American National Standard

Adopted for Use by
the Federal Government

FIPS PUB 21-1
See Notice on Inside
Front Cover

programming language COBOL

X3.23-1974

american national standards institute, inc.
1430 broadway, new york, new york 10018

AMERICAN NATIONAL STANDARD

An American National Standard implies a consensus of those substantially concerned with its scope and provisions. An American National Standard is intended as a guide to aid the manufacturer, the consumer, and the general public. The existence of an American National Standard does not in any respect preclude anyone, whether he has approved the standard or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standard. American National Standards are subject to periodic review and users are cautioned to obtain the latest editions.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken to reaffirm, revise, or withdraw this standard no later than five years from the date of publication. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

ACKNOWLEDGMENT

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

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

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

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

This standard has been adopted for federal government use.

Details concerning its use within the federal government are contained in FIPS PUB 21-1, COMMON BUSINESS ORIENTED LANGUAGE (COBOL). For a complete list of the publications available in the FEDERAL INFORMATION PROCESSING STANDARDS Series, write to the Office of Technical Information and Publications, National Bureau of Standards, Washington, D.C. 20234.

Published by

American National Standards Institute
1430 Broadway, New York, New York 10018

Printed in the United States of America

P2M477/15

ANSI
X3.23-1974
Revision of
X3.23-1968

American National Standard Programming Language COBOL

Secretariat

Computer and Business Equipment Manufacturers Association

Approved May 10, 1974

American National Standards Institute, Inc

FOREWORD

(This Foreword is not a part of American National Standard Programming Language COBOL, X3.23-1974.)

This standard is a revision of American National Standard COBOL, X3.23-1968. The language specifications contained in this standard were drawn from both American National Standard X3.23-1968 and the CODASYL COBOL Journal of Development. Like its predecessor, this document provides specifications for both the form and interpretation of programs expressed in COBOL. It is intended to provide a high degree of machine independence in such programs in order to permit their use on a variety of automatic data processing systems.

The organization of COBOL specifications in this standard is based on a functional processing concept. The standard defines a Nucleus and eleven functional processing modules: Table Handling, Sequential I-O, Relative I-O, Indexed I-O, Sort-Merge, Report Writer, Segmentation, Library, Debug, Inter-Program Communication, and Communication. Each module contains two or three levels with nine modules having a null set as the lowest level. In all cases, lower levels are proper subsets of the higher levels within the same module. The minimum standard is defined as the low level of the Nucleus plus the low level of the Table Handling and Sequential I-O modules. Full American National Standard COBOL is defined as the highest level of the Nucleus and the eleven processing modules. The major technical differences between this standard and its predecessor are detailed in Appendix B on pages XIV-9 through XIV-34.

The Technical Committee responsible for this standard, X3J4, evolved from Committee X3.4.4 and its subordinate working groups (the bodies responsible for the original COBOL standard). X3J4 began the task of preparing a revision of the COBOL standard in 1969 with the development of criteria against which each candidate for inclusion in the proposed revision was to be matched. Detailed work on the revision began in early 1970 and, with the committee meeting every 4 to 6 weeks, a draft was completed in June 1972. COBOL Information Bulletins 14, 15, and 16, published in the first half of 1972, kept the COBOL community informed on the progress being made.

American National Standards Committee on Computers and Information Processing, X3, approved the publication of the draft in July 1972, and the full text of the proposed revision was made available to the community for comment in September 1972. It was approved as an American National Standard on May 10, 1974.

This standard was processed and approved for submittal to ANSI by American National Standards Committee on Computers and Information Processing, X3. Committee approval of the standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

J. F. Auwaerter, Chairman
V. E. Henriques, Vice-Chairman
R. M. Brown, Secretary

Organization Represented	Name of Representative
Addressograph Multigraph Corporation	A. C. Brown
	D. S. Bates (Alt)
Air Transport Association	F. C. White
American Bankers Association	M. E. McMahon
	J. Booth (Alt)

Organization Represented	Name of Representative
American Institute of Certified Public Accountants	N. Zakin
	P. B. Goodstat (Alt)
	C. A. Phillips (Alt)
	F. Schiff (Alt)
American Library Association	J. R. Rizzolo
	J. C. Kountz (Alt)
	M. S. Malinconico (Alt)
American Newspaper Publishers Association	W. D. Rinehart
American Nuclear Society	D. R. Vondy
	M. K. Butler (Alt)
American Society of Mechanical Engineers	R. W. Rau
	R. T. Woythal (Alt)
Association for Computing Machinery	P. Skelly
	J. A. N. Lee (Alt)
	L. Revens (Alt)
	H. Thiess (Alt)
Association for Educational Data Systems	C. Wilkes
Association for Systems Management	A. H. Vaughan
Association of American Railroads	R. A. Petrash
Association of Computer Programmers and Analysts	T. G. Grieb
	G. Thomas (Alt)
Association of Data Processing Service Organizations	J. B. Christiansen
Burroughs Corporation	E. Lohse
	J. F. Kalbach (Alt)
Control Data Corporation	S. F. Buckland
	C. E. Cooper (Alt)
Data Processing Management Association	A. E. Dubnow
	D. W. Sanford (Alt)
Edison Electric Institute	R. Bushner
	J. P. Markey (Alt)
Electronic Industries Association	(Representation Vacant)
	A. M. Wilson (Alt)
General Electric Company	R. R. Hench
	J. K. Snell (Alt)
General Services Administration	D. L. Shoemaker
	M. W. Burris (Alt)
GUIDE International	T. E. Wiese
	D. Stanford (Alt)
Honeywell Information Systems Inc	T. J. McNamara
	E. H. Clamons (Alt)
Institute of Electrical and Electronics Engineers, Communications Society	R. Gibbs
Institute of Electrical and Electronics Engineers, Computer Society	G. C. Schutz
	C. W. Rosenthal (Alt)
Insurance Accounting and Statistical Association	W. Bregartner
	J. R. Kerber (Alt)
International Business Machines Corporation	L. Robinson
	W. F. McClelland (Alt)
Joint Users Group	T. E. Wiese
	L. Rodgers (Alt)
Life Office Management Association	B. L. Neff
	A. J. Tufts (Alt)
Litton Industries	I. Danowitz
National Association of State Information Systems	G. H. Roehm
	C. Vorlander (Alt)

Organization Represented	Name of Representative
National Bureau of Standards	H. S. White, Jr
	J. O. Harrison (Alt)
National Cash Register Company	R. J. Mindlin
	T. W. Kern (Alt)
National Machine Tool Builders' Association	O. A. Rodriques
	E. J. Loeffler (Alt)
National Retail Merchants Association	I. Solomon
Olivetti Corporation of America	E. J. Almquist
Pitney-Bowes Inc	D. J. Reyen
	B. Lyman (Alt)
Printing Industries of America	N. Scharpf
	E. Masten (Alt)
Scientific Apparatus Makers Association	A. Savitsky
	J. French (Alt)
SHARE Inc	T. B. Steel, Jr
	R. H. Wahlen (Alt)
Society of Certified Data Processors	A. Taylor
	J. J. Martin (Alt)
Telephone Group	V. N. Vaughan, Jr
	S. M. Garland (Alt)
	J. C. Nelson (Alt)
UNIVAC, Division of Sperry Rand Corporation	M. W. Bass
	C. D. Card (Alt)
U.S. Department of Defense	W. L. McGreer
	W. B. Rinehuls (Alt)
	W. B. Robertson (Alt)
Xerox Corporation	J. L. Wheeler

Technical Committee X3J4, which developed this standard, had the following personnel:

R. Kearney, Chairman — J. Couperus, Vice-Chairman — P. A. Beard, Secretary

G. Abrams	R. M. Bland	D. N. Gumina	R. E. Rountree, Jr
D. G. Ashland	J. Collica	C. R. Kelleher	S. D. Schiffman
G. N. Baird	M. D. Dent	C. L. Kent	R. Solt
R. M. Barton	J. P. Desmond	J. N. Kirkeng	L. J. Soma
R. P. Belmont	N. O. Eaddy	A. M. Nienhaus	J. J. Strain
W. E. Bender	M. Fedora	P. Olshansky	L. Sturges
R. F. Betscha	R. C. Fredette	W. C. Rinehuls	D. L. Tucker
J. E. Bishop	P. R. Gustafson	S. Root	M. Vickers

Others who contributed to the work on the revision were as follows:

H. Bromberg	H. S. Gile	A. N. McMahan	C. A. Schulz
C. K. Cheng	G. H. Goe	R. M. Opsata	J. G. Solomon
J. S. Cousins	J. S. Grant	R. S. Pettus	D. F. Wendell
R. L. Dover	H. Hicks	M. L. Rakestraw	C. E. Wilder
H. R. Fletcher	J. Holloway	R. R. Risley	

The members of Technical Committee X3J4 wish to note the special contribution of the secretary, Miss P. A. Beard, whose devotion and hard work made this revision possible.

TABLE OF CONTENTS

SECTION I: INTRODUCTORY INFORMATION

SECTION II: NUCLEUS

SECTION III: TABLE HANDLING MODULE

SECTION IV: SEQUENTIAL I-O MODULE

SECTION VIII: REPORT WRITER MODULE

SECTION IX: SEGMENTATION MODULE

1. INTRODUCTION TO THE STANDARD

1.1 SCOPE AND PURPOSE

The scope of this standard is to specify both the form and interpretation of programs expressed in COBOL. Its purpose is to promote a high degree of machine independence in such programs in order to permit their use on a variety of automatic data processing systems.

1.2 STRUCTURE OF LANGUAGE SPECIFICATIONS

The organization of COBOL specifications in this standard is based on a functional processing module concept. The standard defines a Nucleus and eleven functional processing modules: Table Handling, Sequential I-O, Relative I-O, Indexed I-O, Sort-Merge, Report Writer, Segmentation, Library, Debug, Inter-Program Communication, and Communication. Each module contains either two or three levels. In all cases, the lower levels are proper subsets of the higher levels within the same module. Nine modules contain a null set as their lowest level.

This organization provides the flexibility necessary to tailor specifications in such a way that they will satisfy the requirements of a large variety of data processing applications. At the same time, inherent in this organization is the ability to determine, with a greater degree of certainty than previously possible, the elements of the standard that are included in a given compiler.

The following is a characterization of the contents of the component levels of each module.

The Nucleus contains lanugage elements that are necessary for internal processing. This module is divided into two levels. The low level supplies elements necessary to perform basic internal operations, i.e., the more elementary options of the various clauses and verbs. The high level of the Nucleus provides more extensive and sophisticated internal processing capabilities.

The Table Handling module contains the language elements necessary for: (1) the definition of tables, (2) the identification, manipulation and use of indices, and (3) reference to the items within tables. This module is divided into two levels. The low level provides the ability to define fixed length tables of up to three dimensions, and to refer to items within them using either a subscript or an index. The high level provides for the definition of variable length tables. In addition, facilities for serial and nonserial lookup are provided by the SEARCH verb and its attendant Data Division clauses.

The Sequential I-O module contains the language elements necessary for the definition and access of sequentially organized external files. The module is divided into two levels. The low level contains the basic facilities for the definition and access of sequential files and for the specification of checkpoints. The high level contains more complete facilities for defining and accessing these files.

The Relative I-O module provides the capability of defining and accessing mass storage files in which records are identified by relative record numbers.

This module contains a null set as its lowest level, and two processing levels. The low processing level provides basic facilities. The high level provides more complete facilities, including the capability of accessing the file both randomly and sequentially in the same COBOL program.

The Indexed I-O module provides the capability of defining mass storage files in which records are identified by the value of a key and accessed through an index. This module contains a null set as its lowest level, and two processing levels. The low processing level provides basic facilities. The high level provides more complete facilities, including alternate keys, and the capability of accessing the file both randomly and sequentially in the same COBOL program.

The Sort-Merge module allows for the inclusion of one or more sorts in a COBOL program, and consists of a null set and two processing levels. The low processing level contains facilities sufficient to implement basic sorting, while the high level provides extended sorting capabilities, including a merge facility.

The Report Writer module provides for the semi-automatic production of printed reports. This module consists of a null set and one processing level.

The Segmentation module provides for the overlaying at object time of Procedure Division sections. This module consists of a null set and two processing levels. The low processing level provides for section segment-numbers and fixed segment limits. The high level adds the capability for varying the segment limit.

The Library module consists of a null set and two processing levels. It provides for the inclusion into a program of predefined COBOL text. The low processing level contains the basic COPY verb, to which the high level adds the REPLACING phrase.

The Debug module provides a means by which the user can specify his debugging algorithm -- the conditions under which data or procedure items are monitored during execution of the program. It consists of a null set and two processing levels. The low processing level provides a basic debugging capability, including the ability to specify selective or full paragraph monitoring. The high level provides the full COBOL debugging capability.

The Inter-Program Communication module provides a facility by which a program can communicate with one or more other programs. This module consists of a null set and two processing levels. The low processing level provides a capability to transfer control to another program known at compile time, and the ability for both programs to have access to certain common data items. The high level adds the ability to transfer control to a program not identified at compile time as well as the ability to determine the availability of object time memory for the called program. The high level also provides the capability for the release of memory areas occupied by called programs.

The Communication module provides the ability to access, process and create messages or portions thereof, and to communicate through a Message Control System with local and remote communication devices. This module consists of a null set and two processing levels. The low processing level provides

basic facilities to send or receive complete messages. The high level provides a more sophisticated facility including the capability to send or receive segments of a message.

1.3 ORGANIZATION OF DOCUMENT

This document is divided into fifteen sections. The first section is composed of the introduction, a list of elements by module, a list of elements showing their disposition among the various modules, definitions, a discussion of overall language considerations, and a composite language skeleton. Sections II through XIII contain specifications for the Nucleus and for each of the functional processing modules. These sections comprise the detailed specifications of American National Standard COBOL. Section XIV contains the appendices to the document and Section XV contains the index.

The previous version of this standard contained a chapter for each level of the Nucleus and of the functional processing modules. This revision, in order to show more clearly the relationship of levels within a module, contains one section for each module. In each section, specifications unique to the high level are enclosed in boxes.

1.4 HOW TO USE THE STANDARD

It is envisioned that the standard will be examined from several different viewpoints. In addition to the table of contents and the index, the list of elements by module and the list of elements showing disposition are also intended to serve as a key to the standard. To determine the contents of any level, the list of elements by module beginning on page I-10 should be used. This list contains a detailed breakdown of each element of American National Standard COBOL and is organized by level. In addition, page and paragraph numbers indicate where within the standard the specification for each element is to be found. For example, to ascertain the contents of the low level of Sequential I-O, reference is made to that module within the list of elements by module (see page I-21). There will be found a list of all COBOL elements including overall language considerations, Environment Division and Data Division entries and Procedure Division verbs that pertain to Sequential I-O. Because levels are nested, in order to determine the contents of the highest level, the entire module must be examined. To obtain more detailed information concerning a specific element, the page and paragraph numbers that accompany each element in the list may be used as a key to the technical specification section of the standard.

To determine in which level or levels a specific language feature appears, the list of elements showing disposition is used. (See pages I-40 through I-51.) This list shows in detail all elements of American National Standard COBOL and their occurrences within the various levels. In addition, for each appearance of an element, the appropriate page numbers are shown. Those elements which are not completely contained within one level are shown in sufficient detail to specify the location of each subelement. If more detailed information is desired concerning the use of a specific element in any level, the page numbers adjacent to each element in the list may be used as a guide to the technical specification section of the standard. For example, to locate where the READ statement appears within the standard, the list of elements showing disposition is used. It will be seen that the READ statement appears in low level of Sequential I-O, Relative I-O, and Indexed I-O. Because certain

phrases of the READ statement appear only in the high levels of these modules, its subelements are listed separately. A page number appears for each appearance of a subelement.

When the list of elements by module is used to determine the contents of a level and subsequently it is desired to ascertain where else in the standard a particular element is used, reference would be made to the list of elements showing disposition, and from there, to the detailed technical specifications, if necessary.

For general information regarding overall language considerations or concepts, the table of contents or index may be used as a key to the standard.

Finally, to determine the content of an implementation of American National Standard COBOL, the schematic diagram on page I-5 should be used. The schematic diagram is a graphic representation of the division of COBOL into the various functional processing modules and the Nucleus. Further, the schematic shows the hierarchy of levels within each functional processing module and within the Nucleus.

1.5 DEFINITION OF AN IMPLEMENTATION OF AMERICAN NATIONAL STANDARD COBOL

In terms of the schematic diagram on page I-5, an implementation of American National Standard COBOL can be represented by a combination of boxes, consisting of one box from each of the twelve vertical columns. As illustrations, and for convenience of discourse, the following definitions are provided:

(1) The full American National Standard COBOL is composed of the highest level of the Nucleus and of each of the functional processing modules.

(2) A subset of American National Standard COBOL is any combination of levels of the Nucleus and of each of the functional processing modules other than the full American National Standard COBOL.

(3) The minimum American National Standard COBOL is composed of the lowest level of the Nucleus and of each of the functional processing modules. (Because of the presence of null sets, the minimum standard consists of the low levels of the Nucleus, Table Handling and Sequential I-O.)

