
    [;ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE  {identifier-1}  BY  {identifier-2}
        {literal-1   }      {literal-2   }

    GIVING    identifier-3  [ROUNDED]  [, identifier-4  [ROUNDED]] ...
    [;ON SIZE ERROR imperative-statement]
```

Format 4

```
DIVIDE  {identifier 1}  INTO    {identifier-2}
        {literal-1   }          {literal-2   }

    GIVING    identifier-3  [ROUNDED]
    REMAINDER identifier-4  [; ON SIZE ERROR imperative-statement]
```

Format 5

```
DIVIDE  {identifier 1}  BY  {identifier-2}
        {literal-1   }      {literal-2   }

    GIVING    identifier-3  [ROUNDED]

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

Syntax Rules

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The composite of operands, which is the hypothetical data item resulting from the superimposition of all receiving data items (except the REMAINDER data item) of a given statement aligned on their decimal points, must not contain more than eighteen digits.

General Rules

1. See The ROUNDED Phrase, The SIZE ERROR Phrase, The Arithmetic Statements, Overlapping Operands and Multiple Results in Arithmetic Statements in this Chapter for a description of these functions.

2. When Format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient; similarly for identifier-1 or literal-1 and literal-3, etc.

3. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4 etc.

4. When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4 etc.

5. Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If identifier-3 is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field which contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field which contains the quotient of the DIVIDE statement, truncated rather than rounded.

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

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

   a. If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both identifier-3 and identifier-4 will remain unchanged.

   b. If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 remains unchanged, However, as with other instances of multiple results of arithmetic statements, the user will have to do his own analysis to recognize which situation has actually occurred.

THE ENTER STATEMENT

Function

The ENTER statement provides a means of allowing the use of more than one language in the same program.

General Format

```
ENTER  language-name      [routine-name]
```

Syntax Rule

This statement is treated as if for documentation purposes only.

General Rule

Access to other languages can be achieved by means of CALL.

THE EXIT STATEMENT

Function

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

General Format

```
EXIT
```

Syntax Rules

1. The EXIT statement must appear in a sentence by itself.

2. The EXIT sentence must be the only sentence in the paragraph.

General Rule

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

THE GO TO STATEMENT

Function

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

General Format

Format 1

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

Format-2

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

Syntax Rules

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

2. When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format 1 GO TO statement.

3. A Format 1 GO TO statement, without procedure-name-1, can only appear in a single statement paragraph.

4. If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements within a sentence, it appears as the last statement in that sequence.

General Rules

1. When a GO TO statement, represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.

2. If procedure-name-1 is not specified in Format 1, an ALTER statement, referring to this GO TO statement, must be executed prior to the execution of this GO TO statement.

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

THE IF STATEMENT

Function

The IF statement causes a condition to be evaluated (see CONDITIONAL EXPRESSIONS in this Chapter). The subsequent action of the object program depends on whether the value of the condition is true or false.

General Format

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

Syntax Rules

1. Statement-1 and statement-2 represent either an imperative statement or a conditional statement, and either may be followed by a conditional statement.

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

General Rules

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

   a. If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not contain a procedure branching or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

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

   c. If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching or conditional statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

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

2. Statement-1 and/or statement-2 may contain an IF statement. In this case the IF statement is said to be nested.

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

THE INSPECT STATEMENT

Function

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

General Format

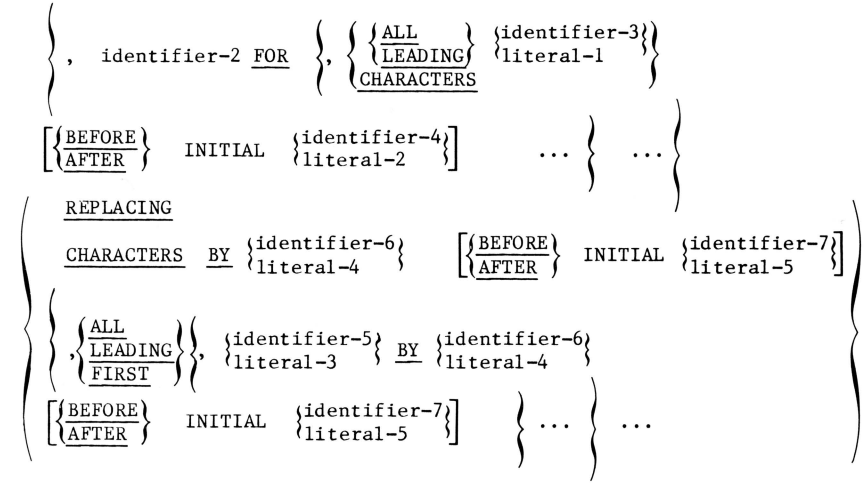
Format 1

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

Format 2

```
INSPECT  identifier-1  REPLACING
{                                                                                  }
{   CHARACTERS  BY  {identifier-6}  [{BEFORE}  INITIAL  {identifier-7}]            }
{                   {literal-4   }  [{AFTER }           {literal-5  }]            }
{   {                                                                              }
{   {,{ALL     }}, {identifier-5}  BY  {identifier-6}                              }
{   {,{LEADING}}   {literal-3   }       {literal-4   }                              }
{   {,{FIRST   }}                                                                  }
{   [{BEFORE}  INITIAL  {identifier-7}]  }  ...  }                                 }
{   [{AFTER }           {literal-5   }]  }       }                                 }
{                                                                                  }
```

Format 3



Syntax Rules

All Formats

1. Identifier-1 must reference either a group item or any category of elementary item, described (either implicitly or explicitly) as USAGE IS DISPLAY.

2. Identifier-3 ... identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (either implicitly or explicitly) as USAGE IS DISPLAY.

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

4. Literal-1, literal-2, literal-3, literal-4, and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, can be any number of characters in length.

Formats 1 and 3 Only

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

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

Formats 2 and 3 Only

7. The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

8. When the CHARACTERS phrase is used, literal-4, literal-5, or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

9. When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

General Rules

All Formats

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 4 through 6.

2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 will be treated as follows:

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

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

   c. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 is described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in general rule 2b had been applied. (See THE MOVE STATEMENT later in this Chapter.)

3. In general rules 4 through 11 all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Formats 2 and 3). Data items to be referenced by the INSPECT verb should be placed such that they lie within the first 10,000 bytes of intermediate code.

5. The comparison operation to determine the occurrences of literal-1 to be tallied, and/or occurrences of literal-3 to be replaced, occurs as follows:

   a. The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1, literal-3 is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal, character for character.

   b. If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

   c. Whenever a match occurs, tallying and/or replacing takes place as described in general rules 8 through 10. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1, literal-3.

   d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

   e. If the CHARACTERS phrase is specified, an implied one-character operand participates in the cycle described in paragraphs 5a through 5d above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in general rule 5 is affected by the BEFORE and AFTER phrases as follows:

   a. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 5.

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

   c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Format 1

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

8. The rules for tallying are as follows:

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

   b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

   c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 are incremented by one for each character matched, in the sense of general rule 5e, within the contents of the data item referenced by identifier-1.

Format 2

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

10. The rules for replacement are as follows:

   a. When the CHARACTERS phrase is specified, each character matched in the sense of general rule 5e in the contents of the data item referenced by identifier-1 is replaced by literal-4.

   b. When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

   c. When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

   d. When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

Format 3

11. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the other statement being a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement.

EXAMPLES

Six examples of the use of the INSPECT statement follow:

```
INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A", count-1 FOR
LEADING "A" BEFORE INITIAL "L".

     Where word = LARGE, count = 1, count-1 = 0.
     Where word = ANALYST, count = 0, count-1 = 1.


INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING "A" BY "E" AFTER
INITIAL "L".

     Where word = CALLAR, count = 2, word = CALLAR.
     Where word = SALAMI, count = 1, word = SALEMI.
     Where word = LATTER, count = 1, word = LETTER.


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

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


INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J" REPLACING ALL
"A" BY "B".

     Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
     Where word = JACK, count = 3, word = JBCK.
     Where word = JUJMAB, count = 5, word = JUJMBB.
```

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

     Where word = RXXBQWY, word = RYYZQQY.
     Where word = YZACDWBR, word = YZACDWZR.
     Where word = RAWRXEB, word = RAQRYEZ.


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

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

THE MOVE STATEMENT

Function

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

General Format

Format 1

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

Format 2

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

Syntax Rules

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3, ..., represent the receiving area.

2. CORR is an abreviation for CORRESPONDING.

3. When the CORRESPONDING phrase is used, both identifiers must be group items.

4. An index data item cannot appear as an operand of a MOVE statement. (See THE USAGE CLAUSE in this Chapter).

General Rules

1. If the CORRESPONDING phrase is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given in The CORRESPONDING Phrase in this Chapter. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

2. The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, ... The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, ..., is evaluated immediately before the data is moved to the respective data item.

   Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the

receiving operands. The result of the statement:

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

is equivalent to:

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

where 'temp' is an intermediate result item provided by the implementor.

3. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

   The following rules apply to an elementary move between these categories:

   a. The figurative constant SPACE, or on alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

   c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

   d. The result of moving a blank numeric-edited field to a numeric field is unpredictable and therefore not permitted.

   e. All other elementary moves are legal and are performed according to the rules given in general rule 4.

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

   a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as defined under STANDARD ALIGNMENT RULES in this Chapter. If the

size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved; if the operational sign occupies a separate character position (see THE SIGN CLAUSE in this Chapter), that character will not be moved and the size of the sending item will be considered to be one less than its actual size (in terms of standard data format characters).

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the STANDARD ALIGNMENT RULES in Chapter 2, except where zeroes are replaced because of editing requirements.

When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. (See THE SIGN CLAUSE in this Chapter). Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the STANDARD ALIGNMENT RULES in Chapter 2. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in general rule 4 of the OCCURS clause.

6. Data in Table 3-8 summarizes the legality of the various types of MOVE statements. The general rule reference indicates the rule that prohibits the move or the behavior of a legal move.

Table 3-8. MOVE Statement Data Categories.

Category of Sending Data Item		Category Of Receiving Data Item[1]		
		Alphabetic	Alphanumeric Edited Alphanumeric	Numeric Integer Numeric Non-Integer Numeric Edited
ALPHABETIC		Yes/3c	Yes/3a	No/2a
ALPHANUMERIC		Yes/3c	Yes/3a	Yes/3b
ALPHANUMERIC EDITED		Yes/3c	Yes/3a	No/2a
NUMERIC	INTEGER	No/2b	Yes/3a	Yes/3b
	NON-INTEGER	No/2b	No/2c	Yes/3b
NUMERIC EDITED		No/2b	Yes/3a	Yes/2a

1 - The relevant rule number is quoted in these columns

THE MULTIPLY STATEMENT

Function

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

General Format

Format 1

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

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

Format 2

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

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

Syntax Rules

1. Each identifier must refer to a numeric elementary item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The composite of operands, which is that hypothetical data item resulting from the superimposition of all receiving data items aligned on their decimal points must not contain more than 18 digits.

General Rules

1. See The ROUNDED Phrase, The SIZE ERROR Phrase, The Arithmetic Statements, Overlapping Operands and Multiple Results in Arithmetic Statements in this Chapter.

2. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product; similarly for identifier-1 or literal-1 and identifier-3, etc.

3. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

THE PERFORM STATEMENT

Function

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

General Format

Format 1

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

Format 2

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

Format 3

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

Format 4

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

    VARYING { identifier-2 } FROM { identifier-3 }
            { index-name-1 }      { index-name-2 }
                                  { literal-1    }
        BY { identifier-4 } UNTIL condition-1
           { literal-2    }

  [ AFTER { identifier-5 } FROM { identifier-6 }
          { index-name-3 }      { index-name-4 }
                                { literal-3    }
      BY { identifier-7 } UNTIL condition-2
         { literal-4    }