An implementation is defined to meet the requirements of the American National Standard COBOL specification if that implementation includes a fully implemented specified level of each of the functional processing modules and of the nucleus as defined in this standard. It follows from this that, in order to meet the requirements of this standard, an implementation must:

(1) Not require the inclusion of substitute or additional language elements in the source program, in order to accomplish any part of the function of any of the standard language elements.

(2) Accept all standard language elements contained in a given level of a module which is specified as being included in the implementation, except as specifically exempted by paragraph 1.7 on page I-8.

NUCLEUS	FUNCTIONAL PROCESSING MODULES										
	TABLE HANDLING	SEQUENTIAL I-O	RELATIVE I-O	INDEXED I-O	SORT-MERGE	REPORT WRITER	SEGMEN-TATION	LIBRARY	DEBUG	INTER-PROGRAM COMMUNICATION	COMMUNI-CATION
2 NUC 1,2	2 TBL 1,2	2 SEQ 1,2	2 REL 0,2	2 INX 0,2	2 SRT 0,2	1 RPW 0,1	2 SEG 0,2	2 LIB 0,2	2 DEB 0,2	2 IPC 0,2	2 COM 0,2
			1 REL 0,2	1 INX 0,2	1 SRT 0,2		1 SEG 0,2	1 LIB 0,2	1 DEB 0,2	1 IPC 0,2	1 COM 0,2
1 NUC 1,2	1 TBL 1,2	1 SEQ 1,2	null	null	null	null	null	null	null	null	null

These points are of particular pertinence in two areas:

(1) There are throughout the American National Standard COBOL specification certain language elements whose syntax or effect is specified to be, in part, implementor-defined. (See paragraph 1.6 on page I-7 for a list of these elements.) While the implementor specifies the constraints on that portion of each element's syntax or rules that is indicated in this standard to be implementor-defined, such constraints may not include any requirement for the inclusion in the source program of substitute or additional language elements.

(2) When a function is provided outside the source program that accomplishes a function specified by any particular standard COBOL element, then the implementation must not require, except for Environment Division elements, the specification of that external function in place of or in addition to that standard language element.

The following qualifications apply to the American National Standard COBOL specification:

(1) There are certain language elements which pertain to specific types of hardware components (see paragraph 1.7 on page I-8 for a list of these elements). In order for an implementation to meet the requirements of this standard, the implementor must specify the minimum hardware configuration required for that implementation and the hardware components that it supports. Further, when support is thus claimed for a specific hardware component, all standard language elements that pertain to that component must be implemented if the module in which they appear is included in the implementation. Language elements that pertain to specific hardware components for which support is not claimed, need not be implemented. However, the absence of such elements from an implementation of American National Standard COBOL must be specified.

(2) An implementation of American National Standard COBOL may include the ENTER statement or not, at the option of the implementor.

(3) An implementation that includes, in addition to a specified level of each of the functional processing modules and of the Nucleus, elements or functions that either are not defined in the American National Standard COBOL specification or are defined in a given level of a standard module not otherwise included in the implementation, meets the requirements of this standard. This is true even though it may imply the extension of the list of reserved words by the implementor, and prevent proper compilation of some programs that meet the requirements of this standard. The implementor must specify any optional language (language not defined in a specified level but defined elsewhere in the standard) or extensions (language elements or functions not defined in this standard) that are included in the implementation.

(4) In general, the American National Standard COBOL specification specifies no upper limit on such things as the number of statements in a program, the number of operands permitted in certain statements, etc. It is recognized that these limits will vary from one implementation of American National Standard COBOL to another and may prevent the proper compilation of some programs that meet the requirements of this standard.

(5) For a discussion of character substitution which likewise may prevent the proper compilation of some programs that meet the requirements of this standard, see page I-75, paragraph 5.3.1, Character Set.

1.6 IMPLEMENTOR-DEFINED LANGUAGE SPECIFICATIONS

The language elements in the following lists depend on implementor definitions to complete the specification of the syntax or rules for the elements.

The elements whose syntax is partly implementor-defined are:

Element	Implementor-Defined Aspect
SOURCE-COMPUTER paragraph	computer-name
OBJECT-COMPUTER paragraph	computer-name
MEMORY SIZE clause	integer
alphabet-name	implementor-name; whether implementor-names are provided.
SPECIAL-NAMES paragraph	implementor-name
ASSIGN clause	implementor-name
VALUE OF clause	implementor-name; whether implementor-names are provided.
RERUN clause	implementor-name and the form; the implementor provides at least one of seven specified forms.
CALL and CANCEL statements	relationship between operand and the referenced program.
COPY statement	relationship between library-name, text-name, and the library.
ENTER statement	language-name
Margin R	The location.
Area B	The number of character positions.
Qualification	The number of qualifiers; at least five levels must be supported.

The elements whose effect is partly implementor-defined are:

Element	Implementor-Defined Aspect
alphabet-name	The correspondence between native and foreign character sets.
implementor-name switches	Whether setting can change during execution.
USAGE IS COMPUTATIONAL clause	Representation and whether automatic alignment occurs.

Element	Implementor-Defined Aspect
USAGE IS INDEX clause	Representation and whether automatic alignment occurs.
SYNCHRONIZED clause	Whether implicit FILLER positions are generated; their effect on the size of group items and redefining items.
ACCEPT statement	Maximum size of one transfer of data in Level 1 Nucleus.
DISPLAY statement	Maximum size of one transfer of data in Level 1 Nucleus.
Numeric test	Representation of valid sign in the absence of the SIGN IS SEPARATE clause.
Comparison of nonnumeric items	Collating sequence, where NATIVE or implementor-name collating sequence is implicitly or explicitly specified.
Arithmetic expressions	Number of places carried for intermediate results.

1.7 ELEMENTS THAT PERTAIN TO SPECIFIC HARDWARE COMPONENTS

The standard language elements in the list that follows pertain to specific types of hardware components. These language elements must be implemented in an implementation of American National Standard COBOL when support is claimed, by the implementor, for the specific types of hardware components to which they pertain, and the module in which they are defined is included in that implementation.

Element	Hardware Component
CODE-SET clause	Device capable of supporting the specified code.
MULTIPLE FILE TAPE clause	Reel
CLOSE...REEL/UNIT statement	Reel or mass storage
CLOSE...NO REWIND statement	Reel or mass storage
OPEN...REVERSED statement	Reel with the capability of making records available in the reversed order; mass storage with the capability of making records available in the reversed order.
OPEN...NO REWIND statement	Reel or mass storage
OPEN...I-O statement (Sequential I-O only)	Mass storage
OPEN EXTEND statement	Reel or mass storage
REWRITE statement (Sequential I-O only)	Mass storage
SEND...BEFORE/AFTER ADVANCING statement	Devices capable of vertical positioning; devices capable of action based on mnemonic-names.

Element	Hardware Component
USE...I-O (Sequential I-O only)	Mass storage
WRITE...BEFORE/AFTER ADVANCING	Devices capable of vertical positioning; devices capable of action based on mnemonic-name.

1.8 SHORTHAND NOTATION

Within the schematic diagram on page I-5, the list of elements by module on pages I-10 through I-39, and the list of elements showing disposition on pages I-40 through I-51, a shorthand notation has been adopted to indicate the hierarchical position of any level within the Nucleus or a functional processing module as well as the number of levels into which a module has been divided. This code is composed of, from left to right, a one-digit number indicating the level's position in the hierarchy, a three-character mnemonic name, and a two-digit number indicating the minimum and maximum levels of the module to which the level belongs. A level number of zero indicates a null level. For example, 2 NUC 1,2 indicates that this level is the second level of the Nucleus and that the Nucleus is composed of two levels, neither one of which is a null set. As a further example, 2 SRT 0,2 indicates that this level is the second non-null level of the Sort-Merge module which contains three levels, the lowest of which is a null level.

The mnemonic names that are used in these codes are the following:

Mnemonic Name	Meaning
NUC	Nucleus
TBL	Table Handling
SEQ	Sequential I-O
REL	Relative I-O
INX	Indexed I-O
SRT	Sort-Merge
RPW	Report Writer
SEG	Segmentation
LIB	Library
DEB	Debug
IPC	Inter-Program Communication
COM	Communication

2. LIST OF ELEMENTS BY MODULE

2.1 GENERAL DESCRIPTION

This chapter contains a list of all elements in the American National Standard COBOL organized by the level in which each element is located. Adjacent to each element is a text reference. This reference indicates the page number and the paragraph number of the detailed specification describing the particular element.

NUCLEUS, LEVEL 1 (1 NUC 1,2)

NUCLEUS, LEVEL 1 (1 NUC 1,2)

NUCLEUS, LEVEL 1 (1 NUC 1,2)

NUCLEUS, LEVEL 1 (1 NUC 1,2)

NUCLEUS, LEVEL 1 (1 NUC 1,2)

NUCLEUS, LEVEL 2 (2 NUC 1,2)

NUCLEUS, LEVEL 2 (2 NUC 1,2)

NUCLEUS, LEVEL 2 (2 NUC 1,2)

TABLE HANDLING, LEVEL 1 (1 TBL 1,2)

TABLE HANDLING, LEVEL 2 (2 TBL 1,2)

SEQUENTIAL I-O, LEVEL 1 (1 SEQ 1,2)

SEQUENTIAL I-O, LEVEL 1 (1 SEQ 1,2)

SEQUENTIAL I-O, LEVEL 2 (2 SEQ 1,2)

RELATIVE I-O, LEVEL 1 (1 REL 0,2)

RELATIVE I-O, LEVEL 1 (1 REL 0,2)

RELATIVE I-O, LEVEL 2 (2 REL 0,2)

INDEXED I-O, LEVEL 1 (1 INX 0,2)

INDEXED I-O, LEVEL 1 (1 INX 0,2)

INDEXED I-O, LEVEL 2 (2 INX 0,2)

SORT-MERGE, LEVEL 1 (1 SRT 0,2)

SORT-MERGE, LEVEL 2 (2 SRT 0,2)

REPORT WRITER, LEVEL 1 (1 RPW 0,1)

REPORT WRITER, LEVEL 1 (1 RPW 0,1)

SEGMENTATION, LEVEL 1 (1 SEG 0,2)

SEGMENTATION, LEVEL 2 (2 SEG 0,2)

LIBRARY, LEVEL 1 (1 LIB 0,2)

LIBRARY, LEVEL 2 (2 LIB 0,2)

DEBUG, LEVEL 1 (DEB 0,2)

DEBUG, LEVEL 2 (2 DEB 0,2)

INTER-PROGRAM COMMUNICATION 1 (1 IPC 0,2)

INTER-PROGRAM COMMUNICATION 2 (2 IPC 0,2)

COMMUNICATION (1 COM 0,2)

COMMUNICATION (2 COM 0,2)

3. LIST OF ELEMENTS SHOWING DISPOSITION

3.1 GENERAL DESCRIPTION

This chapter contains a list of all elements in American National Standard COBOL showing the levels in which each element is introduced. Adjacent to each level code is a text reference. This reference indicates the page number of the detailed specification describing the particular element.

4. GLOSSARY

4.1 INTRODUCTION

The terms in this chapter are defined in accordance with their meaning as used in this document describing COBOL and may not have the same meaning for other languages.

These definitions are also intended to be either reference material or introductory material to be reviewed prior to reading the detailed language specifications that follow. For this reason, these definitions are, in most instances, brief and do not include detailed syntactical rules.

4.2 DEFINITIONS

Abbreviated Combined Relation Condition. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode. The manner in which records are to be operated upon within a file.

Actual Decimal Point. The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Alphabet-Name. A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

Alphabetic Character. A character that belongs to the following set of letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space.

Alphanumeric Character. Any character in the computer's character set.

Alternate Record Key. A key, other than the prime record key, whose contents identify a record within an indexed file.

Arithmetic Expression. An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator. A single character, or a fixed two-character combination, that belongs to the following set:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Ascending Key. A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point. A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition. A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.

2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

Block. A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Body Group. Generic name for a report group of TYPE DETAIL, CONTROL HEADING or CONTROL FOOTING.

Called Program. A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

Calling Program. A program which executes a CALL to another program.

Cd-Name. A user-defined word that names an MCS interface area described in a communication description entry within the Communication Section of the Data Division.

Character. The basic indivisible unit of the language.

Character Position. A character position is the amount of physical storage required to store a single standard data format character described as usage is DISPLAY. Further characteristics of the physical storage are defined by the implementor.

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

Class Condition. The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Clause. A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Character Set. The complete COBOL character set consists of the 51 characters listed below:

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

COBOL Word. (See Word)

Collating Sequence. The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging, and comparing.

Column. A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

Combined Condition. A condition that is the result of connecting two or more conditions with the 'AND' or the 'OR' logical operator.

Comment-Entry. An entry in the Identification Division that may be any combination of characters from the computer character set.

Comment Line. A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

Communication Description Entry. An entry in the Communication Section of the Data Division that is composed of the level indicator CD, followed by a cd-name, and then followed by a set of clauses as required. It describes the interface between the Message Control System (MCS) and the COBOL program.

Communication Device. A mechanism (hardware or hardware/software) capable of sending data to a queue and/or receiving data from a queue. This mechanism may be a computer or a peripheral device. One or more programs containing communication description entries and residing within the same computer define one or more of these mechanisms.

Communication Section. The section of the Data Division that describes the interface areas between the MCS and the program, composed of one or more CD description entries.

Compile Time. The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Compiler Directing Statement. A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

Complex Condition. A condition in which one or more logical operators act upon one or more conditions. (See Negated Simple Condition, Combined Condition, Negated Combined Condition.)

Computer-Name. A system-name that identifies the computer upon which the program is to be compiled or run.

Condition. A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, ...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-Name. A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of an implementor-defined switch or device.

Condition-Name Condition. The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression. A simple condition or a complex condition specified in an IF, PERFORM, or SEARCH statement. (See Simple Condition and Complex Condition.)

Conditional Statement. A 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.

Conditional Variable. A data item one or more values of which has a condition-name assigned to it.

Configuration Section. A section of the Environment Division that describes overall specifications of source and object computers.

Connective. A reserved word that is used to:

1. Associate a data-name, paragraph-name, condition-name, or text-name with its qualifier.
2. Link two or more operands written in a series.
3. Form conditions (logical connectives). (See Logical Operator)

Contiguous Items. Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

Control Break. A change in the value of a data item that is referenced in the CONTROL clause. More generally, a change in the value of a data item that is used to control the hierarchical structure of a report.

Control Break Level. The relative position within a control hierarchy at which the most major control break occurred.

Control Data Item. A data item, a change in whose contents may produce a control break.

Control Data-Name. A data-name that appears in a CONTROL clause and refers to a control data item.

Control Footing. A report group that is presented at the end of the control group of which it is a member.

Control Group. A set of body groups that is presented for a given value of a control data item or of FINAL. Each control group may begin with a CONTROL HEADING, end with a CONTROL FOOTING, and contain DETAIL report groups.

Control Heading. A report group that is presented at the beginning of the control group of which it is a member.

Control Hierarchy. A designated sequence of report subdivisions defined by the positional order of FINAL and the data-names within a CONTROL clause.

Counter. A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Currency Sign. The character '$' of the COBOL character set.

Currency Symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

Current Record. The record which is available in the record area associated with the file.

Current Record Pointer. A conceptual entity that is used in the selection of the next record.

Data Clause. A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Data Description Entry. An entry in the Data Division that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Item. A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL program.

Data-Name. A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats, 'data-name' represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

Debugging Line. A debugging line is any line with 'D' in the indicator area of the line.

Debugging Section. A debugging section is a section that contains a USE FOR DEBUGGING statement.

Declaratives. A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative-Sentence. A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

Delimiter. A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key. A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Destination. The symbolic identification of the receiver of a transmission from a queue.

Digit Position. A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage of the data item describing the digit position. Further characteristics of the physical storage are defined by the implementor.

Division. A set of zero, one or more sections of paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

Division Header. A combination of words followed by a period and a space that indicates that beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION  [USING data-name-1 [data-name-2] ... ] .
```

Dynamic Access. An access mode in which specific logical records can be obtained from or placed into a mass storage file in a non-sequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access), during the scope of the same OPEN statement.

Editing Character. A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	stroke (virgule, slash)

Elementary Item. A data item that is described as not being further logically subdivided.

End of Procedure Division. The physical position in a COBOL source program after which no further procedures appear.

Entry. Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

Environment Clause. A clause that appears as part of an Environment Division entry.

Execution Time. (See Object Time)

Extend Mode. The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Figurative Constant. A compiler generated value referenced through the use of certain reserved words.

File. A collection of records.

File Clause. A clause that appears as part of any of the following Data Division entries:

File description (FD)
Sort-merge file description (SD)
Communication description (CD)

FILE-CONTROL. The name of an Environment Division paragraph in which the data files for a given source program are declared.

File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name. A user-defined word that names a file described in a file description entry or a sort-merge file description entry within the File Section of the Data Division.

File Organization. The permanent logical file structure established at the time that a file is created.

File Section. The section of the Data Division that contains file description entries and sort-merge file description entries together with their associated record descriptions.

Format. A specific arrangement of a set of data.

Group Item. A named contiguous set of elementary or group items.

High Order End. The leftmost character of a string of characters.

I-O-CONTROL. The name of an Environment Division paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

I-O Mode. The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

Identifier. A data-name, followed as required, by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

Imperative Statement. A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

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

Index. A computer storage position or register, the contents of which represent the identification of a particular element in a table.

Index Data Item. A data item in which the value associated with an index-name can be stored in a form specified by the implementor.

Index-Name. A user-defined word that names an index associated with a specific table.

Indexed Data-Name. An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed File. A file with indexed organization.

Indexed Organization. The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

Input File. A file that is opened in the input mode.

Input Mode. The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-Output File. A file that is opened in the I-O mode.

Input-Output Section. The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Input Procedure. A set of statements that is executed each time a record is released to the sort file.

Integer. A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Invalid Key Condition. A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

Key. A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key of Reference. The key, either prime or alternate, currently being used to access records within an indexed file.

Key Word. A reserved word whose presence is required when the format in which the word appears is used in a source program.

Language-Name. A system-name that specifies a particular programming language.

Level Indicator. Two alphabetic characters that identify a specific type of file or a position in hierarchy.

Level-Number. A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one or two digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

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

Library Text. A sequence of character-strings and/or separators in a COBOL library.

Line. (See Report Line)

Line Number. An integer that denotes the vertical position of a report line on a page.

Linkage Section. The section in the Data Division of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal. A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator. One of the reserved words AND, OR, or NOT. In the formation of a condition, both or either of AND and OR can be used as logical connectives. NOT can be used for logical negation.

Logical Record. The most inclusive data item. The level-number for a record is 01. (See Report Writer Logical Record)

Low Order End. The rightmost character of a string of characters.

Mass Storage. A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage Control System (MSCS). An input-output control system that directs, or controls, the processing of mass storage files.

Mass Storage File. A collection of records that is assigned to a mass storage medium.

MCS. (See Message Control System)

Merge File. A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

Message. Data associated with an end of message indicator or an end of group indicator. (See Message Indicators)

Message Control System (MCS). A communication control system that supports the processing of messages.

Message Count. The count of the number of complete messages that exist in the designated queue of messages.

Message Indicators. EGI (end of group indicator), EMI (end of message indicator), and ESI (end of segment indicator) are conceptual indications that serve to notify the MCS that a specific condition exists (end of group, end of message, end of segment).

Within the hierarchy of EGI, EMI, and ESI, an EGI is conceptually equivalent to an ESI, EMI, and EGI. An EMI is conceptually equivalent to an ESI and EMI. Thus, a segment may be terminated by an ESI, EMI, or EGI. A message may be terminated by an EMI or EGI.

Message Segment. Data that forms a logical subdivision of a message normally associated with an end of segment indicator. (See Message Indicators)

Mnemonic-Name. A user-defined word that is associated in the Environment Division with a specified implementor-name.

MSCS. (See Mass Storage Control System)

Native Character Set. The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

Native Collating Sequence. The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

Negated Combined Condition. The 'NOT' logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition. The 'NOT' logical operator immediately followed by a simple condition.

Next Executable Sentence. The next sentence to which control will be transferred after execution of the current statement is complete.

Next Executable Statement. The next statement to which control will be transferred after execution of the current statement is complete.

Next Record. The record which logically follows the current record of a file.

Noncontiguous Items. Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.

Nonnumeric Item. A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal. A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character. A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item. A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

Numeric Literal. A literal composed of one or more numeric characters that also may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

Object of Entry. A set of operands and reserved words, within a Data Division entry, that immediately follows the subject of the entry.

Object Program. A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program'.

Object Time. The time at which an object program is executed.

Open Mode. The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

Operand. Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this publication, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign. An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional Word. A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File. A file that is opened in either the output mode or extend mode.

Output Mode. The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified for that file and before the execution of a CLOSE statement for that file.

Output Procedure. A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function has selected the next record in merged order.

Page. A vertical division of a report representing a physical separation of report data, the separation being based on internal reporting requirements and/or external characteristics of the reporting medium.

Page Body. That part of the logical page in which lines can be written and/or spaced.

Page Footing. A report group that is presented at the end of a report page as determined by the Report Writer Control System.

Page Heading. A report group that is presented at the beginning of a report page and determined by the Report Writer Control System.

Paragraph. In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment Divisions, a paragraph header followed by zero, one, or more entries.

Paragraph Header. A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the Identification and Environment Divisions. The permissible paragraph headers are:

In the Identification Division:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

In the Environment Division:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name. A user-defined word that identifies and begins a paragraph in the Procedure Division.

Phrase. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record. (See Block)

Prime Record Key. A key whose contents uniquely identify a record within an indexed file.

Printable Group. A report group that contains at least one print line.

Printable Item. A data item, the extent and contents of which are specified by an elementary report entry. This elementary report entry contains a COLUMN NUMBER clause, a PICTURE clause, and a SOURCE, SUM or VALUE clause.

Procedure. A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-Name. A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified), or a section-name.

Program-Name. A user-defined word that identifies a COBOL source program.

Pseudo-Text. A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters.

Pseudo-Text Delimiter. Two contiguous equal sign (=) characters used to delimit pseudo-text.

Punctuation Character. A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
.	period
"	quotation mark
(	left parenthesis
)	right parenthesis
	space
=	equal sign

Qualified Data-Name. An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier.

1. A data-name which is used in a reference together with another data name at a lower level in the same hierarchy.

2. A section-name which is used in a reference together with a paragraph-name specified in that section.

3. A library-name which is used in a reference together with a text-name associated with that library.

Queue. A logical collection of messages awaiting transmission or processing.

Queue Name. A symbolic name that indicates to the MCS the logical path by which a message or a portion of a completed message may be accessible in a queue.

Random Access. An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

Record. (See Logical Record)

Record Area. A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description. (See Record Description Entry)

Record Description Entry. The total set of data description entries associated with a particular record.

Record Key. A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record-Name. A user-defined word that names a record described in a record description entry in the Data Division.

Reference Format. A format that provides a standard method for describing COBOL source programs.

Relation. (See Relational Operator)

Relation Character. A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition. The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item. (See Relational Operator)

Relational Operator. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meaning are:

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

Relative File. A file with relative organization.

Relative Key. A key whose contents identify a logical record in a relative file.

Relative Organization. The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Report Clause. A clause, in the Report Section of the Data Division, that appears in a report description entry or a report group description entry.

Report Description Entry. An entry in the Report Section of the Data Division that is composed of the level indicator RD, followed by a report name, followed by a set of report clauses as required.

Report File. An output file whose file description entry contains a REPORT clause. The contents of a report file consist of records that are written under control of the Report Writer Control System.

Report Footing. A report group that is presented only at the end of a report.

Report Group. In the Report Section of the Data Division, an 01 level-number entry and its subordinate entries.

Report Group Description Entry. An entry in the Report Section of the Data Division that is composed of the level-number 01, the optional data-name, a TYPE clause, and an optional set of report clauses.

Report Heading. A report group that is presented only at the beginning of a report.

Report Line. A division of a page representing one row of horizontal character positions. Each character position of a report line is aligned vertically beneath the corresponding character position of the report line above it. Report lines are numbered from 1, by 1, starting at the top of the page.

Report-Name. A user-defined word that names a report described in a report description entry within the Report Section of the Data Division.

Report Section. The section of the Data Division that contains one or more report description entries and their associated report group description entries.

Report Writer Control System (RWCS). An object time control system, provided by the implementor, that accomplishes the construction of reports.

Report Writer Logical Record. A record that consists of the Report Writer print line and associated control information necessary for its selection and vertical positioning.

Reserved Word. A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Routine-Name. A user-defined word that identifies a procedure written in a language other than COBOL.

Run Unit. A set of one or more object programs which function, at object time, as a unit to provide problem solutions.

RWCS. (See Report Writer Control System)

Section. A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

Section Header. A combination of words followed by a period and a space that indicates the beginning of a section in the Environment, Data and Procedure Division.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

In the Data Division:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
COMMUNICATION SECTION.
REPORT SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-Name. A user-defined word which names a section in the Procedure Division.

Segment-Number. A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0', '1', ..., '9'. A segment-number may be expressed either as a one or two digit number.

Sentence. A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator. A punctuation character used to delimit character-strings.

Sequential Access. An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

Sequential File. A file with sequential organization.

Sequential Organization. The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Sign Condition. The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Simple Condition. Any single condition chosen from the set:

relation condition
class condition
condition-name condition
switch-status condition
sign condition
(simple-condition)

Sort File. A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

Sort-Merge File Description Entry. An entry in the File Section of the Data Division that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

Source. The symbolic identification of the originator of a transmission to a queue.

SOURCE-COMPUTER. The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Item. An identifier designated by a SOURCE clause that provides the value of a printable item.

Source Program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'source program'.

Special Character. A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

Special-Character Word. A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an Environment Division paragraph in which implementor-names are related to user specified mnemonic-names.

Special Registers. Compiler generated storage areas whose primary use is to store information produced in conjunction with the user of specific COBOL features.

Standard Data Format. The concept used in describing the characteristics of data in a COBOL Data Division under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

Statement. A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Sub-Queue. A logical hierarchical division of a queue.

Subject of Entry. An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

Subprogram. (See Called Program)

Subscript. An integer whose value identifies a particular element in a table.

Subscripted Data-Name. An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Sum Counter. A signed numeric data item established by a SUM clause in the Report Section of the Data Division. The sum counter is used by the Report Writer Control System to contain the result of designated summing operations that take place during production of a report.

Switch-Status Condition. The proposition, for which a truth value can be determined, that an implementor-defined switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

System-Name. A COBOL word which is used to communicate with the operating environment.

Table. A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element. A data item that belongs to the set of repeated items comprising a table.

Terminal. The originator of a transmission to a queue, or the receiver of a transmission from a queue.

Text-Name. A user-defined word which identifies library text.

Text-Word. Any character-string or separator, except space, in a COBOL library or in pseudo-text.

Truth Value. The representation of the result of the evaluation of a condition in terms of one of two values

true
false

Unary Operator. A plus (+) or a minus (-) sign, which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression of +1 or -1 respectively.

Unit. A module of mass storage the dimensions of which are determined by each implementor.

User-Defined Word. A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable. A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb. A word that expresses an action to be taken by a COBOL compiler or object program.

Word. A character-string of not more than 30 characters which forms a user-defined word, a system-name, or a reserved word.

Working-Storage Section. The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

77-Level-Description-Entry. A data description entry that describes a noncontiguous data item with the level-number 77.

5. OVERALL LANGUAGE CONSIDERATION

5.1 INTRODUCTION

The language considerations and rules specified in this chapter, apply to the highest level of the American National Standard COBOL. When a particular level of a module does not allow all of these language concepts, the restrictions will be pointed out in the chapter describing that language element. It should also be noted that restrictions contained in one module might possibly affect other modules. For example, series connectives are not allowed in Level 1 of the Nucleus; therefore, any module which is combined with Level 1 of the Nucleus would have the same restriction. The flowcharts in this document illustrate the logic of the statement under which they are contained and are not meant to dictate implementation.

5.2 NOTATION USED IN FORMATS AND RULES

5.2.1 Definition of a General Format

A general format is the specific arrangement of the elements of a clause or a statement. A clause or a statement consists of elements as defined below. 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. Throughout this document, specifications unique to the high level are enclosed in boxes.

5.2.1.1 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.

5.2.1.2 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.

5.2.1.3 Elements

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

5.2.1.4 Words

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

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

5.2.1.5 Level-Numbers

When specific level-numbers appear in data description entry formats, those specific level-numbers are required when such entries are used in a COBOL program. In this document, the form 01, 02, ... , 09 is used to indicate level-numbers 1 through 9.

5.2.1.6 Brackets and Braces

When a portion of a general format is enclosed in brackets, [ ] , that portion may be included or omitted at the user's choice. Braces, { } , enclosing a portion of a general format means a selection of one of the options contained within the braces must be made. In both cases, a choice is indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of a format, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies. (See paragraph 5.2.1.7, The Ellipsis.) If an option within braces contains only reserved words that are not key words, then the option is a default option (implicitly selected unless one of the other options is explicitly indicated).

5.2.1.7 The Ellipsis

In text, the ellipsis (...) may show the omission of a portion of a source program. 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.

5.2.1.8 Format Punctuation

The punctuation characters comma and semicolon are shown in some formats. Where shown in the formats, they are optional and may be included or omitted by the user. In the source program these two punctuation characters are

interchangeable and either one may be used anywhere one of them is shown in the formats. Neither one may appear immediately preceding the first clause of an entry or paragraph.

If desired, a semicolon or comma may be used between statements in the Procedure Division.

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

5.2.1.9 Use of Certain Special Characters in Formats

The characters '+', '-', '>', '<', '=', when appearing in formats, although not underlined, are required when such formats are used.

5.3 LANGUAGE CONCEPTS

5.3.1 Character Set

The most basic and indivisible unit of the language is the character. The set of characters used to form COBOL character-strings and separators includes the letters of the alphabet, digits and special characters. The character set consists of 51 characters as defined under COBOL Character Set in the glossary on page I-54. In the case of nonnumeric literals, comment-entries, and comment lines, the character set is expanded to include the computer's entire character set. The characters allowable in each type of character-string and as separators are defined in paragraph 5.3.2 and in the glossary beginning on page I-52.

Since the character set of a particular computer may not have the characters defined, single character substitution must be made as required. When such a character set contains fewer than 51 characters, double characters must be substituted for the single characters.

5.3.2 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.

5.3.2.1 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 page I-73, Format Punctuation), by statement and sentence structure definitions (see page I-101, Statements and Sentences), or reference format rules (see page I-105, Reference Format).

(3) The punctuation characters right and left parenthesis are separators. Parentheses 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 page I-106, Continuation of Lines.)

(5) Pseudo-text delimiters are separators. An opening pseudo-text delimiter must be immediately preceded by a space; a closing pseudo-text delimiter must be immediately followed by one of the separators space, comma, semicolon, or period.

Pseudo-text delimiters may appear only in balanced pairs delimiting pseudo-text.

(6) The separator space may optionally immediately precede all separators except:

a. As specified by reference format rules (see page I-105, Reference Format), and

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.

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

(7) The separator space may optionally 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 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.

5.3.2.2 Character-Strings

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

5.3.2.2.1 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.

5.3.2.2.1.1 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', '0', ... '9', and '-', except that the '-' may not appear as the first or last character.

There are seventeen (17) types of user-defined words:

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

Within a given source program, fifteen (15) of these seventeen (17) types of user-defined words are grouped into thirteen (13) disjoint sets. The disjoint sets are:

alphabet-names
cd-names
condition-names, data-names, and record-names
file-names
index-names
library-names
mnemonic-names
paragraph-names
program-names
report-names
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, either because no other user-defined word in the same source program has identical spelling or punctuation, or because uniqueness can be insured by qualification. (See page I-87, Uniqueness of Reference.)

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.

5.3.2.2.1.1.1 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 implementor-defined 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.

5.3.2.2.1.1.2 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 page II-8, The SPECIAL-NAMES Paragraph.)

5.3.2.2.1.1.3 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.

5.3.2.2.1.1.4 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.

5.3.2.2.1.1.5 Other User-Defined Names

See the glossary beginning on page I-52 for definitions of all other types of user-defined words.

5.3.2.2.1.2 System-Names

A system-name is a COBOL word which is used to communicate with the operating environment. Rules for the formation of a system-name are defined by the implementor, except that each character used in the formation of a system-name must be selected from the set of characters 'A', 'B', 'C', ... 'Z', '0', ... '9', and '-', except that the '-' may not appear as the first or last character.

There are three (3) types of system-names:

computer-name
implementor-name
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 on page I-78 are individually defined in the glossary beginning on page I-52.

5.3.2.2.1.3 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 page I-109, Reserved Words.)

There are six (6) types of reserved words:

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

5.3.2.2.1.3.1 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.

5.3.2.2.1.3.2 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.

5.3.2.2.1.3.3 Connectives

There are three types of connectives:

(1) Qualifier connectives that are used to associate a data-name, a condition-name, a text-name, or a paragraph-name with its qualifier: OF, IN

(2) Series connectives that link two or more consecutive operands:
, (separator comma) or ; (separator semicolon)

(3) Logical connectives that are used in the formation of conditions:
AND, OR

5.3.2.2.1.3.4 Special Registers

Certain reserved words are used to name and reference special registers. Special registers are certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features. These special registers include the following: LINAGE-COUNTER (see page IV-3), LINE-COUNTER (see page VIII-1), PAGE-COUNTER (see page VIII-1), and DEBUG-ITEM (see page XI-1).

5.3.2.2.1.3.5 Figurative Constants

Certain reserved words are used to name and reference specific constant values. These reserved words are specified on page I-81, Figurative Constant Values.

5.3.2.2.1.3.6 Special-Character Words

The arithmetic operators and relation characters are reserved words. (See the glossary beginning on page I-52.)

5.3.2.2.2 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.

5.3.2.2.2.1 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. The implementor must allow for nonnumeric literals of 1 through 120 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 page II-18, The PICTURE Clause.)

5.3.2.2.2.2 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 implementor must allow 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 quantity represented by the characters in the numeric literal. Every numeric literal is category numeric. (See page II-18, The PICTURE Clause.) The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

5.3.2.2.2.3 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 as follows:

ZERO ZEROS ZEROES	Represents the value '0', or one or more of the character '0', depending on context.
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 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, as when the figurative constant appears in a DISPLAY, STRING, STOP or UNSTRING statement, the length of the string is one character.

A figurative constant may be used wherever a literal appears in a format, except that whenever the literal is restricted to 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 page II-6, The OBJECT-COMPUTER Paragraph, and page II-8, The SPECIAL-NAMES Paragraph.)

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.

5.3.2.2.3 PICTURE Character-Strings

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. See page II-18, The PICTURE Clause, for the discussion of 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.

5.3.2.2.4 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.

5.3.3 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 COBOL character set to describe nonnumeric data items.

5.3.3.1 Logical Record and File Concept

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

5.3.3.1.1 Physical Aspects of a File

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

(1) The grouping of logical records within the physical limitations of the file medium.

(2) The means by which the file can be identified.

5.3.3.1.2 Conceptual Characteristics of a File

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. A COBOL logical record is a group of related information, uniquely identifiable, and treated as a unit.

A physical record is a physical unit of information whose size and recording mode is convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware dependent and bears no direct relationship to the size of the file of information contained on a device.

A logical record may be contained within a single physical unit; or several logical records may be contained within a single physical unit; or, in the case of mass storage files, a logical record may require more than one physical unit to contain it. There are several source language methods available for describing the relationship of logical records and physical units. When a permissible relationship has been established, control of the accessibility of logical records as related to the physical unit must be provided by the interaction of the object program on the implementor's hardware and/or software system. In this document, references to records means to logical records, unless the term 'physical record' is specifically used.

The concept of a logical record is not restricted to file data but is carried over into the definition of working storage. Thus, working storage may be grouped into logical records and defined by a series of record description entries.

5.3.3.1.3 Record Concepts

The 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.

5.3.3.2 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.

5.3.3.2.1 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. 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.

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.

5.3.3.3 Concept of Classes of Data

The five categories of data items (see page II-18, The PICTURE Clause) 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. The following chart depicts the relationship of the class and categories of data items.

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited
		Alphanumeric Edited
		Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic
		Numeric
		Numeric Edited
		Alphanumeric Edited
		Alphanumeric

5.3.3.4 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 selection of radix is generally dependent upon the arithmetic capability of the computer. If more than one arithmetic radix is provided, the selection is dependent upon factors included in such clauses as USAGE. The binary-coded decimal form is also used to represent characters and symbols that are alphanumeric items.

The selection of the proper binary-coded alphanumeric or binary-coded decimal form is dependent upon the capability of the computer and its external media.

When a computer provides more than one means of representing data, the standard data format must be used if not otherwise specified by the data description. If both the external medium and the computer are capable of handling more than one form of data representation, or if there is no external medium associated with the data, the selection is dependent on factors

included in USAGE, PICTURE, etc., clauses. Each implementor provides a complete explanation of the possible forms on the computer for which he is implementing COBOL. The method used in selecting the proper data form is also provided to allow the programmer to anticipate and/or control the selection.

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

5.3.3.5 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 the implementor.

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

5.3.3.6 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 1a above.

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

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

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in the JUSTIFIED clause on page II-16.

5.3.3.7 Item Alignment for Increased Object-Code Efficiency

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

However, certain uses of data (e.g., in arithmetic operations or in subscripting) may be facilitated if the data is stored so as to be aligned on these natural boundaries. Specifically, additional machine operations in the object program may be required for the accessing and storage of data if portions of two or more data items appear between adjacent natural boundaries, or if certain natural boundaries bifurcate a single data item.

Data items which are aligned on these natural boundaries in such a way as to avoid such additional machine operations are defined to be synchronized. A synchronized item is assumed to be introduced and carried in that form; conversion to synchronized form occurs only during the execution of a procedure (other than READ or WRITE) which stores data in the item.

Synchronization can be accomplished in two ways:

(1) By use of the SYNCHRONIZED clause

(2) By recognizing the appropriate natural boundaries and organizing the data suitably without the use of the SYNCHRONIZED clause. (See page II-34, The SYNCHRONIZED Clause, General Rule 9.)

Each implementor who provides for special types of alignment will specify the precise interpretations which are to be made.

5.3.3.8 Uniqueness of Reference

5.3.3.8.1 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.

5.3.3.8.2 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 page III-2, The OCCURS Clause).

The subscript can be represented either by a numeric literal that is an integer or by a data-name. The data-name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name may be qualified but not subscripted. In the Report Section, neither a sum counter nor the special registers LINE-COUNTER and PAGE-COUNTER can be used as a subscript.

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

5.3.3.8.3 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 either a SET, a SEARCH ALL, or a Format 4 PERFORM statement.

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the 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 (1) nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resultant 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                    }
```