    [ AFTER { identifier-8 } FROM { identifier-9 }
            { index-name-5 }      { index-name-6 }
                                  { literal-5    }
        BY { identifier-10 } UNTIL condition-3 ] ]
           { literal-6     }
```

Syntax Rules

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

2. Each literal represents a numeric literal.

3. The words THRU and THROUGH are equivalent.

4. If an index-name is specified in the VARYING or AFTER phrase, then:

   a. The identifier in the associated FROM and BY phrases must be an integer data item.

   b. The literal in the associated FROM phrase must be a positive integer.

   c. The literal in the associated BY phrase must be a non-zero integer.

5. If an index-name is specified in the FROM phrase, then:

   a. The identifier in the associated VARYING or AFTER phrase must be an integer data item.

   b. The identifier in the associated BY phrase must be an integer data item.

   c. The literal in the associated BY phrase must be an integer.

6. Literal in the BY phrase must not be zero.

7. Condition-1, condition-2, condition-3 may be any conditional expression as described under CONDITIONAL EXPRESSIONS in this Chapter.

8. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declarative section of the program then both must be procedure-names in the same declarative section.

General Rules

1. The data items referenced by identifier-4, identifier-7, and identifier-10 must not have a zero value.

2. If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, then the data item referenced by the identifier must have a positive value.

3. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1 (except as indicated in general rules 4b, 4c, and 4d). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

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

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

   c. If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

   d. If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

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

5. If control passes to these procedures other than via a PERFORM statement the procedures are executed right through to the next executable statement in the main program as if they were just part of the main program.

6. The PERFORM statements operate as follows with rule 5 above applying to all formats:

   a. Format 1 is the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is executed once and then control passes to the next executable statement following the PERFORM statement.

   b. Format 2 is the PERFORM...TIMES. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is

negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the next executable statement following the PERFORM statement.

During execution of the PERFORM statement, references to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

c. Format 3 is the PERFORM...UNTIL. The specified procedures are performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the next executable statement after the PERFORM statement. If the condition is true when the PERFORM statement is entered, no transfer to procedure-name-1 takes place, and control is passed to the next executable statement following the PERFORM statement.

d. Format 4 is the PERFORM...VARYING. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER and FROM (current value) phrases also refers to index-names. When index-name appears in a VARYING and/or AFTER phrase, it is initialized and subsequently augmented (as described below) according to the rules of the SET statement. When index-name appears in the FROM phrase and identifier appears in an associated VARYING or AFTER phrase, identifier is initialized according to the rules of the SET statement; subsequent augmentation is as described below.

In Format 4, when one identifier is varied, identifier-2 is set to the value of literal-1 or the current value of identifier-3 at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The value of identifier-2 is augmented by the specified increment or decrement value (the value of identifier-4 or literal-2) and condition-1 is evaluated again. The cycle continues until this condition is true; at which point, control is transferred to the next executable statement following the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the next executable statement following the PERFORM statement.

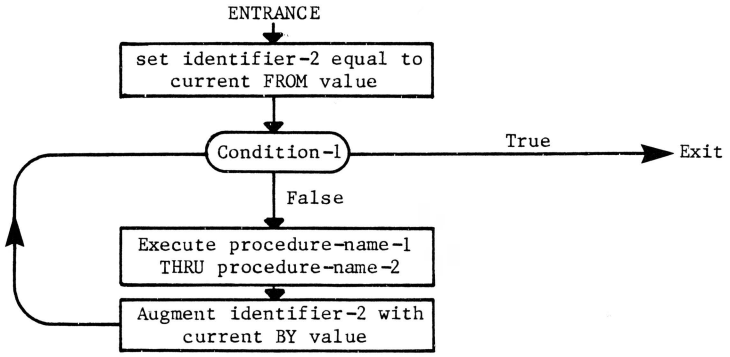


Figure 3-1. Flowchart for VARYING Phrase of a PERFORM Statement Having One Condition.

In Format 4, when two identifiers are varied, identifier-2 and identifier-5 are set to the current value of identifier-3 and identifier-6, respectively. After the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the next executable statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, then identifier-5 is augmented by identifier-7 or literal-4 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When condition-2 is true, identifier-5 is set to the value of literal-3 or the current value of identifier-6, identifier-2 is augmented by identifier-4 and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true.

During the execution of the procedures associated with the PERFORM statement, any change to the VARYING variable (identifier-2 and index-name-1), the BY variable (identifier-4), the AFTER variable (identifier-5 and index-name-3), or the FROM variable (identifier-3 and index-name-2) will be taken into consideration and will affect the operation of the PERFORM statement.

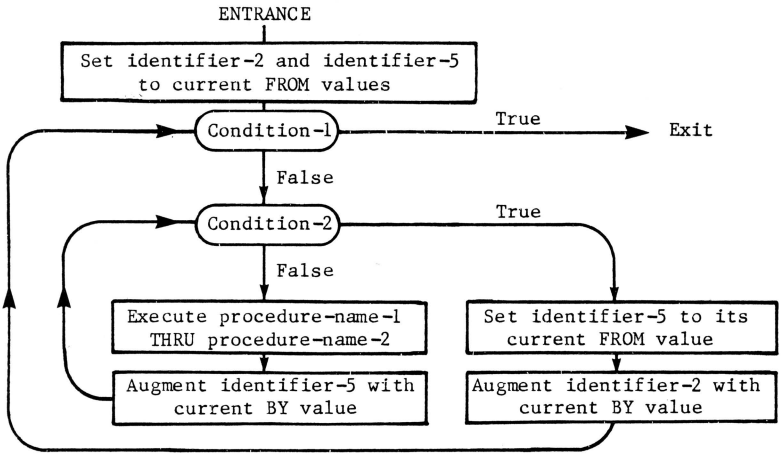


Figure 3-2. Flowchart for VARYING Phrase of PERFORM Statement with Two Conditions.

At the termination of the PERFORM statement identifier-5 contains the current value of identifier-6. Identifier-2 has a value that exceeds the last used setting by an increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-2 contains the current value of identifier-3.

When two identifiers are varied, identifier-5 goes through a complete cycle (FROM, BY, UNTIL) each time identifier-2 is varied.

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

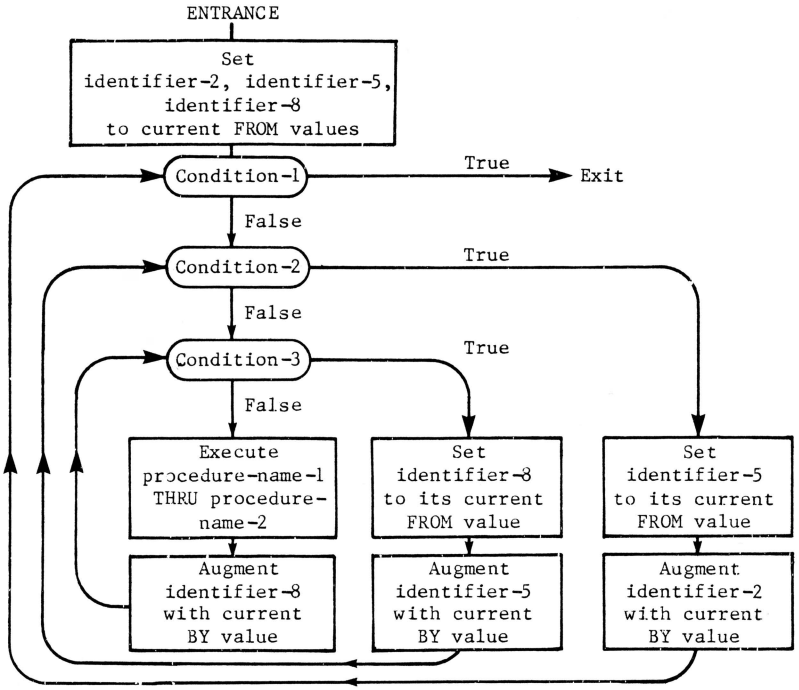


Figure 3-3. Flowchart for VARYING Phrase of PERFORM Statement with Three Conditions.

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

7. If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See Figure 3 - 4.

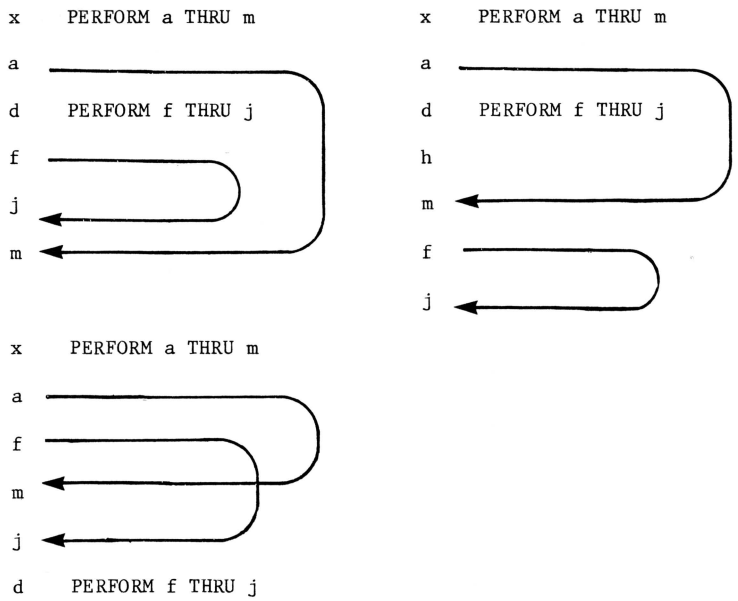


Figure 3-4. PERFORM Statements in Sequence.

8. A PERFORM statement that appears in a section that is not an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

   a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

   b. Sections and/or paragraphs wholly contained in a single independent segment.

9. A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

   a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

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

10. PERFORM statements must not be nested to greater than 22 levels.

THE STRING STATEMENT

Function

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

General Format

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

    [ [ identifier-4 ] [, { identifier-5 } ] ...
    [ [ literal-4    ]    { literal-5    } ]

                  DELIMITED BY { identifier-6 }
                               { literal-6    }  ] ...
                               { SIZE         }

    INTO  identifier-7  [WITH POINTER identifier-8]

    [; ON OVERFLOW imperative-statement]
```

Syntax Rules

1. Each literal may be any figurative constant without the optional word ALL.

2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-8, must be described implicitly or explicitly as usage is DISPLAY.

3. Identifier-8 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size plus 1 of the area referenced by identifier-7. The symbol 'P' may not be used in the PICTURE character-string of identifier-8.

4. Identifier-7 must represent an elementary alphanumeric data item without editing symbols or the JUSTIFIED clause.

5. Where identifier-1, identifier-2, ..., or identifier-3 is an elementary numeric data item, it must be described as an integer without the symbol 'P' in its PICTURE character-string.

General Rules

1. All references to identifier-1, identifier-2, identifier-3, literal-1, literal-2, literal-3 apply equally to identifier-4, identifier-5, identifier-6, literal-4, literal-5 and literal-6, respectively, and all recursions thereof.

2. Identifier-1, literal-1, identifier-2, literal-2 , represent the sending items. Identifier-7 represents the receiving item.

3. Literal-3, identifier-3, indicate the character(s) delimiting the move. If the SIZE phrase is used, the complete data item defined by identifier-1, literal-1, identifier-2, literal-2 , is moved. When a figurative constant is used as the delimiter, it stands for a single character nonnumeric literal.

4. When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit one-character data item where usage is DISPLAY.

5. When the STRING statement is executed, the transfer of data is governed by the following rules:

   a. Those characters from literal-1, literal-2, or from the contents of the data item referenced by identifier-1, identifier-2, are transferred to the contents of identifier-7 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space-filling will be provided. (See THE MOVE STATEMENT.)

   b. If the DELIMITED phrase is specified without the SIZE phrase, the contents of the data item referenced by identifier-1, identifier-2, or the value of literal-1, literal-2, are transferred to the receiving data item in the sequence specified in the STRING statement beginning with the leftmost character and continuing from left to right until the end of the data item is reached, or until the character(s) specified by literal-3, or by the contents of identifier-3 are encountered. The character(s) specified by literal-3, or by the data item referenced by identifier-3 are not transferred.

   c. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, literal-2, or the contents of the data item referenced by identifier-1, identifier-2, are transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-7 until all data has been transferred or the end of the data item referenced by identifier-7 has been reached.

6. If the POINTER phrase is specified, identifier-8 is explicitly available to the programmer, who is then responsible for setting its initial value. The initial value must not be less than one.

7. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-8 with an initial value of 1.

8. When characters are transferred to the data item referenced by identifier-7, the moves behave as though the characters were moved one at a time from the source into the character position of the data item referenced by identifier-7 designated by the value associated with identifier-8, and then identifier-8 was increased by one prior to the move of the next character. The value associated with identifier-8 is changed during execution of the STRING statement only by the behavior specified above.

9. At the end of execution of the STRING statement, only the portion of the data item referenced by identifier-7 that was referenced during the execution of the STRING statement is changed. All other portions of the data item referenced by identifier-7 will contain data that was present before this execution of the STRING statement.

10. If at any point at or after initialization of the STRING statement, but before execution of the STRING statement is completed, the value associated with identifier-8 is either less than one or exceeds the number of character positions in the data item referenced by identifier-7, no (further) data is transferred to the data item referenced by identifier-7, and the imperative statement in the ON OVERFLOW phrase is executed, if specified.

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

THE STOP STATEMENT

Function

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

General Format

$$\underline{\text{STOP}} \quad \left\{ \begin{matrix} \underline{\text{RUN}} \\ \text{literal} \end{matrix} \right\}$$

Syntax Rules

1. The literal may be numeric or non-numeric or may be any figurative constant, except ALL.

2. If the literal is numeric, then it must be an unsigned integer.

3. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules

1. If the RUN phrase is used, then the operating system ending procedure is instituted.

2. If STOP literal is specified, the literal is communicated to the operator. Continuation of the object program begins with the execution of the next executable statement in sequence.

THE SUBTRACT STATEMENT

Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

General Format

Format 1

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

    identifier-m [ROUNDED] [, identifier-n [ROUNDED]] ...

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

Format 2

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

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

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

Format 3

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

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

Syntax Rules

1. Each identifier must refer to a numeric elementary item except that in Format 2, each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item, and in Format 3, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits. (See The Arithmetic Statements in this Chapter).

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

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

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

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. See The Rounded Phrase, The Size Error Phrase, The Arithmetic Statement, Overlapping Operands and Multiple Results in Arithmetic Statements in this Chapter.

2. In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from the current value of identifier-m storing the result immediately into identifier-m, and repeating this process respectively for each operand following the word FROM.

3. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m and the result of the subtraction is stored as the new value of identifier-n, identifier-o, etc.

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

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

THE UNSTRING STATEMENT

Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

General Format

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

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

  [, COUNT IN identifier-6]