5.3.3.8.4 Identifier

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

The general formats for identifiers are:

Format 1

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

   [, subscript-3]] )]
```

Format 2

```
data-name-1  [{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 qualification, 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, subscript or qualifier.

(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.

5.3.3.8.5 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.

5.3.4 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.

5.3.4.1 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 UNTII 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 item contributes to the execution of the statement.

5.3.4.2 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 page I-103, Categories of Statements.) 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.

5.3.4.3 Explicit and Implicit Attributes

Attributes may be implicitly or explicitly 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.

5.4 IDENTIFICATION DIVISION

5.4.1 General Description

The Identification Division must be included in every 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.

5.4.2 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.

5.4.3 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.

5.4.3.1 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] ... ]
```

5.5 ENVIRONMENT DIVISION

5.5.1 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.

5.5.2 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 implementor-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 paragraph 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.

5.5.3 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.

5.5.3.1 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] ]
```

5.5.3.2 Syntax Rules

(1) The Environment Division begins with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

5.6 DATA DIVISION

5.6.1 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:

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

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

c. Constants which are defined by the user.

5.6.2 Physical and Logical Aspects of Data Description

5.6.2.1 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, Linkage, Communication, and Report 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. When the file description specifies a file to be used as a Report Writer output file, no record description entries are permitted for that file. Report description entries appear in a separate section of the Data Division, the Report Section. 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. The Communication Section describes the data item in the source program that will serve as the interface between the MCS and the program. The Report Section describes the content and format of reports that are to be generated.

5.6.2.2 Data Division Structure

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] ...] ...]
[REPORT SECTION.
     [report-description-entry {report-group-description-entry} ...] ...]
```

5.7 PROCEDURE DIVISION

5.7.1 General Description

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

5.7.1.1 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 pages IV-32, V-30, VI-32, VIII-56, and XI-4 for the USE statement.)

5.7.1.2 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.

5.7.1.3 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.

5.7.1.4 Procedure Division Structure

5.7.1.4.1 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] ... ].
```

5.7.1.4.2 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] ... } ...
```

5.7.2 Statements and Sentences

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

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

5.7.2.1 Conditional Statements and Conditional Sentences

5.7.2.1.1 Definition of 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:

a. An IF, SEARCH or RETURN statement.

b. A READ statement that specifies the AT END or INVALID KEY phrase.

c. A WRITE statement that specifies the INVALID KEY or END-OF-PAGE phrase.

d. A START, REWRITE or DELETE statement that specifies the INVALID KEY phrase.

e. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.

f. A RECEIVE statement that specifies a NO DATA phrase.

g. A STRING, UNSTRING or CALL statement that specifies the ON OVERFLOW phrase.

5.7.2.1.2 Definition of Conditional Sentence

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

5.7.2.2 Compiler Directing Statements and Compiler Directing Sentences

5.7.2.2.1 Definition of 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 page X-2, The COPY Statement; page II-63, The ENTER Statement; and The USE Statement on pages IV-32, V-30, VI-32, VIII-56, and XI-4). A compiler directing statement causes the compiler to take a specific action during compilation.

5.7.2.2.2 Definition of Compiler Directing Sentence

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

5.7.2.3 Imperative Statements and Imperative Sentences

5.7.2.3.1 Definition of 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. The imperative verbs are:

ACCEPT	GENERATE	SEND
ADD (1)	GO	SET
ALTER	INITIATE	SORT
CALL (3)	INSPECT	START (2)
CANCEL	MERGE	STOP
CLOSE	MOVE	STRING (3)
COMPUTE (1)	MULTIPLY (1)	SUBTRACT (1)
DELETE (2)	OPEN	SUPPRESS
DISABLE	PERFORM	TERMINATE
DISPLAY	READ (5)	UNSTRING (3)
DIVIDE (1)	RECEIVE (4)	WRITE (6)
ENABLE	RELEASE	
EXIT	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 NO DATA phrase.
(5) Without the optional AT END phrase or INVALID KEY phrase.
(6) 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 or a WHEN phrase associated with a previous SEARCH statement.

5.7.2.3.2 Definition of Imperative Sentence

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

5.7.2.4 Categories of Statements

Category	Verbs
Arithmetic	ADD COMPUTE DIVIDE INSPECT (TALLYING) MULTIPLY SUBTRACT
Compiler Directing	COPY ENTER USE
Conditional	ADD (SIZE ERROR) CALL (OVERFLOW) COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DIVIDE (SIZE ERROR) IF MULTIPLY (SIZE ERROR) READ (END or INVALID KEY) RECEIVE (NO DATA) RETURN (END) REWRITE (INVALID KEY) SEARCH START (INVALID KEY) STRING (OVERFLOW) SUBTRACT (SIZE ERROR) UNSTRING (OVERFLOW) WRITE (INVALID KEY or END-OF-PAGE)
Data Movement	ACCEPT (DATE, DAY, or TIME) ACCEPT MESSAGE COUNT INSPECT (REPLACING) MOVE STRING UNSTRING
Ending	STOP

Category	Verbs
Input-Output	ACCEPT (identifier) CLOSE DELETE DISABLE DISPLAY ENABLE OPEN READ RECEIVE REWRITE SEND START STOP (literal) WRITE
Inter-Program Communicating	CALL CANCEL
Ordering	MERGE RELEASE RETURN SORT
Procedure Branching	ALTER CALL EXIT GO TO PERFORM
Report Writing	GENERATE INITIATE SUPPRESS TERMINATE
Table Handling	SEARCH SET

IF is a verb in the COBOL sense; it is recognized that it is not a verb in English.

5.7.2.4.1 Specific Statement Formats

The specific statement formats, together with a detailed discussion of the restrictions and limitations associated with each, appear in alphabetic sequence in the appropriate section of this document. (See the index beginning on page XV-1 to determine the page containing the discussion of a specific verb.)

5.8 REFERENCE FORMAT

5.8.1 General Description

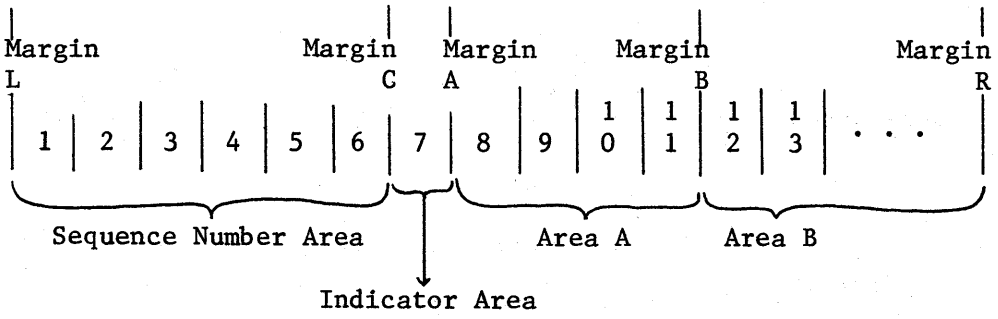
The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions in a line on an input-output medium. Each implementor must define what is meant by lines and character positions for each input-output medium used with his compiler. Within these definitions, each 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.

5.8.2 Reference Format Representation

The reference format for a line is represented as follows:



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.

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 a finite number of character positions specified by the implementor; it begins immediately to the right of margin B and terminates immediately to the left of margin R.

5.8.2.1 Sequence Numbers

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

5.8.2.2 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.

5.8.2.3 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 paragraph 5.8.2.2 above.)

5.8.3 Division, Section, Paragraph Formats

5.8.3.1 Division Header

The division header must start in area A.

5.8.3.2 Section Header

The section header must start in area A.

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

5.8.3.3 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 as indicated in paragraph 5.8.6 on page I-108. 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 paragraph 5.8.2.2 on page I-106.

5.8.4 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, SD, RD, CD.

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 with 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, 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 a level-number 01 or 77, the level-number begins in area A followed by a space and followed in area B by its associated record-name or item-name and appropriate descriptive information.

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.

5.8.5 Declaratives

The key word DECLARATIVES and the key words END DECLARATIVES that precede and follow, respectively, the declaratives portion of the Procedure Division must appear on a line by itself. Each must begin in area A and be followed by a period and a space.

5.8.6 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. 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.

5.9 Reserved Words

The following is a list of reserved words:

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

BEFORE
BLANK
BLOCK
BOTTOM
BY

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

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

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

FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

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

JUST
JUSTIFIED

KEY

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

MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC

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

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE

PLUS
POINTER
POSITION
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES

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

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

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

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

+
-
*
/
**
>
<
=

6. COMPOSITE LANGUAGE SKELETON

6.1 GENERAL DESCRIPTION

This chapter contains the composite language skeleton of the American National Standard COBOL. It is intended to display complete and syntactically correct formats.

The leftmost margin on pages I-112 through I-123 is equivalent to margin A in a COBOL source program. The first indentation after the leftmost margin is equivalent to margin B in a COBOL source program. (See page I-105 for description of margin A and margin B.)

On pages I-124 through I-132 the leftmost margin indicates the beginning of the format for a new COBOL verb. The first indentation after the leftmost margin indicates continuation of the format of the COBOL verb.

The following is a summary of the formats shown on pages I-112 through I-135.

Page I-112: Identification Division general format

Page I-113: Environment Division general format

Page I-115: The three formats of the file control entry

Page I-117: Data Division general format

Page I-119: The three formats for a data description entry

Page I-120: The two general formats for a communication description entry

Page I-121: The three formats for a report group description entry

Page I-123: Procedure Division general format

Page I-124: General format of verbs listed in alphabetical order

Page I-133: General format for conditions

Page I-134: Formats for qualification, subscripting, indexing, and an identifier

Page I-135: General format for COPY statement

GENERAL FORMAT FOR IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

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

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

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

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

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

GENERAL FORMAT FOR ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

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

OBJECT-COMPUTER.   computer-name

    [, MEMORY SIZE integer {WORDS     }]
                           {CHARACTERS}
                           {MODULES   }
    [, PROGRAM COLLATING SEQUENCE IS alphabet-name]
    [, SEGMENT-LIMIT IS segment-number] .

[SPECIAL-NAMES. [, implementor-name

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

    [, alphabet-name IS {STANDARD-1                                                        }
                        {NATIVE                                                            }
                        {implementor-name                                                  }
                        {literal-1 [{THROUGH} literal-2                                    }
                                   {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] .]
```

GENERAL FORMAT FOR ENVIRONMENT DIVISION

```
[INPUT-OUTPUT SECTION.

 FILE-CONTROL.

    {file-control-entry} ...

[I-O-CONTROL.
   [; RERUN [ON {file-name-1      }]
                {implementor-name}

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

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

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

GENERAL FORMAT FOR FILE CONTROL ENTRY

FORMAT 1:

```
SELECT [OPTIONAL] file-name

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

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

   [; ORGANIZATION IS SEQUENTIAL]

   [; ACCESS MODE IS SEQUENTIAL]

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

FORMAT 2:

```
SELECT file-name

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

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

   ; ORGANIZATION IS RELATIVE

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

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

GENERAL FORMAT FOR FILE CONTROL ENTRY

FORMAT 3:

```
SELECT file-name

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

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

    ; ORGANIZATION IS INDEXED

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

    ; RECORD KEY IS data-name-1

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

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

FORMAT 4:

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

GENERAL FORMAT FOR DATA DIVISION

```
DATA DIVISION.
[ FILE SECTION.
[ FD  file-name
    [ ; BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS / CHARACTERS } ]
    [ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
      ; LABEL { RECORD IS / RECORDS ARE } { STANDARD / OMITTED }
    [ ; VALUE OF implementor-name-1 IS { data-name-1 / literal-1 }
          [ , implementor-name-2 IS { data-name-2 / literal-2 } ] ... ]
    [ ; DATA { RECORD IS / RECORDS ARE } data-name-3 [, data-name-4] ... ]
    [ ; LINAGE IS { data-name-5 / integer-5 } LINES [, WITH FOOTING AT { data-name-6 / integer-6 } ]
          [ , LINES AT TOP { data-name-7 / integer-7 } ] [ , LINES AT BOTTOM { data-name-8 / integer-8 } ] ]
    [ ; CODE-SET IS alphabet-name ]
    [ ; { REPORT IS / REPORTS ARE } report-name-1 [, report-name-2] ... ] .
[record-description-entry ] ... ] ...

[SD  file-name
    [ ; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS ]
    [ ; DATA { RECORD IS / RECORDS ARE } data-name-1 [, data-name-2] ... ] .
{record-description-entry} ... ] ... ]

[WORKING-STORAGE SECTION.
[ 77-level-description-entry / record-description-entry ] ... ]
```

GENERAL FORMAT FOR DATA DIVISION

```
[LINKAGE SECTION.

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

[COMMUNICATION SECTION.

[communication-description-entry

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

[REPORT SECTION.

[RD  report-name

     [; CODE literal-1 ]

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

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

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

        [, FOOTING integer-5 ] ] .

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

GENERAL FORMAT FOR DATA DESCRIPTION ENTRY

FORMAT 1:

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

  [; REDEFINES data-name-2]

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

  [; [USAGE IS] {COMPUTATIONAL}]
                {COMP        }
                {DISPLAY     }
                {INDEX       }

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

  [; OCCURS {integer-1 TO integer-2 TIMES DEPENDING ON data-name-3}
            {integer-2 TIMES                                     }

       [{ASCENDING } KEY IS data-name-4 [, data-name-5] ... ] ...
        {DESCENDING}

       [INDEXED BY index-name-1 [, index-name-2] ...]]

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

  [; {JUSTIFIED} RIGHT]
     {JUST     }

  [; BLANK WHEN ZERO]

  [; VALUE IS literal] .
```

FORMAT 2:

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

FORMAT 3:

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

    [, literal-3 [{THROUGH} literal-4 ]] ...  .
                 {THRU   }
```

GENERAL FORMAT FOR COMMUNICATION DESCRIPTION ENTRY

FORMAT 1:

```
CD  cd-name;
                                    [ [; SYMBOLIC QUEUE IS data-name-1]
                                          [; SYMBOLIC SUB-QUEUE-1 IS data-name-2]
                                          [; SYMBOLIC SUB-QUEUE-2 IS data-name-3]
                                          [; SYMBOLIC SUB-QUEUE-3 IS data-name-4]
                                          [; MESSAGE DATE IS data-name-5]
    FOR [INITIAL] INPUT                   [; MESSAGE TIME IS data-name-6]
                                          [; SYMBOLIC SOURCE IS data-name-7]
                                          [; TEXT LENGTH IS data-name-8]
                                          [; END KEY IS data-name-9]
                                          [; STATUS KEY IS data-name-10]
                                          [; MESSAGE COUNT IS data-name-11] ]
                                    [data-name-1, data-name-2, ..., data-name-11] ]
```

FORMAT 2:

```
CD  cd-name; FOR OUTPUT
    [; DESTINATION COUNT IS data-name-1]
    [; TEXT LENGTH IS data-name-2]
    [; STATUS KEY IS data-name-3]
    [; DESTINATION TABLE OCCURS integer-2 TIMES
          [; INDEXED BY index-name-1  [, index-name-2]... ] ]
    [; ERROR KEY IS data-name-4]
    [; SYMBOLIC DESTINATION IS data-name-5] .
```

GENERAL FORMAT FOR REPORT GROUP DESCRIPTION ENTRY

FORMAT 1:

```
01  [data-name-1]
    [; LINE NUMBER IS {integer-1 [ON NEXT PAGE]}]
                      {PLUS integer-2           }
    [; NEXT GROUP IS {integer-3}]
                     {PLUS integer-4}
                     {NEXT PAGE    }
    ; TYPE IS {{REPORT HEADING}                                }
              {{RH           }                                }
              {{PAGE HEADING }                                }
              {{PH           }                                }
              {{CONTROL HEADING} {data-name-2}                 }
              {{CH             } {FINAL     }                 }
              {{DETAIL       }                                }
              {{DE           }                                }
              {{CONTROL FOOTING} {data-name-3}                 }
              {{CF             } {FINAL     }                 }
              {{PAGE FOOTING }                                }
              {{PF           }                                }
              {{REPORT FOOTING}                               }
              {{RF           }                                }
    [; [USAGE IS] DISPLAY] .
```

FORMAT 2:

```
level-number [data-name-1]
    [; LINE NUMBER IS {integer-1 [ON NEXT PAGE]}]
                      {PLUS integer-2           }
    [; [USAGE IS] DISPLAY] .
```

GENERAL FORMAT FOR REPORT GROUP DESCRIPTION ENTRY

FORMAT 3:

```
level-number  [data-name-1]
    [; BLANK WHEN ZERO]
    [; GROUP INDICATE]
    [; {JUSTIFIED} RIGHT]
       {JUST     }
    [; LINE NUMBER IS {integer-1 [ON NEXT PAGE]}]
                      {PLUS integer-2          }
    [; COLUMN NUMBER IS integer-3]
     ; {PICTURE} IS character-string
       {PIC    }
  {  ; SOURCE IS identifier-1                                    }
  {  ; VALUE IS literal                                          }
  {{; SUM identifier-2 [, identifier-3] ...                      }
  {    [UPON data-name-2 [, data-name-3] ...]} ...               }
  {    [RESET ON {data-name-4}]                                  }
  {              {FINAL     }                                    }
    [;  [USAGE IS]  DISPLAY ] .
```

GENERAL FORMAT FOR PROCEDURE DIVISION

FORMAT 1:

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

[DECLARATIVES.

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

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

END DECLARATIVES.]

{section-name SECTION [segment-number] .

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

FORMAT 2:

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

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

GENERAL FORMAT FOR VERBS

```
ACCEPT identifier [ FROM mnemonic-name]

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

ACCEPT cd-name MESSAGE COUNT

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

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

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]

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

  [; ON SIZE ERROR imperative-statement]

ALTER procedure-name-1 TO [ PROCEED TO ] procedure-name-2

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

CALL { identifier-1 } [USING data-name-1 [, data-name-2] ... ]
     { literal-1    }

  [; ON OVERFLOW imperative-statement ]

CANCEL { identifier-1 } [, identifier-2 ] ...
       { literal-1    } [, literal-2    ]

                    [ { REEL } [WITH NO REWIND]  ]
                    [ { UNIT } [FOR REMOVAL     ]  ]
CLOSE file-name-1   [                           ]
                    [ WITH { NO REWIND }        ]
                    [      { LOCK      }        ]

    [                                                          ]
    [                   [ { REEL } [WITH NO REWIND]  ]         ]
    [                   [ { UNIT } [FOR REMOVAL     ]  ]       ]
    [, file-name-2      [                           ]  ]       ] ...
    [                   [ WITH { NO REWIND }        ]  ]       ]
    [                   [      { LOCK      }        ]  ]       ]
    [                                                          ]

CLOSE file-name-1 [WITH LOCK] [, file-name-2 [WITH LOCK] ] ...
```

GENERAL FORMAT FOR VERBS

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

DELETE file-name RECORD [; INVALID KEY imperative-statement]

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

DISPLAY {identifier-1} [, identifier-2] ... [UPON mnemonic-name]
        {literal-1   } [, literal-2   ]

DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
       {literal-1   }
  [, identifier-3 [ROUNDED]] ... [; ON SIZE ERROR imperative-statement ]

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

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

DIVIDE {identifier-1} INTO {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   }      {literal-2   }
   REMAINDER identifier-4 [; ON SIZE ERROR imperative-statement]

DIVIDE {identifier-1} BY {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   }    {literal-2   }
   REMAINDER identifier-4 [; ON SIZE ERROR imperative-statement]

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

ENTER language-name [routine-name] .

EXIT [PROGRAM] .

GENERATE {data-name  }
         {report-name}

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

GENERAL FORMAT FOR VERBS

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

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

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

INSPECT identifier-1 TALLYING

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

INSPECT identifier-1 REPLACING

{CHARACTERS BY {identifier-6} [{BEFORE} INITIAL {identifier-7}]                                                                  }
               {literal-4    } [{AFTER }         {literal-5   }]                                                                 }
{{, {ALL       } {, {identifier-5} BY {identifier-6} [{BEFORE} INITIAL {identifier-7}]} ...} ...                                 }
 {  {LEADING   }    {literal-3   }     {literal-4   }  [{AFTER }         {literal-5   }]}                                       }
 {  {FIRST     }                                                                                                               }

INSPECT identifier-1 TALLYING

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

   REPLACING

{  CHARACTERS BY {identifier-6} [{BEFORE} INITIAL {identifier-7}]                                                                 }
                 {literal-4    } [{AFTER }         {literal-5   }]                                                                }
{{, {ALL       } {, {identifier-5} BY {identifier-6} [{BEFORE} INITIAL {identifier-7}]} ...} ...                                 }
 {  {LEADING   }    {literal-3   }     {literal-4   }  [{AFTER }         {literal-5   }]}                                       }
 {  {FIRST     }                                                                                                               }
```

GENERAL FORMAT FOR VERBS

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

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

  [COLLATING SEQUENCE IS alphabet-name]

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

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

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

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

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

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

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

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

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

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

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

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

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

GENERAL FORMAT FOR VERBS

```
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-3   }

   [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    }

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

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

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

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

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

RELEASE record-name [FROM identifier]

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

REWRITE record-name [FROM identifier]

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

GENERAL FORMAT FOR VERBS

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

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

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

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

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

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

          { imperative-statement-2 }
          { NEXT SENTENCE          }

SEND cd-name FROM identifier-1

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

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

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

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

GENERAL FORMAT FOR VERBS

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1 [ , data-name-2 ] ...
                    { DESCENDING }
        [ ON { ASCENDING  } KEY data-name-3 [ , data-name-4 ] ... ] ...
             { DESCENDING }
  [ COLLATING SEQUENCE IS alphabet-name ]
  { INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
  { USING file-name-2 [ , file-name-3 ] ...                         }
  { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
  { GIVING file-name-4                                               }
```

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

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

```
STRING { identifier-1 } [ , identifier-2 ] ... DELIMITED BY { identifier-3 }
       { literal-1    }   [ , literal-2    ]                { literal-3    }
                                                           { SIZE         }
    [ , { identifier-4 } [ , identifier-5 ] ... DELIMITED BY { identifier-6 } ] ...
        { literal-4    }   [ , literal-5    ]                { literal-6    }
                                                            { SIZE         }
  INTO identifier-7 [ WITH POINTER identifier-8 ]
  [ ; ON OVERFLOW imperative-statement ]
```

```
SUBTRACT { identifier-1 } [ , identifier-2 ] ... FROM identifier-m [ ROUNDED ]
         { literal-1    }   [ , literal-2    ]
  [ , identifier-n [ ROUNDED ] ] ... [ ; ON SIZE ERROR imperative-statement ]
```

GENERAL FORMAT FOR VERBS

```
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]

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

SUPPRESS PRINTING

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

UNSTRING identifier-1
  [DELIMITED BY [ALL] {identifier-2} [, OR [ALL] {identifier-3}] ...]
                      {literal-1   }             {literal-2   }
    INTO identifier-4 [, DELIMITER IN identifier-5] [, COUNT IN identifier-6]
      [, identifier-7 [, DELIMITER IN identifier-8] [, COUNT IN identifier-9]] ...
  [WITH POINTER identifier-10] [TALLYING IN identifier-11]
  [; ON OVERFLOW imperative-statement]

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

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

USE BEFORE REPORTING identifier.
```

GENERAL FORMAT FOR VERBS

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

   [    cd-name-2                                  ]
   [    [ALL REFERENCES OF] identifier-2           ]
   [ ,  file-name-2                                ] ...  .
   [    procedure-name-2                           ]
   [    ALL PROCEDURES                             ]

WRITE record-name [FROM identifier-1]

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

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

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

GENERAL FORMAT FOR CONDITIONS

RELATION CONDITION:

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

CLASS CONDITION:

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

SIGN CONDITION:

```
arithmetic-expression is [NOT] {POSITIVE}
                               {NEGATIVE}
                               {ZERO    }
```

CONDITION-NAME CONDITION:

```
condition-name
```

SWITCH-STATUS CONDITION:

```
condition-name
```

NEGATED SIMPLE CONDITION:

```
NOT simple-condition
```

COMBINED CONDITION:

```
condition {{AND} condition} ...
          {{OR }          }
```

ABBREVIATED COMBINED RELATION CONDITION:

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

MISCELLANEOUS FORMATS

QUALIFICATION:

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

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

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

SUBSCRIPTING:

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

INDEXING:

```
{data-name     } ( {index-name-1 [{±} literal-2]} [, {index-name-2 [{±} literal-4]} [, {index-name-3 [{±} literal-6]}]] )
{condition-name}   {literal-1                    }  [ {literal-3                    }]  [ {literal-5                    }]
```

IDENTIFIER: FORMAT 1

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

IDENTIFIER: FORMAT 2

```
data-name-1 [ {OF} data-name-2 ] ... [ ( {index-name-1 [{±} literal-2]} [, {index-name-2 [{±} literal-4]} [, {index-name-3 [{±} literal-6]}]] ) ]
            [ {IN}             ]       {literal-1                    }  [ {literal-3                    }]  [ {literal-5                    }]
```

GENERAL FORMAT FOR COPY STATEMENT

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

     [ REPLACING { ,  {==pseudo-text-1==} } BY {==pseudo-text-2==} } ... ]
                 {    {identifier-1      }     {identifier-2      }
                 {    {literal-1         }     {literal-2         }
                 {    {word-1            }     {word-2            }
```

1. INTRODUCTION TO THE NUCLEUS

1.1 FUNCTION

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

1.2 LEVEL CHARACTERISTICS

Nucleus Level 1 does not provide full COBOL facilities for qualification, punctuation characters, data-name formation, connectives, and figurative constants. Within the Procedure Division, the Nucleus Level 1 provides limited capabilities for the ACCEPT, ADD, ALTER, DIVIDE, DISPLAY, IF, INSPECT, MOVE, MULTIPLY, PERFORM, and SUBTRACT statements and full capabilities for the ENTER, EXIT, GO, and STOP statements.

Nucleus Level 2 provides full facilities for qualification, punctuation characters, data-name formation, connectives, and figurative constants. Within the Procedure Division, the Nucleus Level 2 provides full capabilities for the ACCEPT, ADD, ALTER, DIVIDE, DISPLAY, IF, INSPECT, MOVE, MULTIPLY, PERFORM, and SUBTRACT statements.

1.3 LEVEL RESTRICTIONS ON OVERALL LANGUAGE

1.3.1 Format Notation

The separators, comma and semicolon, are not included in Level 1. The comma and semicolon are not boxed within the general formats of this document in order to simplify the formats. The separators, comma and semicolon, are included in Level 2.

1.3.2 Name Characteristics

All data-names must begin with an alphabetic character in Level 1. Qualification is not included, therefore, all data-names, paragraph-names, and text-names must be unique in Level 1. In Level 2 data-names need not begin with an alphabetic character; the alphabetic characters may be positioned anywhere within the data-name. Qualification is permitted in Level 2; thus all data-names, condition-names, paragraph-names, and text-names need not be unique.

1.3.3 Figurative Constants

The only figurative constants that may be used in Level 1 are: ZERO, SPACE, HIGH-VALUE, LOW-VALUE, and QUOTE. In Level 2, 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.

1.3.4 Reference Format

In Level 1 a word or numeric literal cannot be broken in such a way that part of it appears on a continuation line. In Level 2 a word or numeric literal can be broken in such a way that part of it appears on a continuation line.

2. IDENTIFICATION DIVISION IN THE NUCLEUS

2.1 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.

2.2 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.

2.2.1 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. Paragraphs 2.3 and 2.4 define the PROGRAM-ID paragraph and the DATE-COMPILED paragraph. While the other paragraphs are not defined, each general format is formed in the same manner.

2.2.1.1 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] ...]
```

2.2.1.2 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. The continuation of the comment-entry by the use of the hyphen in the indicator area is not permitted; however, the comment-entry may be contained on one or more lines.

2.3 THE PROGRAM-ID PARAGRAPH

2.3.1 Function

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

2.3.2 General Format

PROGRAM-ID. program-name.

2.3.3 Syntax Rules

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

2.3.4 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.

2.4 THE DATE-COMPILED PARAGRAPH

2.4.1 Function

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

2.4.2 General Format

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

2.4.3 Syntax Rules

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

2.4.4 General Rules

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

```
DATE-COMPILED.  current date.
```

3. ENVIRONMENT DIVISION IN THE NUCLEUS

3.1 CONFIGURATION SECTION

3.1.1 The SOURCE-COMPUTER Paragraph

3.1.1.1 Function

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

3.1.1.2 General Format

SOURCE-COMPUTER. computer-name.

3.1.1.3 Syntax Rules

(1) Computer-name is a system-name.

3.1.1.4 General Rules

(1) Fixed computer-names are assigned by the individual implementor.

(2) The computer-name may provide a means for identifying equipment configuration, in which case the computer-name and its implied configuration are specified by each implementor.

3.1.2 The OBJECT-COMPUTER Paragraph

3.1.2.1 Function

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

3.1.2.2 General Format

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

3.1.2.3 Syntax Rules

(1) Computer-name is a system-name.

3.1.2.4 General Rules

(1) The computer-name may provide a means for identifying equipment configuration, in which case the computer-name and its implied configuration are specified by each implementor. The configuration definition contains specific information concerning the memory size.

The implementor defines what is to be done if the subset specified by the user is less than the minimum configuration required for running the object program.