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

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

[; ON OVERFLOW imperative-statement]
```

Syntax Rules

1. Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant without the optional word ALL.

2. Identifier-1, identifier-2, identifier-3, identifier-5, and identifier-8 must be described, implicitly or explicitly, as an alphanumeric data item.

3. Identifier-4 and identifier-7 may be described as either alphabetic (except that the symbol 'B' may not be used in the PICTURE character-string), alphanumeric, or numeric (except that the symbol 'P' may not be used in the PICTURE charatcer-string), and must be described as usage is DISPLAY.

4. Identifier-6, identifier-9, identifier-10, and identifier-11 must be described as elementary numeric integer data items (except that the symbol 'P' may not be used in the PICTURE character-string).

5. No identifier may name a level 88 entry.

6. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

General Rules

1. All references to identifier-2, literal-1, identifier-4, identifier-5 and identifier-6, apply equally to identifier-3, literal-2, identifier-7, identifier-8 and identifier-9, respectively, and all recursions thereof.

2. Identifier-1 represents the sending area.

3. Identifier-4 represents the data receiving area. Identifier-5 represents the receiving area for delimiters.

4. Literal-1 or the data item referenced by identifier-2 specifies a delimiter.

5. The data item referenced by identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 isolated by the delimiters for the move to the data item referenced by identifier-4. This value does not include a count of the delimiter character(s).

6. The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1.

7. The data item referenced by identifier-11 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.

8. When a figurative constant is used as the delimiter, it stands for a single character nonnumeric literal.

   When the ALL phrase is specified, one occurrence or two or more contiguous occurrences of literal-1 (figurative constant or not) or the contents of the data item referenced by identifier-2 are treated as if it were only one occurrence, and this occurrence is moved to the receiving data item according to the rules in general rule 13d.

9. When any examination encounters two contiguous delimiters, the current receiving area is either space or zero filled according to the description of the receiving area.

10. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the computer's character set.

11. Each literal-1 or the data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item and in the order given, to be recognized as a delimiter.

12. When two or more delimiters are specified in the DELIMITED BY phrase, an 'OR' condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

13. When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

    a. If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by the contents of the data item referenced by identifier-10. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by the value of literal-1 or the data item referenced by identifier-2 is encountered. (See general rule 11.) If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

       If the end of the data item referenced by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

    c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement. (See THE MOVE STATEMENT.)

    d. If the DELIMITER IN phrase is specified, the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement. (See THE MOVE STATEMENT.) If the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is space-filled.

    e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s) if any) is moved into the area referenced by identifier-1 according to the rules for an elementary move.

f. If the DELIMITED BY phrase is specified the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.

g. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The behaviour described in paragraph 13b through 13f is repeated until either all the characters are exhausted in the data item referenced by identifier-1, or until there are no more receiving areas.

14. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

15. The contents of the data item referenced by identifier-10 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is complete, the contents of the data item referenced by identifier-10 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of the data item referenced by identifier-11 contains a value equal to its initial value plus the number of data receiving items acted upon.

17. Either of the following situations causes an overflow condition:

a. An UNSTRING is initiated, and the value in the data item referenced by identifier-10 is less than 1 or greater than the size of the data item referenced by identifier-1.

b. If, during execution of an UNSTRING statement, all data receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

18. When an overflow condition exists, the UNSTRING operation is terminated. If an ON OVERFLOW phrase has been specified, the imperative statement included in the ON OVERFLOW phrase is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next executable statement.

19. The evaluation of subscripting and indexing for the identifiers is as follows:

    a. Any subscripting or indexing associated with identifier-1, identifier-10, identifier-11 is evaluated only once, immediately before any data is transferred as the result of the execution of the UNSTRING statement.

    b. Any subscripting or indexing associated with identifier-2, identifier-3, identifier-4, identifier-5, identifier-6 is evaluated immediately before the transfer of data into the respective data item.

Chapter 4

Table Handling

INTRODUCTION TO THE TABLE HANDLING MODULE

The Table Handling module provides a capability for defining tables of contiguous data items and accessing an item relative to its position in the table. Language facilities are provided for specifying how many times an item is to be repeated. Each item may be identified through use of a subscript or an index (see Chapter 2).

Table Handling provides a capability for accessing items in fixed length tables of multiple dimensions. The maximum number of multiple dimensions if the ANSI switch is on (see Chapter 2) is restricted to three.

DATA DIVISION IN THE TABLE HANDLING MODULE

THE OCCURS CLAUSE

Function

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indices.

General Format

Format 1

```
OCCURS integer-2 TIMES
   [ { ASCENDING  } KEY IS data-name-2 [, data-name-3]  ... ]  ...
   [ { DESCENDING }                                         ]
     [ [INDEXED BY index-name-1  [, index-name-2]  ... ]  ...
```

Format 2

```
OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1
   [ { ASCENDING  } KEY IS data-name-2  [, data-name-3]  ... ]  ...
   [ { DESCENDING }                                          ]
     [ [INDEXED BY index-name-1  [, index-name-2]  ... ]
```

Syntax Rules

1. Where both integer-1 and integer-2 are used, the value of integer-1 must be less than the value of integer-2.

2. The data description of data-name-1 must describe a positive integer.

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

4. Data-name-2 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.

5. Data-name-3, etc., must be the name of an entry subordinate to the group item which is the subject of this entry.

6. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing. The index-name identified by this clause is not defined elsewhere since its allocation and format are dependent on the hardware, and not being data, cannot be associated with any data hierarchy.

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

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

   a. Has 01, 66, 77 or 88 level-number (if the ANSI switch has been set).

   b. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.

9. In Format 2, the data item defined by data-name-1 must not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.

10. If data-name-2 is not the subject of this entry, then:

    a. All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of this entry.

    b. Items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause.

    c. There must not be any entry that contains an OCCURS clause between the items identified by the data-names in the KEY IS phrase and the subject of this entry.

11. Index-name-1, index-name-2, ... must be unique words within the program.

General Rules

1. The OCCURS clause is used in defining tables and other homogenous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than SEARCH or USE FOR DEBUGGING. Further, if the subject of this entry is the name of a group item, then all data-names belonging to the group must be subscripted or indexed whenever they are used as operands, except as the object of a REDEFINES clause. (See under headings Subscripting, Indexing and Identifier in Chapter 2).

2. Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described. (See restriction in general rule 2 under Data Description Entries Other Than Condition Names in Chapter 3).

3. The number of occurrences of the subject entry is defined as follows:

   a. In Format 1, the value of integer-2 representing the exact number of occurrences.

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

      This format specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

      The value of the data item referenced by data-name-1 must fall within the range of integer-1 through integer-2. Reducing the value of the data item referenced by data-name-1 makes the contents of data items, whose occurrence numbers now exceed the value of the data item referenced by data-name-1, unpredictable.

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

5. The KEY IS phrase is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained data-name-2, data-name-3, etc. The ascending or descending order is determined according to the rules for comparison of operands (see Comparison of Numeric Operands, Comparison of Nonnumeric Operands in Chapter 3). The data-names are listed in their descending order of significance.

THE USAGE CLAUSE

Function

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

General Format

```
[USAGE IS]  INDEX
```

Syntax Rules

1. An index data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.

2. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

General Rules

1. The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

2. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. The compiler will allocate a 2 byte binary field. If a group item is described with the USAGE IS INDEX clause the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in the SEARCH or SET statement or in a relation condition.

3. An index data item can be part of a group which is referred to in a MOVE or input-output statement, in which case no conversion will take place.

PROCEDURE DIVISION IN THE TABLE HANDLING MODULE

RELATION CONDITION

Comparisons Involving Index-Names and/or Index Data Items

Relation tests may be made between the following data items:

* Two index-names. The result is the same as if the corresponding occurrence numbers were compared.

* An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

* An index data item and an index-name or another index data item. The actual values are compared without conversion.

* The result of the comparison of an index data item with any data item or literal not specified above is undefined.

OVERLAPPING OPERANDS

When a sending and a receiving item in a SET statement share a part of their storage areas, the result of the execution of such a statement is undefined.

THE SEARCH STATEMENT

Function

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated indexname to indicate that table element.

General Format

Format 1

```
SEARCH identifier-1  [VARYING  {identifier-2}]
                               {identifier-1}

    [; AT END imperative-statement-1]
     ; WHEN condition-1  [{imperative-statement-2}]
                         {NEXT SENTENCE         }

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

Format 2

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

                 {                                                  }
                 {                                                  }
                 {                  {identifier-3                   }
                 {                  {literal-1                      }
                 {data-name-1  {IS EQUAL TO}  {arithmetic-expression-2}
     ; WHEN      {             {IS =       }  {                    }
                 {                                                  }
                 {condition-name-1                                  }
                 {                                                  }

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

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

NOTE: The required relational character '=' is not underlined to avoid confusion with other symbols.

Syntax Rules

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

2. Identifier-2, when specified, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.

3. In Format 1, condition-1, condition-2, etc., may be any condition as described in CONDITION EXPRESSIONS in Chapter 3.

4. In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of identifier-1. Each data-name-1, data-name-2 may be qualified. Each data-name-1, data-name-2 must be indexed by the first index-name associated with identifier-1 along with other indices or literals as required, and must be referenced in the KEY clause of identifier-1. Identifier-3, identifier-4, or identifiers specified in arithmetic-expression-1, arithmetic-expression-2 must not be referenced in the KEY clause of identifier-1 or be indexed by the first index-name associated with identifier-1.

   In Format 2, when a data-name in the KEY clause of identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY clause of identifier-1 is referenced, all preceding data-names in the KEY clause of identifier-1 or their associated condition-names must also be referenced.

General Rules

1. If Format 1 of the SEARCH is used, a serial type of search operation takes place, starting with the current index setting.

   a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. The number of occurrences of identifier-1, the last of which is the highest permissible, is discussed in the occurs clause. (See THE OCCURS CLAUSE in Chapter 4.) Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next executable sentence.

   b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1 (the number of occurrences of identifier-1, the last of which is the highest permissible is discussed in the OCCURS clause); the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions is satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element outside the permissible range of occurrence values, in which case the search terminates as indicated in 1-a above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

2. In a Format 2 SEARCH, the results of the SEARCH ALL operation are predictable only when:

   a. The data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of identifier-1, and

   b. The contents of the key(s) referenced in the WHEN clause are sufficient to identify a unique table element.

3. If Format 2 of the SEARCH is used, a non-serial type of search operation may take place; the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the

table, or that is less than the value that corresponds to the first element of the table. The length of the table is discussed in the OCCURS clause.

If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next executable sentence when this phrase is not specified; in either case the final setting of the index is not predictable. If all conditions can be satisfied, the index indicates an occurrence that allows the conditions to be satisfied, and control passes to imperative-statement-2.

4. After execution of imperative-statement-1, imperative-statement-2, or imperative-statement-3, that does not terminate with a GO TO statement, control passes to the next executable sentence.

5. In Format 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

6. In Format 1, if the VARYING phrase is not used, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

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

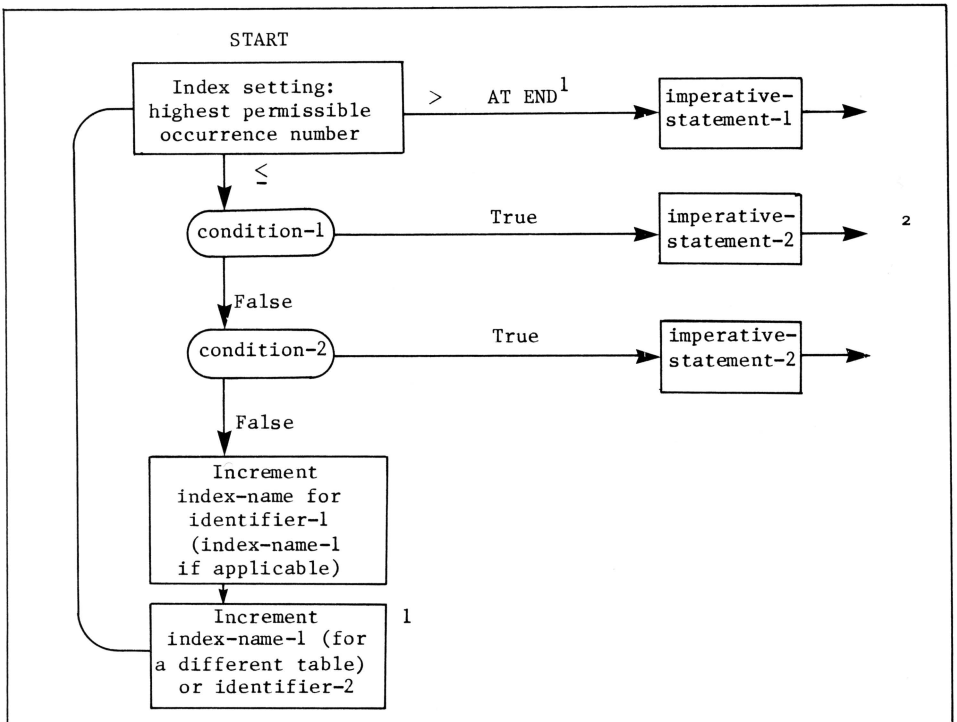
   a. If the VARYING index-name-1 phrase is used, and if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by the index-name associated with identifier-1 is incremented.

   b. If the VARYING identifier-2 phrase is specified, and identifier-2 is an index data item, then the data item referenced by identifier-2 is incremented by the same amount as, and at the same time as, the index associated with identifier-1 is incremented. If identifier-2 is not an index data item, the data item referenced by identifier-2 is incremented by the value (1) at the same time as the index referenced by the index-name associated with identifier-1 is incremented.

8. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two or three dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2

or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire two or three dimensional table is not necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

Figure 4-1 shows a flowchart of the Format 1 SEARCH operation containing two WHEN phrases.



1 - These operations are options included only when specified in the SEARCH statement.

2 - Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

Figure 4-1. Flowchart of SEARCH Operation with Two WHEN Phrases

THE SET STATEMENT

Function

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

General Format

Format 1

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

Format 2

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

Syntax Rules

1. All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

2. Identifier-1 and identifier-3 must name either index data items, or elementary items described as an integer.

3. Identifier-4 must be described as an elementary numeric integer.

4. Integer-1 and integer-2 may be signed. Integer-1 must be positive.

General Rules

1. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

2. If index-name-3 is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the associated table.

If index-name-4, index-name-5 is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. If index-name-1, index-name-2 is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. The value of the index associated with an index-name after the execution of a SEARCH or PERFORM statement may be undefined. (See THE SEARCH STATEMENT and THE PERFORM STATEMENT in Chapter 3.)

3. In Format 1, the following action occurs:

   a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name 1, no conversion takes place.

   b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index item; no conversion takes place in either case.

   c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

   d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4; thereafter, the process is repeated for index-name-5, etc. Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.

5. Data in Table 4-1 represents the validity of various operand combinations in the SET statement. The general rule reference indicates the applicable general rule.

Table 4-1. SET Statement Valid Operand Combinations.

Sending Item	Receiving Item[1]		
	Integer Data Item	Index-Name	Index Data Item
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data Item	No/3c	Valid/3a	No/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b[2]
Index Data Item	No/3c	Valid/3a[2]	Valid/3b[2]

1 = Rule numbers under General Rules above are referred to.
2 = No conversion takes place

Chapter 5

Sequential Input and Output

INTRODUCTION TO THE SEQUENTIAL I-O MODULE

The Sequential I-O module provides a capability to access records of a file in established sequence. The sequence is established as a result of writing the records to the file. It also provides for the specification of re-run points and the sharing of memory areas among files.

LANGUAGE CONCEPTS

Organization

Sequential files are organized such that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except in the case where records are added to the end of the file.

Access Mode

In the sequential access mode, the sequence in which records are accessed is the order in which the records were originally written.

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, or REWRITE statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as Status Key 1 and is set to indicate one of the following conditions upon completion of the input-output operation.

'0' - indicates Successful Completion
'1' - indicates At End
'3' - indicates Permanent Error
'9' - indicates an Operating System Error Message

The meanings of the above indications are as follows:

0 - Successful Completion. The input-output statement was successfully executed.

1 - At End. The sequential READ statement was unsuccessfully executed either as a result of an attempt to read a record when no next logical record exists in the file or as a result of the first READ statement being executed for a file described with the OPTIONAL clause, and that file was not available to the program at the time its associated OPEN statement was executed.

3 - Permanent Error. The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error, or transmission error.