(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 page II-41, Relation Condition.)

b. Explicitly specified in condition-name conditions. (See page II-44, Condition-Name Condition (Conditional Variable).)

c. Implicitly specified by the presence of a CONTROL clause in a report description entry. (See page VIII-28, The CONTROL Clause.)

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

(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. (See page VII-8, The MERGE Statement, and page VII-14, The SORT Statement.)

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

3.1.3 The SPECIAL-NAMES Paragraph

3.1.3.1 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.

3.1.3.2 General Format

```
SPECIAL-NAMES. [, implementor-name

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

[, alphabet-name IS { STANDARD-1
                      NATIVE
                      implementor-name
                      literal-1 [ {THROUGH} literal-2
                                  {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] .
```

3.1.3.3 Syntax Rules

(1) 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.

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

(3) The words THRU and THROUGH are equivalent.

(4) In repetition, a comma may be used before implementor-name.

3.1.3.4 General Rules

(1) If the implementor-name is not a switch, the associated mnemonic-name may be used in the ACCEPT, DISPLAY, SEND, and WRITE statement.

(2) If the implementor-name is a switch, at least one condition-name must be associated with it. The status of the switch is specified by condition-names and interrogated by testing the condition-names (see page II-44, Switch-Status Condition).

(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 page II-6, The OBJECT-COMPUTER Paragraph) or the COLLATING SEQUENCE phrase of a SORT or MERGE statement (see page VII-8, The MERGE Statement, and page VII-14, The SORT Statement), the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry (see page IV-10, The File Description - Complete Entry Skeleton), the alphabet-name clause specifies a character code set.

a. If the STANDARD-1 phrase is specified, the character code set or collating sequence identified is that defined in American National Standard Code for Information Interchange, X3.4-1968. Each character of the standard character set is associated with its corresponding character of the native character set. The implementor defines the correspondence between the characters of the standard character set and the characters of the native character set for which there is no correspondence otherwise specified.

b. If the NATIVE phrase is specified, the native character code set or native collating sequence is used.

c. If the implementor-name phrase is specified, the character code set or collating sequence identified is that defined by the implementor. The implementor also defines the correspondence between characters of the character code set specified by implementor-name and the characters of the native character code set.

d. If the literal phrase is specified, the alphabet-name may not be referenced in a CODE-SET clause (see IV-12, The CODE-SET Clause). The collating sequence identified is that defined according to the following rules:

Rule 1: The value of each literal specifies:

1. The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.

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

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

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

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

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

(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.

a. digits 0 thru 9;

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

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

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.

4. DATA DIVISION IN THE NUCLEUS

4.1 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.

4.1.1 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 as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number, 77.

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

a. level-number 77
b. data-name
c. 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.

4.1.2 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.

4.1.3 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.

4.2 THE DATA DESCRIPTION - COMPLETE ENTRY SKELETON

4.2.1 Function

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

4.2.2 General Format

Format 1

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

  [; REDEFINES data-name-2 ]

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

  [; [USAGE IS] { COMPUTATIONAL } ]
                { COMP          }
                { 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-2 [ { THROUGH } data-name-3 ] .
                                      { THRU    }
```

Format 3

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

    [, literal-3 [ { THROUGH } literal-4 ] ] ... .
                 { THRU    }
```

4.2.3 Syntax Rules

(1) In Level 1, the level-number in Format 1 may be any number from 01-10 or 77. In Level 2, 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.

4.2.4 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 page III-5, The USAGE IS INDEX Clause).

4.3 THE BLANK WHEN ZERO CLAUSE

4.3.1 Function

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

4.3.2 General Format

BLANK WHEN ZERO

4.3.3 Syntax Rules

(1) The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited. (See page II-18, The PICTURE Clause)

4.3.4 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.

4.4 THE DATA-NAME OR FILLER CLAUSE

4.4.1 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.

4.4.2 General Format

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

4.4.3 Syntax Rules

(1) 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.

4.4.4 General Rules

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

4.5 THE JUSTIFIED CLAUSE

4.5.1 Function

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

4.5.2 General Format

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

4.5.3 Syntax Rules

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

(2) JUST is an abbreviation for JUSTIFIED.

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

4.5.4 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 page I-86, Standard Alignment Rules.)

4.6 LEVEL-NUMBER

4.6.1 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.

4.6.2 General Format

level-number

4.6.3 Syntax Rules

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

(2) Data description entries subordinate to an FD, SD or CD entry must have level-numbers with the values 01 thru 10 in Level 1; 01-49, 66 or 88 in Level 2. (See page IV-10 for FD, page VII-5 for SD, and page XIII-3 for CD.)

(3) Data description entries subordinate to an RD entry must have level-numbers with the value 01 thru 10 in Level 1; 01 thru 49 in Level 2. (See page VIII-4 for RD.)

(4) Data description entries in the Working-Storage Section and Linkage Section must have level-numbers with the values 01-10 or 77 in Level 1; 01-49, 66, 77 or 88 in Level 2.

4.6.4 General Rules

(1) The level-number 01 identifies the first entry in each record description or a report group. (See page VIII-6, The Report Group Description.)

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

a. Level-number 77 is assigned to identify noncontiguous working storage data items, noncontiguous linkage data items, and can be used only as described by Format 1 of the data description skeleton. (See page II-12, The Data Description - Complete Entry Skeleton.)

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. (See page II-12, The Data Description - Complete Entry Skeleton.)

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. (See page II-12, The Data Description - Complete Entry Skeleton.)

(3) Multiple level 01 entries subordinate to any given level indicator, other than RD, represent implicit redefinitions of the same area.

4.7 THE PICTURE CLAUSE

4.7.1 Function

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

4.7.2 General Format

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

4.7.3 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.

4.7.4 General Rules

(1) There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.

(2) To define an item as alphabetic:

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

b. 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.

(3) To define an item as numeric:

a. Its 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; and

b. If unsigned, its contents when represented in standard data format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6',

'7', '8', and '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign. (See page II-31, The SIGN Clause.)

(4) To define an item as alphanumeric:

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

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

(5) To define an item as alphanumeric edited:

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

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

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

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

(6) To define an item as numeric edited:

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

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

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

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

(7) 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'.

(8) 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 page II-31, The SIGN Clause.)

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 leading 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.

4.7.5 Editing Rules

(1) 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:

a. Simple insertion
b. Special insertion
c. Fixed insertion
d. Floating insertion

There are two types of suppression and replacement editing:

a. Zero suppression and replacement with spaces
b. Zero suppression and replacement with asterisks

(2) The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given category:

CATEGORY	TYPE OF EDITING
Alphabetic	Simple insertion 'B' only
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion '0', 'B' and '/'
Numeric Edited	All, subject to rules in rule 3 below

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

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

(5) 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.

(6) Fixed Insertion Editing. The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item. The symbol '+' or '-', when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a '+' or a '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character-string. Editing sign control symbols produce the following results depending upon the value of the data item:

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

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

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters 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.

(8) Zero Suppression Editing. The suppression of leading zeroes 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.

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

4.7.6 Precedence Rules

The chart on page II-25 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.

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
	{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

Non-floating insertion symbols '+' and '-', floating insertion symbols 'Z', '*', '+', '-', and 'cs', and other symbol 'P' appear twice in the PICTURE character precedence chart on page II-25. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the chart represents its use to the right of the decimal point position.

4.8 THE REDEFINES CLAUSE

4.8.1 Function

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

4.8.2 General Format

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

NOTE: Level-number, 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 REDEFINES clause.

4.8.3 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 page IV-12, The DATA RECORDS Clause, General Rule 2.)

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

(5) The data description entry for data-name-2 cannot contain a REDEFINES clause. In Level 1, data-name-2 cannot be subordinate to an entry which contains a REDEFINES clause. In Level 2, data-name-2 may be subordinate to an entry which contains a REDEFINES clause. The data description entry for data-name-2 cannot contain an OCCURS clause. However, data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause 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 page III-2, The OCCURS Clause.)

(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.

4.8.4 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.

4.9 THE RENAMES CLAUSE

4.9.1 Function

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

4.9.2 General Format

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

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.

4.9.3 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 level 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 page III-2, The OCCURS Clause.)

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

(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 (see page III-2).

4.9.4 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.

4.10 THE SIGN CLAUSE

4.10.1 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.

4.10.2 General Format

$$\left[\underline{\text{SIGN}}\ \text{IS}\right]\ \left\{\begin{matrix}\underline{\text{LEADING}}\\ \underline{\text{TRAILING}}\end{matrix}\right\}\ \left[\underline{\text{SEPARATE}}\ \text{CHARACTER}\right]$$

4.10.3 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.

4.10.4 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, the implementor will define the position and representation of the operational sign. 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).

c. The implementor defines what constitutes valid sign(s) for data items.

(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.

4.11 THE SYNCHRONIZED CLAUSE

4.11.1 Function

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory (see page I-87, Item Alignment for Increased Object-Code Efficiency).

4.11.2 General Format

$$\left\{\begin{matrix}\underline{\text{SYNCHRONIZED}}\\ \underline{\text{SYNC}}\end{matrix}\right\} \left[\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right]$$

4.11.3 Syntax Rules

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

(2) SYNC is an abbreviation for SYNCHRONIZED.

4.11.4 General Rules

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

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

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

(2) 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. The specific positioning is, however, determined by the implementor.

(3) 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.

(4) 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.

(5) 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.

(6) 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.

(7) 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. (See general rule 8b.)

(8) This clause is hardware dependent and in addition to rules 1 through 7, the implementor must specify how elementary items associated with this clause are handled regarding:

a. The format on the external media of records or groups containing elementary items whose data description contains the SYNCHRONIZED clause.

b. Any necessary generation of implicit FILLER, if the elementary item immediately preceding an item containing the SYNCHRONIZED clause does not terminate at an appropriate natural boundary. Such automatically generated FILLER positions are included in:

1) The size of any group to which the FILLER item belongs; and

2) The number of character positions allocated when the group item of which the FILLER item is a part appears as the object of a REDEFINES clause.

(9) An implementor may, at his option, specify automatic alignment for any internal data formats except, within a record, data items whose usage is DISPLAY. However, the record itself may be synchronized.

(10) Any rules for synchronization of the records of a data file, as this effects the synchronization of elementary items, will be specified by the implementor.

4.12 THE USAGE CLAUSE

4.12.1 Function

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

4.12.2 General Format

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

4.12.3 Syntax Rules

(1) The PICTURE character-string of a COMPUTATIONAL item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', one or more 'P's. (See page II-18, The PICTURE Clause.)

(2) COMP is an abbreviation for COMPUTATIONAL.

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

(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.

4.13 THE VALUE CLAUSE

4.13.1 Function

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

4.13.2 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   }
```

4.13.3 Syntax Rules

(1) The words THRU and THROUGH are equivalent.

(2) The VALUE clause cannot be stated for any items whose size is variable. (See page III-2, The OCCURS Clause.)

(3) A signed numeric literal must have associated with it a signed numeric PICTURE character-string.

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

4.13.4 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 page I-86, Standard Alignment Rules.)

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 page I-86, Standard Alignment Rules.) Editing characters in the PICTURE clause are included in determining the size of the data item (see page II-18, The PICTURE Clause) 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.

4.13.5 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. (See page I-91, Condition-Name.) Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

4.13.6 Data Description Entries Other Than Condition-Names

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

a. In Level 1, the VALUE clause cannot be used in the File Section. In the File Section, the VALUE clause may be used only in condition-name entries.

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

c. In Level 1, the VALUE clause cannot be used in the Linkage Section. In the Linkage Section, the VALUE clause may be used only in condition-name entries.

d. In the Report Section, if the elementary report entry containing the VALUE clause does not contain a GROUP INDICATE clause, then the printable item will assume the specified value each time its report group is printed. However, when the GROUP INDICATE clause is also present, the specified value will be presented only when certain object time conditions exist. (See page VIII-31, The GROUP INDICATE Clause.)

(2) The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries. (See page III-2, The OCCURS Clause.)

(3) The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

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

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

5. PROCEDURE DIVISION IN THE NUCLEUS

5.1 ARITHMETIC EXPRESSIONS

5.1.1 Definition of an Arithmetic Expression

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. 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 1, Combination of Symbols in Arithmetic Expressions, on page II-40.

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

5.1.2 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 must be preceded by a space and followed by a space.

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

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

5.1.3 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 1, 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 1. 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 parentheses 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 on page II-55. Each implementor will indicate the techniques used in handling arithmetic expressions.

5.2 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.

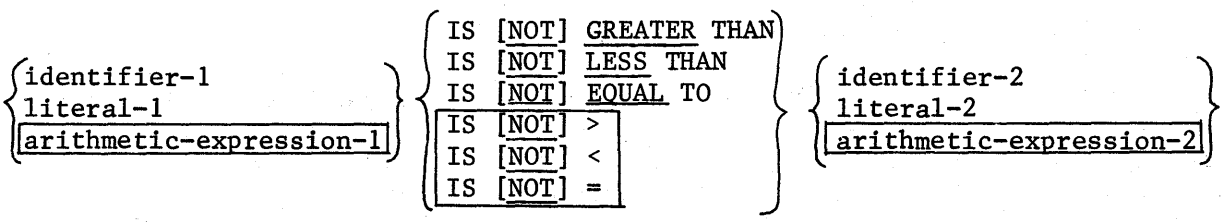
5.2.1 Simple Conditions

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.

5.2.1.1 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:



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

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

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word 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 follows:

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] =

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

5.2.1.1.1 Comparison of Numeric Operands

For operands whose class is numeric (see page I-85, paragraph 5.3.3.3), 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.

5.2.1.1.2 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 page II-6, The OBJECT-COMPUTER Paragraph). If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

a. 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 page II-74, The MOVE Statement, and page II-20, the PICTURE character 'P'.)

b. If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item were then compared to the nonnumeric operand. (See page II-74, The MOVE Statement, and page II-20, the PICTURE character 'P'.)

c. 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: operands of equal size and operands of unequal size.

(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.

5.2.1.2 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 implementor defines what constitutes valid sign(s) for data items not described with the SIGN IS SEPARATE clause.

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

5.2.1.3 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.

5.2.1.4 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.

5.2.1.5 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 the 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.

5.2.2 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.

5.2.2.1 Negated Simple Conditions

A simple condition (see page II-41) 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

5.2.2.2 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 $\left\{\left\{\begin{matrix}\underline{\text{AND}} \\ \underline{\text{OR}}\end{matrix}\right\}\ \text{condition}\right\}$ ...

Where 'condition' may be:

(1) A simple condition, or

(2) A negated simple condition, or

(3) A combined condition, or

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

(5) Combinations of the above, specified according to the rules summarized in table 2, 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 2, Combinations of Conditions, Logical Operators, and Parentheses, on the next page and paragraph 5.2.4, Condition Evaluation Rules, on page II-48.)

Table 2 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 to the left of its corresponding right parenthesis.

Given the following element	Location in conditional expression		In a left-to-right sequence of elements:	
	First	Last	Element, when not first, may be immediately preceded by only:	Element, when not last, may be immediately followed by only:
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	simple-condition, (
(	Yes	No	OR, NOT, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)

Table 2. Combinations of Conditions, Logical Operators, and Parentheses

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.

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

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 2, Combinations of Conditions, Logical Operators, and Parentheses, shown above. 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))))

5.2.4 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 on page II-39.)

(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 simple conditions are established.

(4) Truth values for combined conditions are established:

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

(5) Truth values for negated combined conditions are established.

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

5.3 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.

In the discussion below, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

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

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

5.3.2 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 of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR phrase is specified.

(1) If the SIZE ERROR phrase is not specified and a size error condition occurs, the 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.

(2) If the SIZE ERROR phrase is specified and 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.

5.3.3 The CORRESPONDING Phrase

For the purpose of this discussion, 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 REDEFINES or OCCURS clauses. (See page III-2, The OCCURS Clause.)

5.3.4 The Arithmetic Statements

The arithmetic statements are the ADD, [COMPUTE,] DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common features.

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

(2) The maximum size of each operand is eighteen (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 page II-55, The ADD Statement; page II-61, The DIVIDE Statement; page II-77, The MULTIPLY Statement; and page II-89, The SUBTRACT Statement) must not contain more than eighteen decimal digits.

5.3.5 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.

5.3.6 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.

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 implementor.

5.3.7 Incompatible Data

Except for the class condition (see page II-43, The Class Condition), 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.

5.4 THE ACCEPT STATEMENT

5.4.1 Function

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

5.4.2 General Format

Format 1

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

Format 2

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

5.4.3 Syntax Rules

(1) The mnemonic-name in Format 1 must also be specified in the SPECIAL-NAMES paragraph of the Environment Division and must be associated with a hardware device.

5.4.4 General Rules

FORMAT 1

(1) The ACCEPT statement causes the transfer of data from the hardware device. This data replaces the contents of the data item named by the identifier.

(2) The implementor will define, for each hardware device, the size of a data transfer.

(3) If a hardware device is capable of transferring data of the same size as the receiving data item, the transferred data is stored in the receiving data item.

(4) If a hardware device is not capable of transferring data of the same size as the receiving data item, then:

a. If the size of the receiving data item [(or of the portion of the receiving data item not yet currently occupied by transferred data)] exceeds the size of the transferred data, the transferred data is stored aligned to the left in the receiving data item [(or the portion of the receiving data item not yet occupied, and additional data is requested.] In Level 1, only one transfer of data is provided.

b. If the size of the transferred data exceeds the size of the receiving data item (or of the portion of the receiving data item not yet occupied by transferred data), only the leftmost characters of the transferred data are stored in the receiving data item (or in the portion remaining). The remaining characters of the transferred data which do not fit into the receiving data item are ignored.

(5) If the FROM phrase is not given, the device that the implementor specifies as standard is used.

FORMAT 2

(6) 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.

(7) 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 order to low order (left to right), year of century, month of year, and day of month. Therefore, July 1, 1968 would be expressed as 680701. 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.

(8) 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 if it had been described in a COBOL program as an unsigned elementary numeric integer data item five digits in length.

(9) 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.

5.5 THE ADD STATEMENT

5.5.1 Function

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

5.5.2 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]
```

5.5.3 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 page II-51, The Arithmetic Statements).

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

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

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

(4) CORR is an abbreviation for CORRESPONDING.

5.5.4 General Rules

(1) See page II-50, The ROUNDED Phrase; page II-50, The SIZE ERROR Phrase; page II-51, The CORRESPONDING Phrase; page II-51, The Arithmetic Statements; page II-51, Overlapping Operands; and page II-51, Multiple Results in Arithmetic Statements.

(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 values 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 insures that enough places are carried so as not to lose any significant digits during execution.

5.6 THE ALTER STATEMENT

5.6.1 Function

The ALTER statement modifies a predetermined sequence of operations.

5.6.2 General Format

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

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

5.6.3 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.

5.6.4 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 page IX-2, Independent Segments).

(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. (See Section IX, Segmentation.)

5.7 THE COMPUTE STATEMENT

5.7.1 Function

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

5.7.2 General Format

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

5.7.3 Syntax Rules

(1) Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric edited item.

5.7.4 General Rules

(1) See page II-50, The ROUNDED Phrase, page II-50, The SIZE ERROR Phrase; page II-51, The Arithmetic Statements; page II-51, Overlapping Operands; and page II-51, 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. (See page II-39, Arithmetic Expressions.)

(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.

Thus, each implementor will indicate the techniques used in handling arithmetic expressions.

5.8 THE DISPLAY STATEMENT

5.8.1 Function

The DISPLAY statement causes low volume data to be transferred to an appropriate hardware device.

5.8.2 General Format

```
DISPLAY {identifier-1} [, identifier-2] ... [UPON mnemonic-name]
        {literal-1   }  [, literal-2   ]
```

5.8.3 Syntax Rules

(1) The mnemonic-name is associated with a hardware device in the SPECIAL-NAMES paragraph in the Environment Division.

(2) Each literal may be any figurative constant, except ALL.

(3) If the literal is numeric, then it must be an unsigned integer.

5.8.4 General Rules

(1) The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed.

(2) The implementor will define, for each hardware device, the size of a data transfer.

(3) If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

(4) If the hardware device is capable of receiving data of the same size as the data item being transferred, then the data item is transferred.

(5) If the hardware device is not capable of receiving data of the same size as the data item being transferred, then one of the following applies:

a. If the size of the data item being transferred exceeds the size of the data that the hardware device is capable of receiving in a single transfer, the data beginning with the leftmost character is stored aligned to the left in the receiving hardware device and additional data is requested. In Level 1, only one transfer of data is provided.

b. If the size of the data item that the hardware device is capable of receiving exceeds the size of the data being transferred, the transferred data is stored aligned to the left in the receiving hardware device.

(6) 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.

(7) If the UPON phrase is not used, the implementor's standard display device is used.

5.9 THE DIVIDE STATEMENT

5.9.1 Function

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

5.9.2 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} GIVING identifier-3 [ROUNDED]
       {literal-1   }      {literal-2   }

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

Format 3

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

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

Format 4

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

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

Format 5

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

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

5.9.3 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.

6.9.4 General Rules

(1) See page II-50, The ROUNDED Phrase; page II-50, The SIZE ERROR Phrase; page II-51, The Arithmetic Statements; page II-51, Overlapping Operands; and page II-51, Multiple Results in Arithmetic Statements; for a description of these functions. See also general rules 5 through 7 below for a discussion of the ROUNDED phrase and the SIZE ERROR phrase as they pertain to Formats 4 and 5.

(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 identifier-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 alignment and 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.

5.10 THE ENTER STATEMENT

5.10.1 Function

The ENTER statement provides a means of allowing the use of more than one language in the same program.

5.10.2 General Format

ENTER language-name [routine-name] .

5.10.3 Syntax Rules

(1) The language-name may refer to any programming language which the implementor specifies may be entered through COBOL. Language-name is specified by the implementor.

(2) A routine-name is a COBOL word and it may be referred to only in an ENTER sentence.

(3) The sentence ENTER COBOL must follow the last other-language statement in order to indicate to the compiler where a return to COBOL source language takes place.

5.10.4 General Rules

(1) The other language statements are executed in the object program as if they had been compiled into the object program following the ENTER statement.

(2) Implementors will specify, for their compilers, all details on how the other language(s) are to be written.

(3) If the statements in the entered language cannot be written in-line, a routine-name is given to identify the portion of the other language coding to be executed at this point in the procedure sequence. If the other language statements can be written in-line, routine-name is not used.

5.11 THE EXIT STATEMENT

5.11.1 Function

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

5.11.2 General Format

EXIT.

5.11.3 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.

5.11.4 General Rules

(1) 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.

5.12 THE GO TO STATEMENT

5.12.1 Function

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

5.12.2 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

5.12.3 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 must appear as the last statement in that sequence.

5.12.4 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.

5.13 THE IF STATEMENT

5.13.1 Function

The IF statement causes a condition (see page II-41, Conditional Expressions) to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

5.13.2 General Format

$$\underline{\text{IF}}\ \text{condition;}\ \left\{\begin{matrix}\text{statement-1}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}\ \left\{\begin{matrix};\ \underline{\text{ELSE}}\ \text{statement-2}\\ ;\ \underline{\text{ELSE}}\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}$$

5.13.3 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.

5.13.4 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. (See page I-103, Categories of Statements.) 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. (See page I-103, Categories of Statements.) 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.

5.14 THE INSPECT STATEMENT

5.14.1 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.

5.14.2 General Format

Format 1

```
INSPECT identifier-1 TALLYING

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

Format 2

```
INSPECT identifier-1 REPLACING

{CHARACTERS BY {identifier-6|literal-4} [{BEFORE|AFTER} INITIAL {identifier-7|literal-5}]
 {, {ALL|LEADING|FIRST} {, {identifier-5|literal-3} BY {identifier-6|literal-4} [{BEFORE|AFTER} INITIAL {identifier-7|literal-5}]} ...} ...}
```

Format 3

```
INSPECT identifier-1 TALLYING

{, identifier-2 FOR {, {{ALL|LEADING} {identifier-3|literal-1}} [{BEFORE|AFTER} INITIAL {identifier-4|literal-2}]} ...} ...
 {, {CHARACTERS} {identifier-3|literal-1}} [{BEFORE|AFTER} INITIAL {identifier-4|literal-2}]} ...} ...

  REPLACING

{CHARACTERS BY {identifier-6|literal-4} [{BEFORE|AFTER} INITIAL {identifier-7|literal-5}]
 {, {ALL|LEADING|FIRST} {, {identifier-5|literal-3} BY {identifier-6|literal-4} [{BEFORE|AFTER} INITIAL {identifier-7|literal-5}]} ...} ...}
```

5.14.3 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) In Level 1, 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 must be one character in length. Except as specifically noted in syntax and general rules, this restriction on length does not apply to Level 2.

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.

5.14.4 General Rules

(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 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

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

c. If any of the identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in general rule 2b had been applied. (See page II-74, The MOVE Statement.)

(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).

(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 is 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 is incremented by one (1) 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 is incremented by one (1) 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 is incremented by one (1) 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 adjective 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.

5.14.5 Examples

Following are six examples of the INSPECT statement:

```
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
```

5.15 THE MOVE STATEMENT

5.15.1 Function

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

5.15.2 General Format

Format 1

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

Format 2

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

5.15.3 Syntax Rules

(1) Identifier-1 and literal represent the sending area; identifier-2, identifier-3, ..., represent the receiving area.

(2) CORR is an abbreviation for CORRESPONDING.

(3) When the CORRESPONDING phrase is used, both identifiers must be group items.

(4) An index data item cannot appear as an operand of a MOVE statement. (See page III-5, The USAGE Clause.)

5.15.4 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 paragraph 5.3.3, The CORRESPONDING Phrase, on page II-51. 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, a numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

d. All other elementary moves are legal and are performed according to the 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 on page I-86. 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 occupied a separate character position (see page II-31, The SIGN Clause), 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 on page I-86, except where zeroes are replaced because of editing requirements.

1. When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. (See page II-31, The SIGN Clause). 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.

2. 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.

3. 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 on page I-86. 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 (see page III-4).

(6) Data in the following chart 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.

CATEGORY OF SENDING DATA ITEM		CATEGORY OF RECEIVING DATA ITEM		
		ALPHABETIC	ALPHANUMERIC EDITED ALPHANUMERIC	NUMERIC INTEGER NUMERIC NON-INTEGER NUMERIC EDITED
ALPHABETIC		Yes/4c	Yes/4a	No/3a
ALPHANUMERIC		Yes/4c	Yes/4a	Yes/4b
ALPHANUMERIC EDITED		Yes/4c	Yes/4a	No/3a
NUMERIC	INTEGER	No/3b	Yes/4a	Yes/4b
	NON-INTEGER	No/3b	No/3c	Yes/4b
NUMERIC EDITED		No/3b	Yes/4a	No/3a

5.16 THE MULTIPLY STATEMENT

5.16.1 Function

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

5.16.2 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]
```

5.16.3 Syntax Rules

(1) Each identifier must refer to a numeric elementary item, except that in Format 2 each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

(2) Each literal must be a numeric literal.

(3) The composite of operands, which is that hypothetical data item resulting from the superimposition of all receiving data items of a given statement aligned on their decimal points, must not contain more than eighteen (18) digits.

5.16.4 General Rules

(1) See page II-50, The ROUNDED Phrase; page II-50, The SIZE ERROR Phrase; page II-51, The Arithmetic Statements; page II-51, Overlapping Operands; and page II-51, Multiple Results in Arithmetic Statements.

(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.

5.17 THE PERFORM STATEMENT

5.17.1 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.

5.17.2 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    }
```

5.17.3 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 on page II-41, Conditional Expressions.

(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.

5.17.4 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 6b, 6c, and 6d). 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 by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

(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.

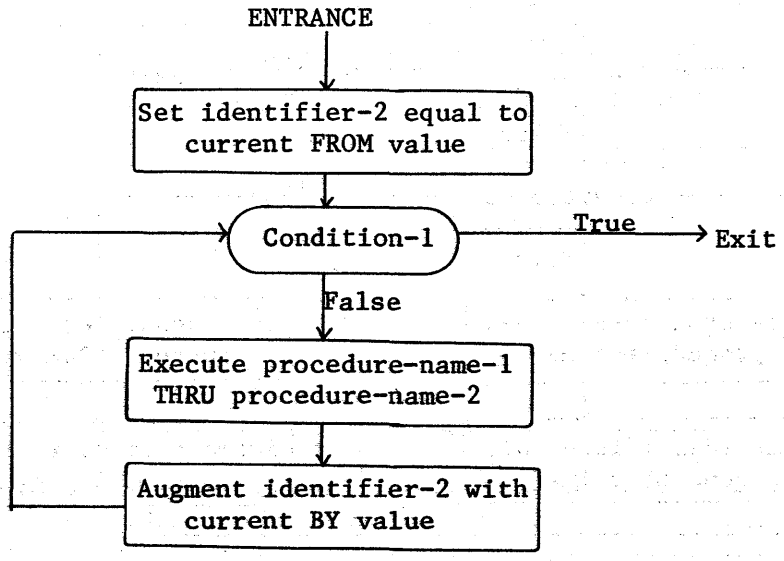
b. Format 2 is the PERFORM...TIMES. The procedures are performed the number of times specified by integer-1 or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 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, identifier, when it appears in an associated VARYING or AFTER phrase, is initialized according to the rules of the SET statement; subsequent augmentation is as described below.

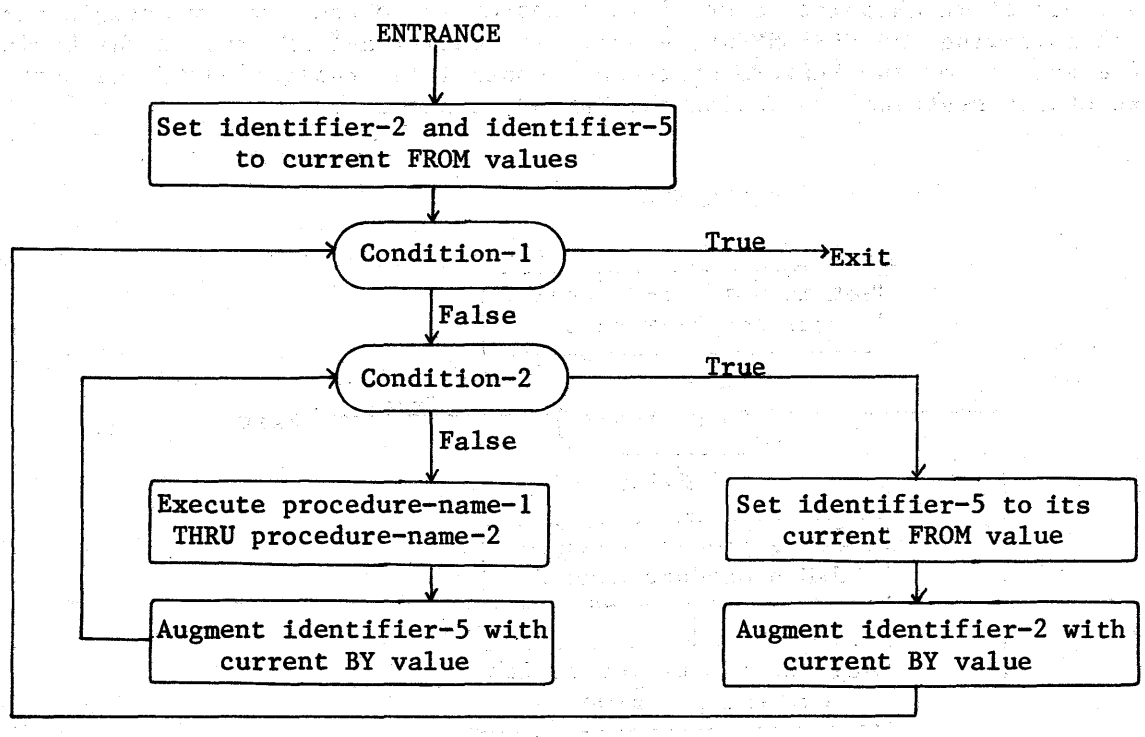
In Format 4, when one identifier is varied, identifier-2 is set to the value of literal-1 or the current value of identifier-3 at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the sequence of procedures, procedure-name-1 through procedure-name-2, 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.



Flowchart for the VARYING Phrase of a PERFORM Statement Having One Condition

In Format 4, when two identifiers are varied, identifier-2 and identifier-5 are set to the current value of identifier-3 and identifier-6, respectively. After the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the 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.

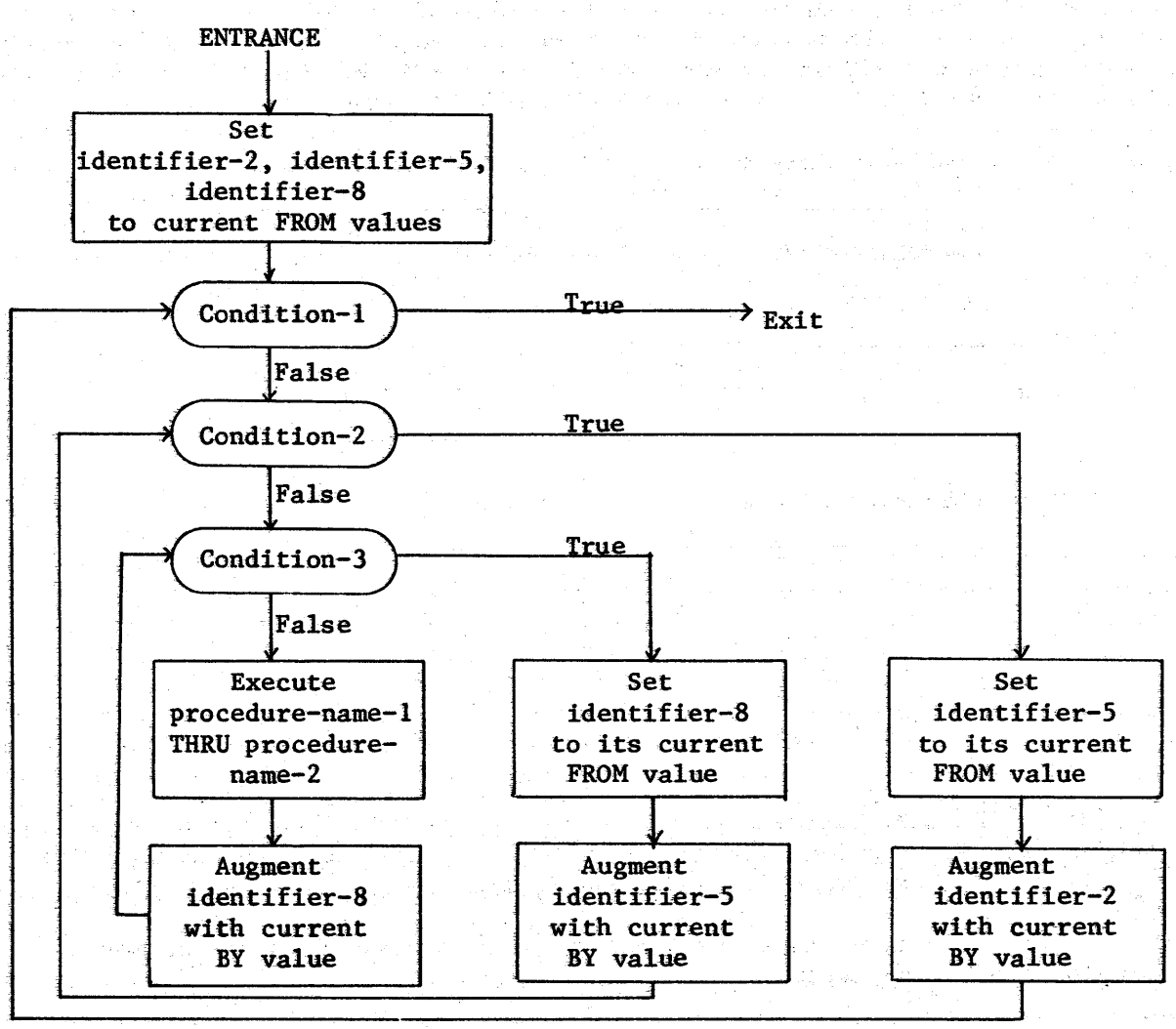


Flowchart for the VARYING Phrase of a PERFORM Statement Having Two Conditions

At the termination of the PERFORM statement identifier-5 contains the current value of identifier-6. Identifier-2 has a value that exceeds the last used setting by an increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-2 contains the current value of identifier-3.

When two identifiers are varied, identifier-5 goes through a complete cycle (FROM, BY, UNTIL) each time identifier-2 is varied.

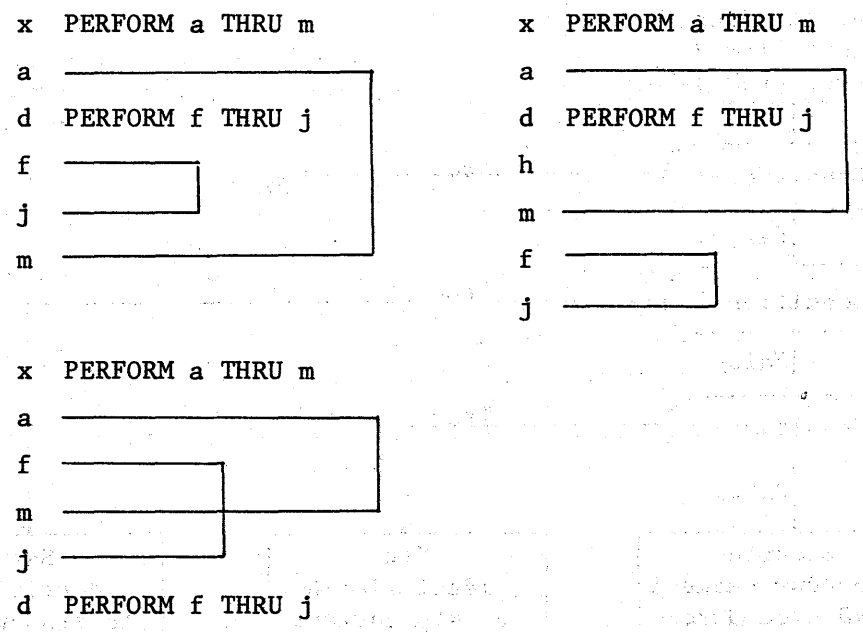
For three identifiers the mechanism is the same as for two identifiers except that identifier-8 goes through a complete cycle each time that identifier-5 is augmented by identifier-7 or literal-4, which in turn goes through a complete cycle each time identifier-2 is varied.



Flowchart for the VARYING Phrase of a PERFORM Statement Having Three Conditions.

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

(7) If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See the illustration below.



(8) A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

a. Sections and/or paragraphs wholly contained in one or more 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.

5.18 THE STOP STATEMENT

5.18.1 Function

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

5.18.2 General Format

$$\underline{\text{STOP}} \left\{ \begin{matrix} \underline{\text{RUN}} \\ \text{literal} \end{matrix} \right\}$$

5.18.3 Syntax Rules

(1) The literal may be numeric or nonnumeric 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.

5.18.4 General Rules

(1) If the RUN phrase is used, then the ending procedure established by the installation and/or the compiler 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.

5.19 THE STRING STATEMENT

5.19.1 Function

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

5.19.2 General Format

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

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

    INTO identifier-7 [ WITH POINTER identifier-8 ]

    [; ON OVERFLOW imperative-statement]
```