9 - Operating System Error Message. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the Operating System Error Message. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the values of status key 1 and status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as Status Key 2 and is used to further describe the results of the input-output operation. This character will contain a value as follows:

* If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

* When status key 1 contains a value of '3' an irrecoverable error has occurred. This is treated as a fatal error by the Operating System.

* When status key 1 contains a value of '9', the value of status key 2 is the operating system error message number (for those operating systems which designate errors numerically). The Apple III COBOL Introduction and Operating System Manual contains details of this status-key-2 representation.
Note that it is not possible to extract this number directly.

Valid Combinations of Status Keys 1 and 2

The valid permissible combinations of the values of status key 1 and status key 2 are shown in the following table. An 'X' at an intersection indicates a valid permissible combination.

Status Key 1	Status Key 2
	No Further Information (0)
Successful Completion (0)	X
At End (1)	X
Permanent Error (3)	X
Implementor Defined (9)	O/S Error Number

The AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition, see THE READ STATEMENT later in this Chapter.

LINAGE-COUNTER

The reserved word LINAGE-COUNTER is a name for a special register generated by the presence of a LINAGE clause in a file description entry. The implicit description is that of an unsigned integer whose size is equal to integer-1 or the data item referenced by data-name-1 in the LINAGE clause. See THE LINAGE CLAUSE later in this Chapter.

ENVIRONMENT DIVISION IN THE SEQUENTIAL I-O MODULE

INPUT-OUTPUT SECTION

The FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information. (See also Appendix I in this manual).

General Format

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

The FILE CONTROL Entry

Function

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

General Format

```
SELECT    [OPTIONAL]  file-name
     ASSIGN TO { external-file-name-literal } [ , { external-file-name-literal } ]
               { file-identifier               ]   { file-identifier               }
     [ ; RESERVE integer-1 [ AREA ] ]
                           [ AREAS ]
     ; ORGANIZATION IS  SEQUENTIAL
                        LINE SEQUENTIAL
     [; ACCESS MODE IS SEQUENTIAL]
     [; FILE STATUS IS data-name-1]
```

Syntax Rules

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

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

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. Data-name-1 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section.

5. Data-name-1 may be qualified.

6. When the ORGANIZATION IS SEQUENTIAL clause is not specified, the ORGANIZATION IS SEQUENTIAL clause is implied.

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

General Rules

1. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium. See Appendix F in the Apple III COBOL Operating Guide. The first assignment takes effect. Subsequent assignments within any one ASSIGN clause are for documentation purposes only.

2. The RESERVE clause allows the user to specify the number of input-output areas allocated. If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-1.

3. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

4. When LINE SEQUENTIAL ORGANIZATION is specified, the file is treated as consisting of variable length records, each record containing one line of data. The definition of a line of data varies with different operating systems. Some terminate line "records" with the Carriage Return and Line Feed characters, or one of them, and some pad out as fixed length records. Apple III COBOL therefore is always compatible with the Editor software in any Operating System in this respect.

5. Records in the file are accessed in the sequence dictated by the file organization. This sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the file is created or extended.

6. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-1 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement (See I-O STATUS in this Chapter).

The I-O-CONTROL Paragraph

Function

The I-O CONTROL paragraph specifies the points at which re-run is to be established, the memory area which is to be shared by different files, and the location of files on a multiple file reel.

General Format

```
I-O-CONTROL.
[                                                  {  {[END OF] {REEL}  }                    }
                                                   {  {       {UNIT}  }                    }
                                                   {  {  integer-1 RECORDS } OF file-name-2 }..
; RERUN [ON {file-name-1        }] EVERY          {integer-2 CLOCK-UNITS                    }
          {implementor-name     }                 {condition-name                           }
[; SAME [RECORD] AREA FOR file-name-3 {, file-name-4} ... ] ...

[; MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-3]

        [file-name-6 [POSITION integer-4]] ...]...
```

Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. File-name-1 must be a sequentially organized file.

3. The END OF REEL/UNIT clause may only be used if file-name-2 is a sequentially organized file.

4. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, implementor-name must be given in the RERUN clause.

5. More than one RERUN clause may be specified for a given file-name-2 subject to the following restrictions:

   a. When multiple integer-1 RECORD clauses are specified, no two of them can specify the same file-name-2.

   b. When multiple END OF REEL or END OF UNIT clauses are specified, no two of them may specify the same file-name-2.

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

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

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

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

c. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in the SAME RECORD AREA clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

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

General Rules

1. The RERUN clause is treated as for documentation purposes only.

2. The SAME AREA clause specifies that two or more files that do not represent sort or merge files are to use the same memory area during processing. The area being stored includes all storage area assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time. (See Syntax Rule 6c.)

3. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area i.e., records are aligned on the leftmost character position.

4. The MULTIPLE FILE clause is treated as for documentation purposes only.

DATA DIVISION IN THE SEQUENTIAL I-O MODULE

FILE SECTION

In a L/II COBOL program the file description entry (FD) represents the highest level of organisation in the File Section. The File Section header is followed by a file description entry consisting of a level indicator (FD), a file-name and a series of independent clauses. The FD clauses specify the size of the logical and physical records, the presence or absence of label records, the value of implementor-defined label items, the names of the data records which comprise the file. The entry itself is terminated by a period.

RECORD DESCRIPTION STRUCTURE

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A record description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in CONCEPT OF LEVELS in Chapter 2, while the elements allowed in a record description are shown in the Data Description - Complete Entry Skeleton in Chapter 3.

THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

Function

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

General Format

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

Syntax Rules

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.

2. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial. All clauses are optional when the ANSI switch is unset (See Chapter 2).

3. One or more record description entries must follow the file description entry.

THE BLOCK CONTAINS CLAUSE

Function

The BLOCK CONTAINS clause specifies the size of a physical record.

General Format

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

General Rule

This clause is required for documentation purposes only.

THE CODE-SET CLAUSE

Function

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

General Format

```
CODE-SET IS alphabet-name
```

Syntax Rules

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

2. The alphabet-name clause referenced by the CODE-SET clause must not specify the literal phrase.

3. The CODE-SET clause may only be specified for non-disk files.

General Rule

The CODE-SET clause is specified for documentation purposes only.

THE DATA RECORDS CLAUSE

Function

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

General Format

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

Syntax Rule

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

General Rule

The DATA RECORDS clause is specified for documentation purposes only.

THE LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present.

General Format

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

Syntax Rule

This clause is required in every file description entry, when the ANSI switch is set.

General Rule

This clause is used for documentation purposes only.

THE LINAGE CLAUSE

Function

The Linage clause provides a means for specifying the depth of a logical page in terms of number of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

General Format

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

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

Syntax Rules

1. Data-name-1, data-name-2, data-name-3, data-name-4 must reference elementary unsigned numeric integer data items.

2. The value of integer-1 must be greater than zero.

3. The value of integer-2 must not be greater than integer-1.

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

General Rules

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

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

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

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

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

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

   The footing area comprises the area of the logical page between the line represented by the value of integer-2 or the data item referenced by data-name-2 and the line represented by the value of integer-1 or the data item referenced by data-namel, inclusive.

6. The value of integer-1, integer-3, and integer-4, if specified, will be used at the time the file is opened by the execution of an OPEN

statement with the OUTPUT phrase, to specify the number of lines that comprise each of the indicated sections of a logical page. The value of integer-2, if specified, will be used at that time to define the footing area. These values are used for all logical pages written for the file during a given execution of the program.

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

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

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

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

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

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

   b. LINAGE-COUNTER may be referenced, but may not be modified, by Procedure Division statements. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.

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

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

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

* When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one. (See THE WRITE STATEMENT.)

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

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

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

THE RECORD CONTAINS CLAUSE

Function

The RECORD CONTAINS clause specifies the size of data records.

General Format

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

General Rule

The size of each data record is completely defined within the record description entry, therefore this clause is never required. The RECORD CONTAINS clause is specified for documentation purposes only.

THE VALUE OF CLAUSE

Function

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

General Format

```
VALUE OF data-name-1 IS {data-name-2}
                        {literal-1   }

     [, data-name-3  IS {data-name-4} ] ...
                        {literal-2   }
```

Syntax Rules

1. Data-name-2, data-name-3, etc should be qualified when necessary but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

2. Data-name-2, data-name-3, etc must be in the Working-Storage Section.

General Rules

1. This clause is used for documentation purposes only.

2. On input data-name-1 is checked against data-name-2 or literal-1 as specified and data-name-3 against data-name-4 or literal-2 as specified etc.

   On output data-name-2 or literal-1 are substituted for data-name-1 as specified and data-name-4 or literal-2 from data-name-3 etc.

3. A figurative constant may be substituted in the format above wherever a literal is specified.

PROCEDURE DIVISION IN THE SEQUENTIAL I-O MODULE

THE CLOSE STATEMENT

Function

The CLOSE statement terminates the processing of files.

General Format

```
CLOSE file-name-1 [ { REEL } [ WITH NO REWIND ]
                    { UNIT } [ FOR REMOVAL    ]
                  [ WITH   ] { NO REWIND }
                             { LOCK      }
  [ , file-name-2 [ { REEL } [ WITH NO REWIND ]
                    { UNIT } [ FOR REMOVAL    ]
                  [ WITH   ] { NO REWIND }
                             { LOCK      } ] ] ....
```

Syntax Rule

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

General Rules

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

2. The action taken if the file is in the open mode when a STOP RUN statement is executed is to close the file. The action taken for a file that has been opened in a called program and not closed in that program prior to the execution of a CANCEL statement for that program is to close the file.

3. If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement (except the SORT or MERGE statements with the USING or GIVING phrases) can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

4. Following the successful execution of a CLOSE statement the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

THE OPEN STATEMENT

Function

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

General Format

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

Syntax Rules

1. The REVERSED and NO REWIND phrases can only be used with sequential files and are for documentation purposes only.

2. The I-O phrase can be used only for disk files, except for files in Line Sequential organization.

3. The EXTEND phrase can be used only for sequential files, and Line Sequential files.

4. The EXTEND phrase must not be specified with multiple file reels.

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

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The successful execution of an OPEN statement makes the associated record area available to the program.

3. Prior to the successful execution of an OPEN statement for a given file, no statement (except for a SORT or MERGE statement with the USING or GIVING phrases) can be executed that references that file, either explicitly or implicitly.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statement. In Table 5-1, 'X' at an intersection indicates that the specified statement, used in the sequential access mode, may be used with the sequential file organization and open mode given at the top of the column.

Table 5-1. Permissible Combinations of Statements and OPEN Modes for Sequential I-O.

Statement	Open Mode			
	Input	Output	Input-Output[1]	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

1 - This OPEN mode is not supported for ORGANIZATION line sequential files.

5. A file may be opened with the INPUT, OUTPUT, EXTEND and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, for that file.

6. Execution of the OPEN statement does not obtain or release the first data record.

7. The ASSIGNed name in the SELECT statement for a file is processed as follows:

   a. When the INPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNed name to be checked in accordance with the operating system conventions for opening files for input.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNed name to be written in accordance with the operating system conventions for opening files for output.

8. The file description entry for file-name-1, file-name-2, file-name-5, file-name-6, file-name-7 and file-name-8 must be equivalent to that used when this file was created.

9. If an input file is designated with the OPTIONAL phrase in its SELECT clause, the object program causes an interrogation for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the AT END condition to occur.

10. If the storage medium for the file permits rewinding, execution of the OPEN statement causes the file to be positioned at its beginning.

11. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file will result in an AT END condition. If the file does not exist, OPEN INPUT will cause an error status.

12. When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file had been opened with the OUTPUT phrase.

13. The I-O phrase permits the opening of a disk file for both input and output operations except for files in ORGANIZATION LINE SEQUENTIAL.

14. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records. If a file of the same name exists it will be deleted. If write-protected on error will occur.

THE READ STATEMENT

Function

The READ statement makes available the next logical record from a file.

General Format

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

Syntax Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

2. The AT END phrase must be specified if no applicable USE procedure is specified for file-name.

General Rules

1. The associated file must be open in the INPUT or I-O mode at the time this statement is executed. (See THE OPEN STATEMENT in this Chapter).

2. The record to be made available by the READ statement is determined as follows:

   a. If the current record pointer was positioned by the execution of the OPEN statement, the record pointed to by the current record pointer is made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See I-O STATUS in this Chapter).

4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available to the object program prior to the execution of any statement following the READ statement.

5. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement without the CORRESPONDING phrase.

6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

7. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

8. If, at the time of execution of a READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

9. If the end of a reel or unit is recognized during the execution of a READ statement, an end-of-file status condition exists.

   a. The standard ending reel/unit label procedure.

   b. A reel/unit swap.

   c. The standard beginning reel/unit label procedure.

   d. The first data record of the new reel/unit is made available.

10. If a file described with the OPTIONAL clause is not present at the time the file is opened, then at the time of the execution of the first READ statement for the file, the AT END condition occurs and the execution of the READ statement is unsuccessful. The standard end of file procedures are not performed. (See The FILE-CONTROL paragraph and The OPEN and the USE statement descriptions in this Chapter.) Execution of the program then proceeds as in general rule 12.

11. If, at the time of the execution of a READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful. (See I-O STATUS).

12. When the AT END condition is recognized the following actions are taken in the specified order:

a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. (See I-O STATUS).

b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file and that procedure is executed.

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

13. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

14. When the AT END condition has been recognized, a READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

THE REWRITE STATEMENT

Function

The REWRITE statement logically replaces a record existing in a disk file.

General Format

REWRITE record-name [FROM identifier]

Syntax Rules

1. Record-name and identifier must not refer to the same storage area.

2. Record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

3. The INVALID KEY phrase must not be specified for a REWRITE statement that references a file in sequential access mode.

General Rules

1. The file associated with record-name must be a disk file and must be open in the I-O mode at the time of execution of this statement. (See THE OPEN STATEMENT in this Chapter).

2. The last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The operating system logically replaces the record that was accessed by the READ statement.

3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is saved in a SAME RECORD AREA clause. In this case, not only is the record still available to the program in the record area as a record of this file, but as a record of other files named in the SAME RECORD AREA clause.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   MOVE identifier TO record-name

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See I-O STATUS in this Chapter).

8. The REWRITE statement cannot be used with line sequential files.

THE USE STATEMENT

Function

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

General Format

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

Syntax Rules

1. A USE statement, when present, must immediately follow a section header in the declaratives section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedure to be used.

2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

3. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

General Rules

1. If the AT END phase has not been specified in the input-output statement, the designated procedures are executed by the input-output system after completing the standard input-output error routine upon recognition of the AT END condition

2. After execution of a USE procedure, control is returned to the invoking routine.

3. Within a USE procedure, there must not be any reference to any non-declarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

4. Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

THE WRITE STATEMENT

Function

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of lines within a logical page.

General Format

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

[                                                  [LINE ]
  {BEFORE}                                  {integer [LINES]}
  {AFTER }   ADVANCING                      {                }
                                            {TAB             }
                                            {mnemonic-name   }
                                            {PAGE            }
  [; AT {END-OF-PAGE}   imperative-statement]
        {EOP         }
```

Syntax Rules

1. Record-name and identifier-1 must not reference the same storage area.

2. When TAB is specified the result is to cause the paper to throw to the standard vertical tabulation position. A user-defined mnemonic-name can be used instead of TAB if they are associated in the SPECIAL-NAMES paragraph. (See The SPECIAL-NAMES Paragraph earlier in this Chapter.)

3. The record-name is the name of a logical record in the File Section of the Data Division, and may be qualified.

4. When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

5. Integer on the value of the data item referenced by identifier-2 may be zero.

6. If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry for the associated file.

7. The words END-OF-PAGE and EOP are equivalent.

8. The ADVANCING TAB phrase cannot be specified when writing a record to a file whose file description entry contains the LINAGE clause.

General Rules

1. The associated file must be open in the OUTPUT OR EXTEND mode at the time of the execution of this statement. (See THE OPEN STATEMENT in this Chapter).

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful due to a boundary violation.

   The logical record is also available to the program as a record of other files referenced in the SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier-1 TO record-name

      according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

      The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

      After execution of the WRITE statement is complete, the information in the area referenced by identifier-1 is available, even though the information in the area referenced by record-name may not be. (See general rule 2.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See I-O STATUS in this Chapter).

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. The number of character positions on a disk required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

8. The execution of the WRITE statement releases a logical record to the operating system.

9. The ADVANCING phrase allows control of the vertical positioning of each line on a representation of a printed page.

   a. With ORGANIZATION SEQUENTIAL if the ADVANCING phrase is not used, automatic advancing is provided when output is directed to a list-device to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

      i. If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.

      ii. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced.

      iii. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced.

      iv. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file whose description entry contains a LINAGE clause, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause.

   b. With ORGANIZATION LINE SEQUENTIAL, if the ADVANCING phrase is not used, automatic advancing of one line is provided to act in accordance with the convention of your operating system text editor (usually as if the user had specified BEFORE ADVANCING 1 LINE).

      If the ADVANCING phrase is used, advancing is provided according to rules 9a(i) through 9a(iv) above.

10. If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, the imperative-statement specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name.

11. An end-of-page condition is reached whenever the execution of a given WRITE statement with the END-OF-PAGE phrase occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and then the imperative statement in the END-OF-PAGE phrase is executed.

    An automatic page overflow condition is reached whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

    This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. The imperative statement in the END-OF-PAGE clause, if specified, is executed after the record is written and the device has been repositioned.

    If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

    If integer-2 or data-name-2 of the LINAGE clause is specified, but the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value of both integer-2 or the data item referenced by data-name-2 and integer-1 or the data item referenced by data-name-1, then the operation proceeds as if integer-2 or data-name-2 had not been specified.

12. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

    a. The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation. (See I-O STATUS in this Chapter).

    b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure will then be executed.

    c. If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.

Chapter 6

Relative Input and Output

INTRODUCTION TO THE RELATIVE I-O MODULE

The Relative I-O module provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in a relative file is uniquely identified by an integer value greater than zero which specifies the record's ordinal position in the file.

LANGUAGE CONCEPTS

Organization

Relative file organization is permitted only on disk devices. A relative file consists of records which are identified by relative record numbers. The file may be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Records are stored and retrieved based on this number. For example, the tenth record area, whether or not records have been written in the first through the ninth record areas.

Access Modes

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records which currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in a relative key data item.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

Current Record Pointer

The current record printer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN, START and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation.

'0' - indicates Successful Completion
'1' - indicates At End
'2' - indicates Invalid Key
'3' - indicates Permanent Error
'9' - indicates an Operating System Error Message

The meaning of the above indications are as follows:

'0' - Successful Completion. The input-output statement was successfully executed.

'1' - At End. The Format 1 READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

'2' - Invalid Key. The input-output statement was unsuccessfully executed as a result of one of the following:

* Duplicate Key
* No Record Found
* Boundary Violation

'3' - Permanent Error. The input-output statement was unsuccessfully executed as the reult of an input-output error, such as data check, parity error or transmission error.

'9' - Operating System Error Message. The input-output statement was unsuccessfully executed as the result of a condition that is specified by the Operating System. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the values of status key 1 and status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the reults of the input-output operation. This character contains a value as follows:

* If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'

* When status key 1 contains a value of '2' indicating an INVALID KEY condition, status key 2 is used to designate the cause of that condition by the following values:

  2 - Indicates a duplicate key value. An attempt has been made to write a record that would create a duplicate key in a relative file.

  3 - Indicates no record found. An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

  4 - Indicates a boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a relative file. This is normally treated as a fatal error by the Operation System.

* When status key 1 contains a value of '9' the value of status key 2 is the Operating System Error Message number.

* When status key 1 contains a value of '9', the value of status key 2 is the operating system error message number (for those operating systems which designate errors numerically). The Apple III COBOL Introduction and Operating System Manual contains details of the status-key-2 representation.
  Note that it is not possible to extract this number directly.

Valid Combinations of Status Keys 1 and 2

The valid permissible combinations of the values of status key 1 and status key 2 are shown in the table. An 'X' at an intersection indicates a valid permissible combination.

Status Key 1	Status Key 2			
	No Further Information (0)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion (0)	X			
At End (1)	X			
Invalid Key (2)		X	X	X
Permanent Error (3)	X			
Implementor Defined (9)	Operating System Error Message Number			

The INVALID KEY Condition

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE or DELETE statement. For details of the causes of the condition, see The START Statement, The READ Statement, The WRITE Statement, The REWRITE Statement, and The DELETE Statement later in this chapter.

When the INVALID KEY condition is recognised, the Operating System takes these actions in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition. (See I-O Status in this Chapter).

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

3. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognised the condition is unsuccessful, and the file is not affected.

The AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition, see The READ Statement later in this chapter.

ENVIRONMENT DIVISION IN THE RELATIVE I-O MODULE

INPUT-OUTPUT SECTION

The File-Control Paragraph

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information. (See also Appendix F in the Apple III COBOL Introduction and Operating System Manual).

General Format

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

The File Control Entry

Function

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

General Format

```
SELECT file-name

    ASSIGN TO {external-file-name-literal}
              {file-identifier            }
              [, {external-file-name-literal}]
              [  {file-identifier            }]

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

    ; ORGANIZATION IS RELATIVE

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

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

Syntax Rules

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

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

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. Data-name-2 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section, the Report Section, or the Communication Section.

5. Data-name-1 and data-name-2 may be qualified.

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

7. Data-name-1 must not be defined in a record description entry associated with that file-name.

8. The data item referenced by data-name-1 must be defined as an unsigned integer.

General Rules

1. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium. See Appendix F in the Apple III Introduction and Operating System Manual. The first assignment takes place. Subsequent assignments within any one ASSIGN clause are for documentation purposes only.

2. The RESERVE clause allows the user to specify the number of input-output areas allocated. If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-1. The RESERVE clause is treated as for documentation purposes only.

3. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

4. When the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization. This sequence is the order of ascending relative record numbers of existing records in the file.

5. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-2 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement. (See I-O Status in this Chapter).

6. If the access mode is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

7. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. (See General Rules 4 and 6).

8. All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, ...

9. The data item specified by data-name-1 is used to communicate a relative record number between the user and the Operating System.

The I-O-CONTROL Paragraph

Function

The I-O-CONTROL paragraph specifies the points at which rerun is to be established and the memory area which is to be shared by different files.

General Format

```
I-O-CONTROL.

[; RERUN ON {file-name-1     } EVERY {integer-1 RECORDS OF file-name-2 }]
            {implementor-name}       {integer-2 CLOCK-UNITS         }
                                     {condition-name                }

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

Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. File-name-1 must be a sequentially organized file.

3. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, implementor-name must be given in the RERUN clause.

4. More than one RERUN clause may be specified for a given file-name-2, subject to the following restriction:

   When multiple integer-1 RECORDS clauses are specified, no two of them may specify the same file-name-2.

5. Only one RERUN clause containing the CLOCK-UNITS clause may be specified.

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

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

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

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

c. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in the SAME RECORD AREA clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

7. The files referenced in the SAME AREA or SAME RECORD AREA clauses need not all have the same organization or access.

General Rules

1. The RERUN clause is treated as for documentation purposes only.

2. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area i.e., records are aligned on the leftmost character position.

3. The SAME AREA clause specifies that all storage areas of two or more files that do not represent sort or merge files are to use the same memory area during processing. The area being shared includes all storage areas (including alternate areas) assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time.

DATA DIVISION IN THE RELATIVE I-O MODULE

FILE SECTION

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

RECORD DESCRIPTION STRUCTURE

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A record description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in CONCEPTS OF LEVELS in Chapter 2 while the elements allowed in a record description are shown in the DATA DESCRIPTION-COMPLETE ENTRY SKELETON in Chapter 3.

THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

Function

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

General Format

```
FD   file-name

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

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

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

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

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

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

Syntax Rules

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.

2. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial. All clauses are optional when the ANSI switch is unset.

3. One or more record description entries must follow the file description entry.

THE BLOCK CONTAINS CLAUSE

Function

The BLOCK CONTAINS clause specifies the size of a physical record.

General Format

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

General Rule

This clause is required for documentation purposes only.

THE DATA RECORDS CLAUSE

Function

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

General Format

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

Syntax Rule

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

General Rule

The DATA RECORDS clause is specified for documentation purposes only.

THE LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present.

General Format

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

Syntax Rule

This clause is required in every file description entry, when the ANSI switch is set.

General Rule

This clause is used for documentation purposes only.

THE RECORD CONTAINS CLAUSE

Function

The RECORD CONTAINS clause specifies the size of data records.

General Format

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

General Rule

The size of each data record is completely defined within the record description entry, therefore this clause is never required.

The RECORD CONTAINS clause is specified for documentation purposes only.

THE VALUE OF CLAUSE

Function

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

General Format

```
VALUE OF data-name-1 IS {data-name-1}
                        {literal-1   }

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

Syntax Rules

1. Data-names should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause

2. Data-name-2, data-name-4 etc, must be in the Working-Storage Section

General Rules

1. This clause is for documentation purposes only.

   The compiler checks that data-name-1 matches in value data-name-2 or literal-1, data-name-3 matches in value data-name-4 or literal-2, etc, for input files. For output files the value of data-name-2 or literal-1 is substituted for data-name-1, the value of data-name-4 or literal-2 is substituted for data-name-3, etc.

2. A figurative constant may be substituted in the format above wherever a literal is specified.

PROCEDURE DIVISION IN THE RELATIVE I-O MODULE

THE CLOSE STATEMENT

Function

The CLOSE statement terminates the processing of files. The LOCK is for documentary purposes only.

General Format

```
CLOSE file-name-1  [WITH LOCK]  [ ,file-name-2  [WITH LOCK] ]  ...
```

Syntax Rule

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

General Rules

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

2. The action taken if a file is in the open mode when a STOP RUN statement is executed is to close the file. The action taken for a file that has been opened in a called program and not closed in that program prior to the execution of a CANCEL statement for the program is to close the file.

3. If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

4. Following the successful execution of a CLOSE statement, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

THE DELETE STATEMENT

Function

The DELETE statement logically removes a record from a mass storage file.

General Format

```
DELETE file-name RECORD [;INVALID KEY imperative-statement]
```

Syntax Rules

1. The INVALID KEY phrase must not be specified for a DELETE statement which references a file which is in sequential access mode.

2. The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

General Rules

1. The associated file must be open in the I-O mode at the time of the execution of this statement. (See THE OPEN STATEMENT later in this Chapter)

2. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The Operating System logically removes from the file the record that was accessed by that READ statement.

3. For a file in random or dynamic access mode, the Operating System logically removes from the file that record identified by the contents of the RELATIVE KEY data item associated with file-name. If the file does not contain the record specified by the key, an INVALID key condition exists. (See The INVALID KEY Condition in this Chapter).

4. After the succesful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

5. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

6. The current record pointer is not affected by the execution of a DELETE statement.

7. The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with the file-name to be updated. See I-O STATUS in this chapter.

THE OPEN STATEMENT

Function

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

General Format

```
                ⎧INPUT  file-name-1    [,file-name-2 ...]⎫
OPEN            ⎨OUTPUT file-name-3    [,file-name-4 ...]⎬ ...
                ⎩I-O    file-name-5    [,file-name-6 ...]⎭
```

Syntax Rule

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

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The successful execution of the OPEN statement makes the associated record area available to the program.

3. Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file, either explicitly or implicitly.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 6-1, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the relative file organisation and the open mode given at the top of the column.

Table 6-1. Permissible Combinations of Statements and OPEN Modes for Relative I-O.

File Access Mode	Statement	Open Mode		
		Input	Output	Input-Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

5. A file may be opened with the INPUT, OUTPUT, AND I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent execution for that same file must be preceded by the execution of a CLOSE statement, for that file.

6. Execution of the OPEN statement does not obtain or release the first data record.

7. The file description entry for file-name-1, file-name-2, file-name-5 or file-name-6 must be equivalent to that used when this file was created.

8. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed Format 1 READ statement for the file will result in an AT END condition. If the file does not exist, INPUT will cause an error status.

9. The I-O phrase permits the opening of a file for both input and output operations. If the file does not exist, it will be created.

10. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At the time the associated file contains no data records. If a file of the same number exists it will be deleted. If it is write protected, an error status occurs.

11. The ASSIGNed name in the SELECT statement for a file is processed as follows:

    a. When the INPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNed name to be checked in accordance with the operating system conventions for opening files for input.

    b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the ASSIGNed name to be written in accordance with the operating system conventions for opening files for output.

THE READ STATEMENT

Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a disk file.

General Format

Format 1

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

Format 2

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

Syntax Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

2. Format 1 must be used for all files in sequential access mode.

3. The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

4. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

5. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

General Rules

1. The associated files must be open in the INPUT or I-O mode at the time this statement is executed. See THE OPEN STATEMENT in this Chapter.

2. The record to be made available by a Format 1 READ statement is determined as follows:

   a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, which may have been caused by the deletion of the record, the current record pointer is updated to point to the next existing record in the file and that record is then made available.

b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See I-O Status in this Chapter).

4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available to the object program prior to the execution of any statement following the READ statement.

5. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalant to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

7. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

8. If, at the time of execution of a Format 1 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

9. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful. (See I-O Status in this Chapter).

10. When the AT END condition is recognized the following actions are taken in the specified order:

a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. (See I-O Status in this Chapter)

b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file, and that procedure is executed.

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

11. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

12. When the AT END condition has been recognised, a Format 1 READ statement for that file must not be executed without first executing one of the following:

a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

b. A successful START statement for that file.

c. A successful Format 2 READ statement for that file.

13. For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from the file as described in general rule 2.

14. If the RELATIVE KEY phrase is specified, the execution of a Format 1 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

15. The execution of a Format 2 READ statement sets the current record pointer to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file, If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. (See The INVALID KEY Condition in this Chapter).

THE REWRITE STATEMENT

Function

The REWRITE statement logically replaces a record existing in a disk file.

General Format

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

Syntax Rules

1. Record-name and identifier must not refer to the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division, and may be qualified.

3. The INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an appropriate USE procedure is not specified.

General Rules

1. The file associated with record-name must be open in the I-O mode at the time of execution of this statement. (See THE OPEN STATEMENT in this Chapter.)

2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The Operating System logically replaces the record that was accessed by the READ statement.

3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause, in which case the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

```
MOVE identifier TO record-name
```

followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See I-O STATUS in this Chapter).

8. For a file accessed in either random or dynamic access mode, the Operating System logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists. (See THE INVALID KEY CONDITION in this Chapter). The updating operation does not take place and the data in the record area is unaffected.

THE START STATEMENT

Function

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

General Format

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

[; INVALID KEY imperative-statement]

NOTE: The required relational characters '>', and '<' and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

Syntax Rules

1. File-name must be the name of a file with sequential or dynamic access.

2. Data-name may be qualified.

3. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

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

General Rules

1. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed. (See THE OPEN STATEMENT in this Chapter).

2. If the KEY phrase is not specified the relational operator 'IS EQUAL TO' is implied.

3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in general Rule 5.

   a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. (See The INVALID KEY Condition in this Chapter).

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See I-O STATUS in this Chapter).

5. The comparison described in general rule 3 uses the data item referenced by the RELATIVE KEY clause associated with file-name.

THE USE STATEMENT

Function

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

General Format

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

Syntax Rules

1. A USE statement, when present, must immediately follow a section header in the declaratives section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedures to be used.

2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

3. The same file-name can appear in a different specific arrangement of the format. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

5. The files implicitly or explicitly referred in a USE statement need not all have the same organization or access.

General Rules

1. If the INVALID KEY or AT END phrases have not been specified in the input-output statement, the designated procedures are executed by the input-output system after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions.

2. After execution of a USE procedure, control is returned to the invoking routine.

3. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

4. Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

THE WRITE STATEMENT

Function

The WRITE statement releases a logical record for an output or input-output file.

General Format

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

Syntax Rules

1. Record-name and identifier must not reference the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division, and may be qualified.

3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement. (See THE OPEN STATEMENT Chapter.)

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition.

   The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of

   a. The statement:

      MOVE identifier TO record-name

      according to the rules specified for the MOVE statement, followed by:

b. The same WRITE statement without the FROM phrase.

The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be. (See general rule 2 above.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See I-O Status in this Chapter.)

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

8. The execution of the WRITE statement releases a logical record to the operating system.

9. When a file is opened in the output mode, records may be placed into the file by one of the following:

a. If the access mode is sequential, the WRITE statement will cause a record to be released to the Operating System. The first record will have a relative record number of 1 and subsequent records released will have relative record numbers of 2, 3, 4, ... If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item by the Operating System during execution of the WRITE statement.

b. If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the RELATIVE KEY data item must be initialized in the program with the relative record number of be associated with the record in the record area. That record is then released to the Operating System by execution of the WRITE statement.

10. When a file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialised by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the Operating System.

11. The INVALID KEY condition exists under the following circumstances:

    a. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record which already exists in the file, or

    b. When an attempt is made to write beyond the externally defined boundaries of the file.

12. When the INVALID KEY condition is recognised, the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the FILE STATUS data item, if any, of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated in The INVALID KEY Condition in this Chapter (see also I-O Status in this Chapter).

Chapter 7
Indexed Input and Output

INTRODUCTION TO THE INDEXED I-O MODULE

The Indexed I-O module provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of one or more keys within that record.

LANGUAGE CONCEPTS

Organization

A file whose organization is indexed is a mass storage file in which data records may be accessed by the value of a key. A record description may include one or more key data items, each of which is associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the record key for that index.

The data item named in the RECORD KEY clause of the file control entry for a file is the prime record key for that file. For purposes of inserting, updating and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique and must not be changed when updating the record.

A data item named in the ALTERNATE RECORD KEY clause of the file control entry for a file is an alternative record key for that file. The value of an alternative record key may be non-unique if the DUPLICATES phrase is specified for it. These keys provide alternative access paths for retrieval of records from the file.

Access Modes

In the sequential access mode, the sequence in which records are accessed is the ascending order of the record key values. The order of retrieval of records within a set of records having duplicate record key values is the orer in which the records were written into the set.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing the value of its record key in the record key data item.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened only in the output mode. The setting of the current record pointer is affected only by the OPEN, START and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation.

'0' - Successful Completion
'1' - At End
'2' - Invalid Key
'3' - Permanent Error
'9' - Operating System Error Message

The meaning of the above indications are as follows:

0 - Successful Completion. The input-output statement was successfully executed.

1 - At End. The Format 1 READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

2 - Invalid Key. The input-output statement was unsuccessfully executed as a result of one of the following:

Sequence Error
Duplicate Key
No Record Found
Boundary Violation

3 - Permanent Error. The input-output statement was unsuccessful as the result of an input-output error, such as data check, parity error, or transmission error.

9 - Operating System Error Message. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the Operating System Error Message number. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the value of status key 1 and status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character will contain a value as follows:

If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

When status key 1 contains a value of '0' indicating a successful completion, status key 2 may contain a value of '2' indicating a duplicate key. This condition indicates one of two possibilities:

1. For a READ statement, the key value for the current key of reference is equal to the value of that same key in the next record within the current key of reference.

2. For a WRITE or REWRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

3. When status key 1 contains a value of '2' indicating an INVALID KEY condition, status key 2 contains values to designate the cause of that condition as follows:

   1 Indicates a sequence error for a sequentially accessed indexed file. The ascending sequence requirements of successive record key values have been violated (see The WRITE Statement later in this Chapter), or the prime record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.

   2 Indicates a duplicate key value. An attempt has been made to write or rewrite a record that would create a duplicate key in an indexed file.

   3 Indicates no record found. An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

   4 Indicates a boundary violation. An attempt has been made to write beyond the externally defined boundaries of an indexed file. This is usually treated as a fatal error by Operating Systems.

When status key 1 contains a value of '9' the value of status key 2 is the operating system error message number (for those operating systems which designate errors numerically). The Apple III COBOL Introduction and Operating System Manual contains details of the status-key-2 representation. Note that it is not possible to extract this number directly.

Valid Combinations of Status Keys 1 and 2

The valid permissible combinations of the value of status key 1 and status key 2 are shown in the following table. An 'X' at an intersection indicates a valid permissible combination.

Status Key 1	Status Key 2				
	No Further Information (0)	Sequence Error (1)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion (0)	X				
At End (1)	X				
Invalid Key (2)		X	X	X	X
Permanent Error (3)	X				
Implementor Defined (9)	Operating System Error Message Number				

The INVALID KEY Condition

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE or DELETE statement. For details of the causes of the condition see THE START STATEMENT, THE READ STATEMENT, THE WRITE STATEMENT, and THE DELETE STATEMENT later in this Chapter.

When the INVALID KEY condition is recognized, the Operating System takes these actions in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition. (See I-O Status).

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

3. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected.

The AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. For details of the causes of the condition, see THE READ STATEMENT later in this Chapter.

ENVIRONMENT DIVISION IN THE INDEXED I-O MODULE

INPUT-OUTPUT SECTION

The File Control Paragraph

Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information. (See also Appendix F in the Apple III COBOL Introduction and Operating System Manual).

General Format

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

The File Control Entry

Function

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

General Format

```
SELECT file-name

     ASSIGN TO {external-file-name-literal}
               {file-identifier           }
          [, {external-file-name-literal}]
             {file-identifier           }

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

 ; ORGANIZATION IS INDEXED

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

 ; RECORD KEY IS data-name-1

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

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

Syntax Rules

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

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

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. Data-name-3 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section.

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

6. The data items referenced by data-name-1 and data-name-2 must each be defined as a data item of the category alphanumeric within a record description entry associated with that file-name.

7. Neither data-name-1 nor data-name-2 can describe an item whose size is variable. (See THE OCCURS CLAUSE in Chapter 4.)

8. Data-name-2 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name-1 or by any other data-name-2 associated with this file.

General Rules

1. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium. See Appendix F in the Apple III COBOL Introduction and Operating System Manual. The first assignment takes effect. Subsequent assignments within any one ASSIGN clause are for documentation purposes only.

2. The RESERVE clause allows the user to specify the number of input-output areas allocated. If the RESERVE clause is specified, the number of input-output areas allocated is equal to the value of integer-1.

3. The ORGANIZATION clause specifies the logical structure of a file. The File organization is established at the time a file is created and cannot subsequently be changed.

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

5. When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-3 after the

execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement. (See I-O STATUS in this Chapter).

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

7. When the access mode is dynamic, records in the file may be accessed sequentially and/or randomly. (See general rules 4 and 6).

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

9. An ALTERNATE RECORD KEY clause specifies a record key that is an alternative record key for the file. This alternate record key provides an alternate access path to records in an indexed file.

10. The data description of data-name-1 and data-name-2 as well as relative locations within a record must be the same as that used when the file was created. The number of alternate keys for the file must also be the same as that used when the file was created.

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

The I-O Control Paragraph

Function

The I-O-CONTROL paragraph specifies the points at which rerun is to be established and the memory area which is to be shared by different files.

General Format

I-O-CONTROL

```
[; RERUN ON {file-name-1      } EVERY {integer-1 RECORDS OF file-name-2}]
            {implementor-name}       {integer-2 CLOCK-UNITS         }
                                     {condition-name                }

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

Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. File-name-1 must be a sequentially organized file.

3. When either the integer-1 RECORDS clause or the integer-2 CLOCK-UNITS clause is specified, implementor-name must be given in the RERUN clause.

4. When multiple integer-1 RECORDS clauses are specified, no two of them may specify the same file-name-2.

5. Only one RERUN clause containing the CLOCK-UNITS clause may be specified.

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

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

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

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

   c. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in the SAME AREA clause must appear in the SAME RECORD AREA clause. However, additional file-names not appearing in that SAME AREA clause may also appear in that SAME RECORD AREA clause. The rule that only one of the files mentioned in a SAME AREA clause can be open at any given time takes precedence over the rule that all files mentioned in a SAME RECORD AREA clause can be open at any given time.

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

General Rules

1. The RERUN clause is treated as for documentation purposes only.

2. The SAME AREA clause specifies that two or more files that do not represent sort or merge files are to use the same memory area during processing. The area shared includes all storage areas assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time. (See Syntax Rule 6c.)

4. The SAME RECORD AREA clause specifies that two or more files are to use the same memory area for processing of the current logical record. All of the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and of the most recently read input file whose file-name appears in this SAME RECORD AREA clause. This is equivalent to an implicit redefinition of the area, i.e., records are aligned on the leftmost character position.

DATA DIVISION IN THE INDEXED I-O MODULE

FILE SECTION

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

RECORD DESCRIPTION STRUCTURE

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name if required, followed by a series of independent clauses as required. A record description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in CONCEPTS OF LEVELS in Chapter 2 while the elements allowed in a record description are shown in THE DATA DESCRIPTION - COMPLETE ENTRY SKELETON in Chapter 3.

THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

Function

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

General Format

```
FD file-name

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

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

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

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

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

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

Syntax Rules

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.

2. The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial. All clauses are optional when the ANSI switch is set on.

3. One or more record description entries must follow the file description entry.

THE BLOCK CONTAINS CLAUSE

Function

The BLOCK CONTAINS clause specifies the size of a physical record.

General Format

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

General Rule

This clause is required for documentation purposes only.

THE DATA RECORDS CLAUSE

Function

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

General Format

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

Syntax Rule

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

General Rules

1. The presence of more than one data-name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

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

THE LABEL RECORDS CLAUSE

Function

The LABEL RECORDS clause specifies whether labels are present.

General Format

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

General Rule

This clause is used for documentation purposes only.

THE RECORD CONTAINS CLAUSE

Function

The RECORD CONTAINS clause specifies the size of data records.

General Format

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

General Rule

The size of each data record is completely defined within the record description entry, therefore this clause is never required.

THE VALUE OF CLAUSE

Function

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

General Format

```
VALUE OF implementor-name-1 IS { data-name-1 }
                               { literal-1   }

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

Syntax Rules

1. Data-name-1, data-name-2, etc., should be qualified when necessary, but cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

2. Data-name-1, data-name-2, etc., must be in the Working-Storage Section.

General Rules

1. For an input file, the appropriate label routine checks to see if the value of implementor-name-1 is equal to the value of literal-1, or of data-name-1, whichever has been specified.

   For an output file, at the appropriate time the value of implementor-name-1 is made equal to the value of literal-1, or of a data-name-1, whichever has been specified.

2. A figurative constant may be substituted in the format above wherever a literal is specified.

3. This clause is required for documentation purposes only.

PROCEDURE DIVISION IN THE INDEXED I-O MODULE

THE CLOSE STATEMENT

Function

The CLOSE statement terminates the processing of files. The LOCK phrase is for documentation purposes only.

General Format

```
CLOSE file-name-1  [WITH LOCK] [, file-name-2  [WITH LOCK]  ...
```

Syntax Rule

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

General Rules

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

2. The action taken if a file is in the open mode when a STOP RUN statement is executed is to close the file. The action taken for a file that has been opened in a called program and not closed in that program prior to the execution of a CANCEL statement for that program is to close the file.

3. If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

4. Following the successful execution of a CLOSE statement, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

THE DELETE STATEMENT

Function

The DELETE statement logically removes a record from a mass storage file.

General Format

DELETE file-name RECORD [; INVALID KEY imperative-statement]

Syntax Rules

1. The INVALID KEY phase must not be specified for a DELETE statement which references a file which is in sequential access mode.

2. The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

General Rules

1. The associated file must be open in I-O mode at the time of the execution of this statement. (See THE OPEN STATEMENT later in this Chapter.)

2. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The Operating System logically removes from the file the record that was accessed by that READ statement.

3. For a file in random or dynamic access mode, the Operating System logically removes from the file the record identified by the contents of the prime record key data item associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists. (See THE INVALID KEY CONDITION in this Chapter.)

4. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

5. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

6. The current record pointer is not affected by the execution of a DELETE statement.

7. The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated. (See I-O STATUS in this Chapter.)

THE OPEN STATEMENT

Function

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

General Format

```
                 ⎧ INPUT file-name-1     [,file-name-2] ... ⎫
OPEN             ⎨ OUTPUT file-name-3    [,file-name-4] ... ⎬ ...
                 ⎩ I-O file-name-5       [,file-name-6] ... ⎭
```

Syntax Rule

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

General Rules

1. The successful execution of the OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The successful execution of the OPEN statement makes the associated record area available to the program.

3. Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file, either explicitly or implicitly.

4. An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In Table 7-1, Permissible Statements, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the indexed file organization and the open mode given at the top of the column.

Table 7-1. Permissible Combinations of Statements and OPEN Modes for Indexed I-O.

File Access Mode	Statement	Open Mode		
		Input	Output	Input-Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	WRITE		X	X
	REWRITE		X	X
	START	X		X
	DELETE			X

5. A file may be opened with the INPUT, OUTPUT and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement for that file.

6. Execution of the OPEN statement does not obtain or release the first data record.

7. The assigned name in the select statement for a file is processed as follows:

   a. When the INPUT phrase is specified, the execution of the OPEN statement causes the assigned name to be checked in accordance with the operating system conventions for opening files for input.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the assigned name to be written in accordance with the operating system conventions for opening files for output.

8. The file description entry for file-name-1, file-name-2, file-name-5, or file-name-6 must be equivalent to that used when this file was created.

9. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. For indexed files, the prime record is established as the key of reference and is used to determine the first record to be accessed. If no records exist in the file, the current record pointer is set such that the next executed Format 1 READ statement for the file will result in an AT END condition. If the file does not exist, INPUT will cause an error status.

10. The I-O phrase permits the opening of a file for both input and output operations. If the file does not exist, it will be created.

11. Upon successful execution of an OPEN statement with the output phrase specified, a file is created. At that time the associated file contains no data records. If a file of the same name exists it will be deleted. If write protected, an error status occurs.

THE READ STATEMENT

Function

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

General Format

Format 1

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

  [;AT END imperative-statement]
```

Format 2

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

  [;KEY IS data-name]

  [;INVALID KEY imperative-statement]
```

Syntax Rules

1. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the storage area which is the record area associated with file-name must not be the same storage area.

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

3. Data-name may be qualified.

4. Format 1 must be used for all files in sequential access mode.

5. The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

6. Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

7. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

General Rules

1. The associated file must be open in the INPUT or I-O mode at the time this statement is executed. (See THE OPEN STATEMENT in this Chapter).

2. The record to be made available by a Format 1 READ statement is determined as follows:

   a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, which may have been caused by the deletion of the record or a change in an ALTERNATE RECORD key. The current record pointer is updated to point to the next existing record within the established key of reference and that record is then made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference and then that record is made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See I-O Status in this Chapter).

4. Regardless of the method used to overlap access time with processing time, the concept of the READ statement is unchanged in that a record is available to the object program prior to the execution of any statement following the READ statement.

5. When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

6. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

   When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

8. If, at the time of execution of a Format 1 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

9. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful. (See I-O Status in this Chapter).

10. When the AT END condition is recognised the following actions are taken in the specified order:

    a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. (See I-O STATUS in this Chapter).

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative statement. Any USE procedure specified for this file is not executed.

    c. If the AT END phrase is not specified, then a USE procedure must be specified, either explictly or implicitly, for this file, and that procedure is executed.

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

11. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined. For indexed files the key of reference is also undefined.

12. When the AT END condition has been recognised, a Format 1 READ statement for that file must not be executed without first executing one of the following:

    a. A succesfful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful START statement for that file.

    c. A successful Format 2 READ statement for that file.

13. For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in general rule 2 above.

14. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of rewrite statements which create such duplicate values.

15. If the KEY phrase is not specified in a Format 2 READ statement, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statement for the file.

16. For an indexed file if the KEY phrase is specified in a Format 2 READ statement, data-name is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for the file.

17. Execution of a Format 2 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. The current record pointer is positioned to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful. (See The INVALID KEY Condition in this Chapter).

THE REWRITE STATEMENT

Function

The REWRITE statement logically replaces a record existing in a mass storage file.

General Format

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

Syntax Rules

1. Record-name and identifier must not refer to the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division, and may be qualified.

3. The INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

General Rules

1. The file associated with record-name must be open in the I-O mode at the time of execution of this statement. (See THE OPEN STATEMENT in this Chapter).

2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The Operating System logically replaces the record that was accessed by the READ statement.

3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause, in which case the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

```
MOVE identifier TO record-name
```

followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See I-O Status).

8. For a file in the sequential access mode, the record to be replaced is specified by the value contained in the prime record key. When the REWRITE statement is executed the value contained in the prime record key data item of the record to be replaced must be equal to the value of the prime record key of the last record read from this file.

9. For a file in the random or dynamic access mode, the record to be replaced is specified by the prime record key data item.

10. The contents of alternative record key data items of the record being rewritten may differ from those in the record being replaced. The Operating System utilizes the content of the record key data items during the execution of the REWRITE statement in such a way that subsequent access of the record may be made based upon any of those specified record keys.

11. The INVALID KEY condition exists when:

    a. The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record key of the last record read from this file or,

    b. The value contained in the prime record key data item does not equal that of any record stored in the file, or

    c. The value contained in an alternate record key data item for which a DUPLICATES clause has not been specified is equal to that of a record already stored in the file.

    The updating operation does not take place and the data in the record area is unaffected. (See The INVALID KEY Condition in this Chapter).

THE START STATEMENT

Function

The START statement provides a basis for logical positioning within an indexed file, for subsequent sequential retrieval of records.

General Format

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

    ;INVALID KEY imperative-statement
```

NOTE: The required relational characters '>', '<' and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

Syntax Rules

1. File-name must be the name of an indexed file.

2. File-name must be the name of a file with sequential or dynamic access.

3. Data-name may be qualified.

4. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

5. If file-name is the name of an indexed file, and if the KEY phrase is specified, data-name may reference a data item specified as a record key associated with file-name, or it may reference any data item of category alpanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.

General Rules

1. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed. (See THE OPEN STATEMENT in this Chapter).

2. If the KEY phrase is not specified the relational operator 'IS EQUAL TO' is implied.

3. The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item as specified in general rule 5. If file-name references an indexed file and the operands are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE clause will have no effect on the comparison. (See Comparison of Nonnumeric Operands).

   a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

   b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined. (See The INVALID KEY Condition in this Chapter)

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See I-O Status).

5. If the KEY phrase is specified, the comparison described in general rule 3 uses the data item referenced by data-name.

6. If the KEY phrase is not specified, the comparison described in general rule 3 uses the data item referenced in the RECORD KEY clause associated with file-name.

7. Upon completion of the successful execution of the START statement, a key of reference is established and used in subsequent Format 1 READ statements as follows: (See THE READ STATEMENT in this Chapter).

   a. If the KEY phrase is not specified, the prime record key specified for file-name becomes the key of reference.

   b. If the KEY phrase is specified, and data-name is specified as a record key for file-name, that record key becomes the key of reference.

   c. If the KEY phrase is specified, and data-name is not specified as a record key for file-name, the record key whose leftmost character position corresponds to the leftmost character position of the data item specified by data-name, becomes the key of reference.

8. If the execution of the START statement is not successful, the key of reference is undefined.

THE USE STATEMENT

Function

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

General Format

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

Syntax Rules

1. A USE statement, when present, must immediately follow a section header in the declaratives section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedures to be used.

2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

3. The same file-name can appear in a different specific arrangement of the format. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

5. The files implicitly referenced in a USE statement need not all have the same organization or access.

General Rules

1. If the INVALID KEY phrase on the AT END phrase have not been specified in the input-output statements the designated procedures are executed by the input-output system after completing the standard input-output routine upon recognition of the INVALID KEY or AT END condition.

2. After execution of a USE procedure, controlis returned to the invoking routine.

3. Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

4. Within a USE procedure, there must not be the excecution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

THE WRITE STATEMENT

Function

The WRITE statement releases a logical record for an output or input-output file.

General Format

WRITE record-name FROM identifier ; INVALID KEY imperative-statement

1. Record-name and identifier must not reference the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division, and may be qualified.

3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement. (See THE OPEN STATEMENT in this Chapter).

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition. The logical record is available to the program from the file associated with record-name and from other files referenced in the same SAME RECORD AREA clause as the associated output file.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of:

   a. The statement:

      MOVE identifier TO record-name

      according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

      The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be. (See general rule 2 above).

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. (See I-O Status in this Chapter).

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

8. The execution of the WRITE statement releases a logical record to the operating system.

9. Execution of the WRITE statement causes the contents of the record area to be released. The Operating System utilizes the content of the record keys in such a way that subsequent access of the record may be made based upon any of those specified record keys.

10. The value of the prime record key must be unique within the records in the file.

11. The data item specified as the prime record key must be set by the program to the desired value prior to the execution of the WRITE statement.

12. If sequential access mode is specified for the file, records must be released to the Operating System in ascending order of prime record key values.

13. If random or dynamic access mode is specified, records may be released to the Operating System in any program-specified order.

14. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the Operating System provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the Operating System.

15. The INVALID KEY condition exists under the following circumstances:

    a. When sequential access mode is specified for a file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, or

    b. When the file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file, or

    c. When the file is opened in the output or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file, or

    d. When an attempt is made to write beyond the externally defined boundaries of the file.

16. When the INVALID KEY condition is recognised the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected and the FILE STATUS data item, if any, associated with file-name of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated under THE INVALID KEY CONDITION (see also I-O Status in this Chapter).

Figures and Tables
Volume 1

12	Figure 1-1	Sample Program Listing Showing Source Format
42	Figure 2-1	Reference Format for a COBOL Source Line
20	Table 2-1	Figurative Constants and their Reserved Words
23	Table 2-2	Data Levels, Classes and Categories
24	Table 2-3	Numeric Data Storage for the COMP(UTATIONAL) PICTURE Clause
25	Table 2-4	Numeric Data Storage for the COMPUTATION-3 PICTURE Clause
138	Figure 3-1	Flowchart for VARYING Phrase of a PERFORM Statement Having One Condition
139	Figure 3-2	Flowchart for VARYING Phrase of PERFORM Statement with Two Conditions
140	Figure 3-3	Flowchart for VARYING Phrase of PERFORM Statement with Three Conditions
141	Figure 3-4	PERFORM Statements in Sequence
66	Table 3-1	Editing Types for Data Categories
67	Table 3-2	Editing Symbols in PICTURE Character-Strings
70	Table 3-3	PICTURE Character Precedence Chart
85	Table 3-4	Combination of Symbols in Arithmetic Expressions
87	Table 3-5	Relational Operators
92	Table 3-6	Combinations of Conditions, Logical Operations and Parentheses
102	Table 3-7	Cursor Repositioning Keys
131	Table 3-8	MOVE Statement Data Categories
161	Figure 4-1	Flowchart of SEARCH Operation with Two WHEN Phrases
164	Table 4-1	SET Statement Valid Operand Combinations
183	Table 5-1	Permissible Combinations of Statements and OPEN Modes for Sequential I-O

214 Table 6-1 Permissible Combinations of Statements and OPEN Modes for Relative I-O

247 Table 7-1 Permissible Combinations of Statements and OPEN Modes for Indexed I-O

Figures and Tables
Volume 2

51 Table 13-1 Communication Status Key Condition

67 Table 14-1 Data Dictionary Entry Sizing

Index

Pages in Volume 2 are shown in [].

A

Abbreviated Combined Relation Conditions 92
ACCEPT MESSAGE COUNT Statement [52]
ACCEPT Statement 99
Access Mode 165, 197, 229
ADD Statement 104
Algebraic Signs 26
Alignment Rules, Standard 26
Alphabetic Data Rules 62
Alphanumeric Data Rules 63
Alphanumeric Edited Data Rules 63
ALTER Statement 106, [22]
ANSI (ANS) Compiler Directive 33
Area, Indicator 11
Arithmetic Expressions 84
Arithmetic Operators 84
Arithmetic Statements 96
ASSIGN Clause 169, 202, 235
AT END Condition 167, 200, 233
Attributes, Explicit and Implicit 33

B

Blank Lines 43
BLANK WHEN ZERO Clause 58
BLOCK CONTAINS Clause 174, 207, 240
Body, Procedure Division 39

C

CALL Statement [36]
CANCEL Statement [38]
Character Representation 24
Character Sets 13
Character Strings 15
Character Strings, PICTURE 21
Characteristics, Name 47
Class Condition 88
Classes of Data, Concepts 23
Classification, Segmentation [20]
Clause,
  ASSIGN 169, 202, 235
  BLANK WHEN ZERO 58
  BLOCK CONTAINS 174, 207, 240
  CODE-SET 174
  CURSOR IS 51
  DATA-NAME or FILLER 59
  DATA RECORDS 174, 208, 241, [4]
  FILE STATUS 169, 202, 235
  JUSTIFIED 60
  LABEL RECORDS 165, 208, 241
  OCCURS 153
  ORGANIZATION 169, 202, 235
  PICTURE 62
  RECORD CONTAINS 179, 209, 242, [5]
  RECORD KEY 236
  REDEFINES 72
  RENAMES 74
  SELECT 168, 201, 234
  SIGN 76
  SYNCHRONIZED 78
  USAGE 80, 156
  VALUE 81
  VALUE OF 179, 210, 242
  WITH DEBUGGING MODE [28]
CLOSE Statement 181, 211, 244
COBOL, Apple III 7
COBOL Words 15
CODE-SET Clause 174
Combined and Negated Simple Conditions 91
Comment Entries 21
Comment Lines 45
Communication Description, Entry Skeleton [41]
Communication Module [41]
Communication Section [41]
Communication,
  Data Division in [41]
  Procedure Division in [52]
COMP(UTATIONAL) (-3) PICTURE Clause 25
Comparison Involving Index Names and/or Index Data Items 156
Comparison of Nonnumeric

Operands 87
Comparison of Numeric Operands 87
Compile Time Debug Switch [28]
Compiler Directives, "ANSI Switch" 33
COMPLEX Conditions 90
COMPUTE Statement 107
Computer Independent Data Description, Concept of 21
Concept, Classes of Data 23
Concepts,
Computer Independent Data Description 20
Language 13
Levels 22
Condition Evaluation Rules 93
Condition-Name 16, 30
Condition-Name Condition 89
Condition-Name Rules 82
Conditional Expressions 86
Conditions,
Abbreviated Combined Relation 92
AT END 167, 200, 233
Class 88
Complex 90
INVALID KEY 199, 232
Negated Simple 91
Relation 86, 90
Sign 90
Simple 86
Switch-Status 89
CONFIGURATION SECTION 50
Connectives 18
Constants, Figurative 18, 47
Continuation of Lines 43
COPY Statement [26]
CORRESPONDING Phrase 96
CRT Devices 98
Current Record Pointer 165, 197, 229
CURSOR IS Clause 51

D

Data Description,
Computer Independent, Concept of 21
Entries Other than Condition-Names 82
Entry Skeleton 55
Data Dictionary [66]
Data Division Entries 44
Data Division in Communications [41]
Data Division in Indexed I-O Module 239
Data Division in Interprogram Communication Module [33]
Data Division in Nucleus 55
Data Division in Relative I-O Module 206
Data Division in Sequential I-O Module 172
Data, Incompatible 97
DATA-NAME or FILLER Clause 59
DATA RECORDS Clause 174, 208, 241, [4]
DATE-COMPILED Paragraph 49
Debug [27]
DEBUG,
Environment Division in [28]
Object Time Switch [28]
Procedure Division in [29]
Debug, Run Time [27]
Debugging Lines [27]
Declaratives 38
DELETE Statement 212, 245
DISABLE Statement [53]
DISPLAY Statement 108
DIVIDE Statement 111
Division Format 44
Division Header 44

E

Editing Symbols 64
Editing Types for Data Categories 63
Elements 10
ENABLE Statement [55]
ENTER Statement 114
Entries, Comment 21
Entry,
Communication Description [41]
FILE-CONTROL 168, 201, 234, [1]
Environment Division in COBOL Debug [28]
Environment Division in Indexed I-O Module 234
Environment Division in

Nucleus 50
Environment Division in Relative I-O Module 201
Environment Division in Sequential I-O Module 168
Environment Division in Sort-Merge Module [1]
Evaluation Rules 84
Execution, Procedure Division 39
EXIT PROGRAM Statement [39]
EXIT Statement 115
Explicit Specifications 31
Expressions,
Arithmetic 84
Conditional 86
Extra Intermediate Code Files 22

F

Figurative Constant Values 19
Figurative Constants 18, 47
File Description Entry Skeleton 173, 206, 240, [4]
FILE SECTION 172, 206, 239, [4]
FILE STATUS Clause 169, 202, 235
FILE-CONTROL Entry 168, 201, 234, [1]
FILE-CONTROL Paragraph 168, 201, 234, [1]
FILLER or DATA-NAME Clause 59
Fixed Insertion Editing Rules 67
Fixed Portion [19]
Format, Reference 47
Formats,
Division 44
General 10
Paragraph 44
Reference 42
Section 44
Source 11
Formulation Rules 84

G

General Formats 10
GO TO Statement 116

H

Header,
Division 44
Paragraph 44
Procedure Division 44
Section 44
Hexadecimal Characters 19
Hints, Useful [65]

I

I-O Control Paragraph 170, 204, 235, [2]
Identification Division 34
Identification Division, in Nucleus 48
Identifier 30
IF Statement 118
Implicit Specifications 31
Independent Segments [19]
Index Data Items 156
Index-Names 156
Indexed I-O Module 229
Indexed I-O Module,
Data Division in 239
Environment Division in 234
Procedure Division in 244
Indexing 29
Indicator Area 10
Input-Output Section 168, 201, 234, [1]
Input-Output Status 165, 201, 230
Insertion Editing Rules,
Fixed 67
Floating 68
Simple 67
Special 67
INSPECT Statement 120
Inter Program Communication Module,
Data Division in [33]
Procedure Division in [35]
Intermediate Code Files, Extra [22]
INVALID KEY Condition 199, 232

J

JUSTIFIED Clause 60

K

Keys, Status 165, 198, 230

L

LABEL RECORDS Clause 175, 208, 241
Language Concepts 13, 165, 197, 229
Language, Overall 47
Language Structure 13
Levels,
  Concept 22
  Number 22, 61
Library Module [25]
LINE SEQUENTIAL Organization 168
Lines,
  Blank 43
  Comment 45
  Continuation of 43
  Debugging [27]
Linkage Section [33]
Literals,
  Nonnumeric 18
  Numeric 19

M

MERGE Statement [6]
Mnemonic-Name 16
Mode, Access 165, 197, 229
MOVE Statement 128
Multiple Results in Arithmetic Statements 97
MULTIPLY Statement 132

N

Name Characteristics 47
Name,
  Condition 16, 30
  Mnemonic 16
  Paragraph 17
  Section 17
  System 17
  User-Defined 17
Negated Simple Conditions 91
Nonnumeric Literals 18
Nucleus Function 47
Nucleus,
  Data Division in 55
  Environment Division in 50
  Identification Division in 48
  Organization 48
  Procedure Division in 84
  Structure 47
Number,
  Level 22, 61
  Sequence 11, 43
Numeric Data Rules 62
Numeric Edited Data Rules 63
Numeric Literals 19
Numeric Operands, Comparison of 87

O

OBJECT-COMPUTER Paragraph 50
OBJECT Time DEBUG Switch [28]
OCCURS Clause 153
OPEN Statement 182, 213, 246
Operand Comparison 87
Operand, Overlapping 97, 156
Operators, Arithmetic 84
Organization,
  Data Division 37
  Environment Division 35
  Identification Division 34
  Indexed Input-Output Module 229
  LINE SEQUENTIAL 168
  Nucleus 47
  Procedure Division 38
  Relative Input-Output Module 197
  Segmentation [19]
  Sequential Input-Output Module 165
ORGANIZATION IS INDEXED 234
ORGANIZATION IS LINE SEQUENTIAL 168
ORGANIZATION IS RELATIVE 201
ORGANIZATION IS SEQUENTIAL 168
Overlapping Operands 97, 156

P

Paragraph Format 44
Paragraph-Name 17
Paragraph,
  DATE-COMPILED 49
  FILE-CONTROL 168, 201, 234, [1]
  I-O CONTROL 170, 204, 236, [1]
  OBJECT-COMPUTER 50
  PROGRAM ID 49
  SOURCE-COMPUTER 50
  SPECIAL-NAMES 51
PERFORM Statement 134, [22]
Phrase,
  CORRESPONDING 96
  ROUNDED 95
  SIZE ERROR 95
PICTURE Character Strings 21
PICTURE Clause 62
Portion, Fixed [19]
Precedence Rules 69
Procedure Division Header 38, [35]
Procedure Division in COBOL DEBUG [29]
Procedure Division in the Communication Module [52]
Procedure Divsion in Indexed I-O Module 244
Procedure Division in the Interprogram Communication Module [35]
Procedure Division in the Nucleus 84
Procedure Division in the Sequential I-O Module 181
Procedure Division in the Relative I-O Module 211
Procedure Division in the Sort-Merge Module [6]
Procedure Division 38
Procedure Division,
  Body 39
  Declaratives 38
  Execution 39
  General Format 39
Procedures 38
PROGRAM-ID Paragraph 49
Program Segments [19]
Program Structure 9, 33, [21]
Programming Techniques [65]

Q

Qualification 27

R

READ Statement 185, 216, 249
RECEIVE Statement [57]
RECORD CONTAINS Clause 179, 209, 242, [5]
Record Description Format 44
Record Description Structure 172, 206, 239
RECORD KEY Clause 234
Record Pointer, Current 165, 197, 229
REDEFINES Clause 72
Reference Format 42
Reference, Uniqueness of 27
Relation Condition 86
Relation Condition, Table Handling 156
Relative Input-Output Module,
  Data Division in 206
  Environment Division in 201
  Procedure Division in 211
RELEASE Statement [10]
RENAMES Clause 74
Reserved Words 18, 46, [69]
RETURN Statement [11]
REWRITE Statement 188, 219, 251
ROUNDED Phrase 95
Rules,
  Alignment, Standard 26
  Alphabetic Data 62
  Alphanumeric Data 63
  Alphanumeric Edited Data 63
  Condition-Name 82
  Editing 66
  Editing,
    Fixed Insertion 67
    Floating Insertion 68
    Simple Insertion 67
    Special Insertion 67
    Zero Suppression 69
  Elementary Item Size 64
  Evaluation Condition 93
  Evaluation 84
  Formulation 84
  General 10
  Numeric Data 62

Numeric Edited Data 63
Precedence 69
Symbols Used 64
Syntax 10
Run Time Debug [27]

S

Section Format 44
Section Input-Output 168, 201, 234, [1]
Section Name 17
SECTION,
COMMUNICATION [41]
CONFIGURATION 50
FILE 172, 206, 239
LINKAGE [33]
WORKING-STORAGE 55
Segmentation [19]
Segmentation Classification [20]
Segmentation Control [20]
Segmentation Organization [19]
Segments,
Independent [19]
Program [19]
SELECT Clause 168, 201, 234
Selection of Character Representation and Radix 24
SEND Statement [60]
Sentences 39
Sentences,
Compiler Directing 40
Imperative 41
Separators 14
Separators, Conditional 14
Sequence Number 11, 43
Sequential I-O Module,
Relationship with Sort-Merge [1]
Sequential Input-Output Module 165
Sequential Input-Output Module,
Environment Division in 168
Procedure Division in 181
SET Statement 162
SIGN Clause 76
Sign Condition 90
Signs, Algebraic 26
Simple Conditions 86
Simple Insertion Editing Rules 67
SIZE ERROR Phrase 95
Sizing [66]
Sort-Merge Module [1]
Sort-Merge Module,
Data Division in [4]
DATA RECORDS Clause [4]
Environment Division in [1]
FILE-CONTROL Entry [1]
FILE-CONTROL Paragraph [1]
File Description Entry Skeleton [4]
File Section [4]
I-O Control Paragraph [1]
INPUT-OUTPUT Section [1]
Procedure Division in [6]
RECORD CONTAINS Clause [5]
SORT Statement [13]
SOURCE-COMPUTER Paragraph 50
Source Format 11
Special Insertion Editing Rules 67
SPECIAL-NAMES Paragraph 51
Specifications, Explicit and Implicit 31
Standard Alignment Rules 26
START Statement 221, 255
STATEMENT,
ACCEPT 99
ACCEPT MESSAGE COUNT [52]
ADD 104
ALTER 106, [22]
CALL [36]
CANCEL [38]
CLOSE 181, 211, 244
COMPUTE 107
COPY [26]
DELETE 212, 245
DISABLE [53]
DISPLAY 108
DIVIDE 111
ENABLE [55]
ENTER 114
EXIT 115
EXIT PROGRAM [39]
GO TO 116
IF 118
INSPECT 120
MERGE [6]
MOVE 128
MULTIPLY 132
OPEN 182, 213, 246
PERFORM 134

READ 185, 216, 249
RECEIVE [57]
RELEASE [10]
RETURN [11]
REWRITE 188, 219, 251
SEND [60]
SET 162
SORT [13]
START 221, 255
STOP 145
STRING 142
SUBTRACT 146
UNSTRING 148
USE 190, 223, 257
USE FOR DEBUGGING [29]
WRITE 191, 225, 257
Statements,
Arithmetic 96
Compiler Directing 40
Conditional 40
Imperative 41
Status Keys 165, 198, 230, [51]
Status, Input-Output 165, 198, 230
STOP Statement 145
STRING Statement 142
Structure,
Data Division 37
Environment Division 35
Identification Division 34
Language 13
Nucleus 47
Procedure Division 39
Program 8, 33
Program Segments [21]
Record Description 172, 206, 239
Subscripting 28
SUBTRACT Statement 146
Suppression Editing, Zero 69
Switch, Compile Time [27]
Switch Status Condition 89
Symbols Used Rules 64
SYNCHRONIZED Clause 78
Syntax Rules 10
Syntax Rules, in Nucleus 47
System-Name 17

T

Table Handling 153
Table Handling,
Data Division in 153
Procedure Division in 156
Techniques, Programming [65]
Transfers of Control, Explicit and Implicit 31

U

Uniqueness of Reference 27
UNSTRING Statement 148
USAGE Clause 80
USE FOR DEBUGGING Statement [29]
USE Statement 190, 223, 257
Useful Hints [65]
User-Defined Names 17
User-Defined Words 15

V

VALUE Clause 81
VALUE OF Clause 179, 210, 242

W

WITH DEBUGGING MODE Clause [28]
Words,
COBOL 15
Key 18
Optional 18
Reserved 18, 46, [69]
User Defined 15
Working-Storage, Noncontiguous 55
Working-Storage Records 55
WORKING-STORAGE Section 55
WRITE Statement 191, 225, 257

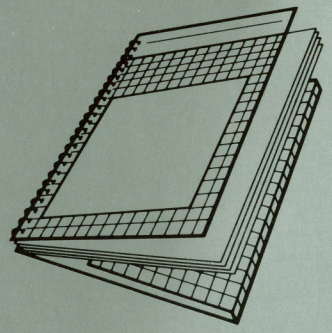
X

Y

Z

Zero-Suppression Editing Rules 69

Tuck end flap
inside back cover
when using manual.



Apple III

Apple III COBOL: Language Reference Manual Volume 1

apple computer
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576
030-0436-A

Photographs courtesy of Vorpal Gallery, San Francisco, New York, Laguna Beach, CA.