5.19.3 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-7 must represent an elementary alphanumeric data item without editing symbols or the JUSTIFIED clause.

(4) 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.

(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.

5.19.4 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 whose usage is DISPLAY.

(5) When the STRING statement is executed, the transfer of data is governed by the following rules:

a. Those characters from literal-1, 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 page II-74, 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, and he is 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.

5.20 THE SUBTRACT STATEMENT

5.20.1 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.

5.20.2 General Format

Format 1

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

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

Format 2

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

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

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

Format 3

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

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

5.20.3 Syntax Rules

(1) Each identifier must refer to a numeric elementary item except that:

a. In Format 2, each identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

b. In Format 3, where 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 page II-51, The Arithmetic Statements.)

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

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

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

(4) CORR is an abbreviation for CORRESPONDING.

5.20.4 General Rules

(1) See page II-50, The ROUNDED Phrase; page II-50, The SIZE ERROR Phrase; page II-51, The CORRESPONDING Phrase; page II-51, The Arithmetic Statement; page II-51, Overlapping Operands; and page II-51, Multiple Results in Arithmetic Statements.

(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 insures enough places are carried so as not to lose significant digits during execution.

5.21 THE UNSTRING STATEMENT

5.21.1 Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

5.21.2 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]
```

5.21.3 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 character-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.

5.21.4 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) Identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 isolated by the delimiters for the move to identifier-4. This value does not include a count of the delimiter 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 page II-74, 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 page II-74, 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-6 according to the rules for an elementary move.

f. If the DELIMITED BY phrase is specified, the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.

g. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The behavior 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 completed, 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.

1. INTRODUCTION TO THE TABLE HANDLING MODULE

1.1 FUNCTION

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 facility is 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 page I-89).

1.2 LEVEL CHARACTERISTICS

Table Handling Level 1 provides a capability for accessing items in up to three-dimensional fixed length tables. This level also provides series options and the ability to vary the contents of indices by an increment or decrement.

Table Handling Level 2 provides a capability for accessing items in up to three-dimensional variable length tables. This level also provides the additional facilities for specifying ascending or descending keys and permits searching a dimension of a table for an item satisfying a specified condition.

2. DATA DIVISION IN THE TABLE HANDLING MODULE

2.1 THE OCCURS CLAUSE

2.1.1 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.

2.1.2 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 ] ...]
```

2.1.3 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 2 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 a 01, 66, 77, or an 88 level-number.

b. Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.

(9) In Format 2, the data item defined by data-name-1 must not occupy a character position within the range of the first character position defined by the data description entry containing the OCCURS clause and the last character position defined by the record description entry containing that OCCURS clause.

(10) If data-name-2 is not the subject of this entry, then:

a. All 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.

2.1.4 General Rules

(1) The OCCURS clause is used in defining tables and other homogeneous 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 page I-89, Subscripting; page I-89, Indexing; page I-90, Identifier.)

(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 on page II-38.)

(3) The number of occurrences of the subject entry is defined as follows:

a. In Format 1, the value of integer-2 represents the exact number of occurrences.

b. In Format 2, the current value of the data item referenced by data-name-1 represents the number of occurrences.

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 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 in data-name-2, data-name-3, etc. The ascending or descending order is determined according to the rules for comparison of operands (see page II-42, Comparison of Numeric Operands, and page II-42, Comparison of Nonnumeric Operands). The data-names are listed in their descending order of significance.

2.2 THE USAGE CLAUSE

2.2.1 Function

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

2.2.2 General Format

```
[USAGE IS]  INDEX
```

2.2.3 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.

2.2.4 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 method of representation and the actual value assigned are determined by the implementor. 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.

(4) The external and internal format of an index data item is specified by the implementor.

3. PROCEDURE DIVISION IN THE TABLE HANDLING MODULE

3.1 RELATION CONDITION

3.1.1 Comparisons Involving Index-Names and/or Index Data Items

Relation tests may be made between:

(1) Two index-names. The result is the same as if the corresponding occurrence numbers were compared.

(2) An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

(3) An index data item and an index-name or another index data item. The actual values are compared without conversion.

(4) The result of the comparison of an index data item with any data item or literal not specified above is undefined.

3.2 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.

3.3 THE SEARCH STATEMENT

3.3.1 Function

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

3.3.2 General Format

Format 1

```
SEARCH identifier-1 [ VARYING { identifier-2 } ]
                              { index-name-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 ]
     ; WHEN { data-name-1 { IS EQUAL TO } { identifier-3                } }
            {             { IS =       } { literal-1                   } }
            { condition-name-1           { arithmetic-expression-1     } }
        [ AND { data-name-2 { IS EQUAL TO } { identifier-4                } } ]...
              {             { IS =       } { literal-2                   } }
              { condition-name-2           { arithmetic-expression-2     } }
       { imperative-statement-2 }
       { NEXT SENTENCE          }
```

NOTE: The required relational character '=' is not underlined to avoid confusion with other symbols.

3.3.3 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 Conditional Expressions, page II-41.

(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.

3.3.4 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 page III-2, The OCCURS Clause.) Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next executable sentence.

b. 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; see page III-2, 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 are satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element outside the permissible range of occurrence values, in which case the search terminates as indicated in 1a 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 nonserial 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 in a manner specified by the implementor, 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. (See page III-2, 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 the 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:

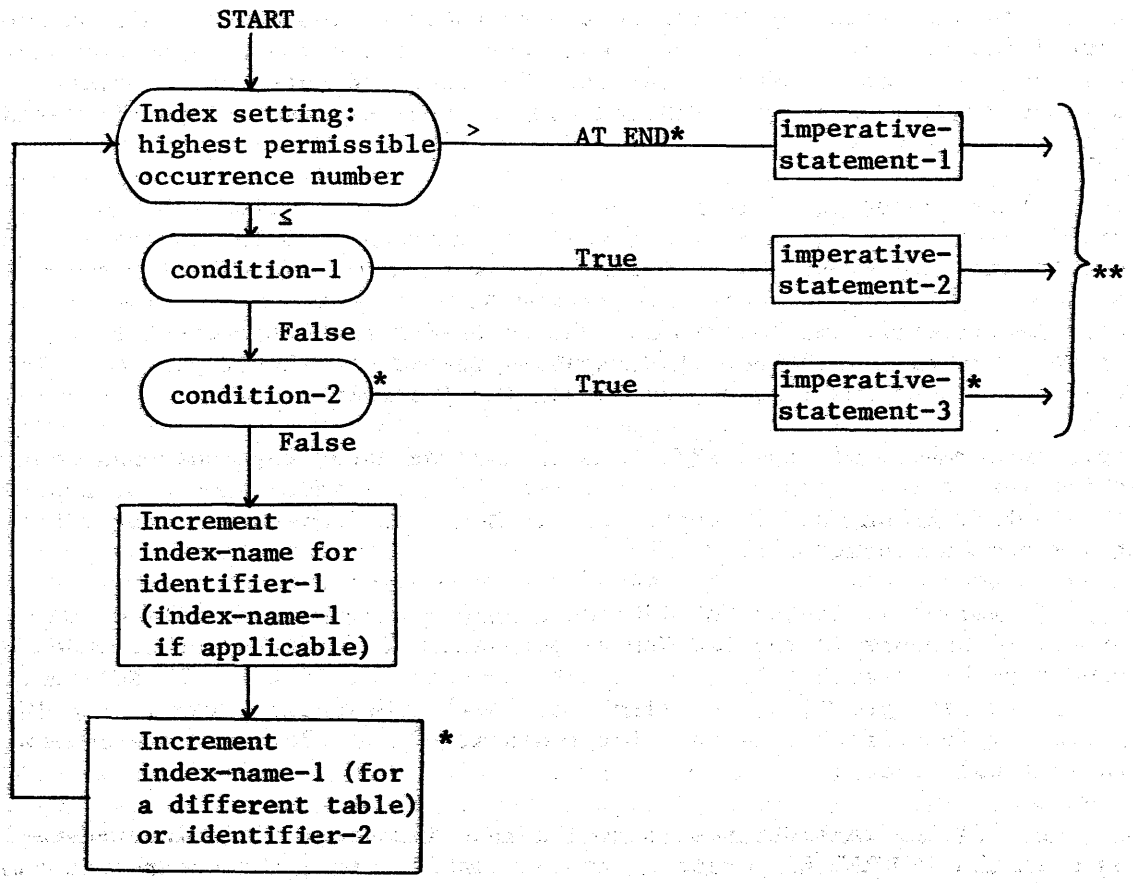
a. If the VARYING index-name-1 phrase is used, and if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

b. If the VARYING identifier-2 phrase is specified, and identifier-2 is an index data item, then the data item referenced by identifier-2 is incremented by the same amount as, and at the same time as, the index associated

with identifier-1 is incremented. If identifier-2 is not an index data item, the data item referenced by identifier-2 is incremented by the value one (1) at the same time as the index referenced by the index-name associated with identifier-1 is incremented.

(8) If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two or three dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire two or three dimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

A flowchart of the Format 1 SEARCH operation containing two WHEN phrases follows:



*These operations are options included only when specified in the SEARCH statement.

**Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

3.4 THE SET STATEMENT

3.4.1 Function

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

3.4.2 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    }
```

3.4.3 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.

3.4.4 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 page III-7, The SEARCH Statement and page II-78, The PERFORM Statement.)

(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 data 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 the following chart represents the validity of various operand combinations in the SET statement. The general rule reference indicates the applicable general rule.

Sending Item	Receiving Item		
	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 *
Index Data Item	No/3c	Valid/3a *	Valid/3b *

*No conversion takes place

1. INTRODUCTION TO THE SEQUENTIAL I-O MODULE

1.1 FUNCTION

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 rerun points and the sharing of memory areas among files.

1.2 LEVEL CHARACTERISTICS

Sequential I-O Level 1 does not provide full COBOL facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, the Sequential I-O Level 1 provides limited capabilities for the CLOSE, OPEN, USE, and WRITE statements and full capabilities for the READ and REWRITE statements, as specified in the formats of this module.

Sequential I-O Level 2 provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, the Sequential I-O Level 2 provides full capabilities for the CLOSE, OPEN, READ, REWRITE, USE, and WRITE statements as specified in the formats of this module. The additional features available in Level 2 include: OPTIONAL files, the RESERVE clause, SAME RECORD AREA, MULTIPLE FILE tapes, REVERSED, EXTEND, and additional flexibility through series options.

1.3 LANGUAGE CONCEPTS

1.3.1 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.

1.3.2 Access Mode

In the sequential access mode, the sequence in which records are accessed is the order in which the records were originally written.

1.3.3 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.

1.3.4 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.

1.3.4.1 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 Implementor Defined

The meaning 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 - Implementor Defined. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the implementor. 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.

1.3.4.2 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:

1. If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

2. When status key 1 contains a value of '3' indicating a permanent error condition, status key 2 may contain a value of '4' indicating a boundary violation. This condition indicates that an attempt has been made to write beyond the externally defined boundaries of a sequential file. The implementor specifies the manner in which these boundaries are defined.

3. When status key 1 contains a value of '9' indicating an implementor-defined condition, the value of status key 2 is defined by the implementor.

1.3.4.3 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 figure. An 'X' at an intersection indicates a valid permissible combination.

Status Key 1	Status Key 2	
	No Further Information (0)	Boundary Violation (4)
Successful Completion (0)	X	
At End (1)	X	
Permanent Error (3)	X	X
Implementor Defined (9)		

1.3.5 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 page IV-28, The READ Statement.

1.3.6 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 the size of integer-1 or the data item referenced by data-name-1 in the LINAGE clause. See page IV-15, The LINAGE Clause, for the rules governing the LINAGE-COUNTER.

2. ENVIRONMENT DIVISION IN THE SEQUENTIAL I-O MODULE

2.1 INPUT-OUTPUT SECTION

2.1.1 The FILE-CONTROL Paragraph

2.1.1.1 Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

2.1.1.2 General Format

```
FILE-CONTROL. {file-control-entry} ...
```

2.1.2 The File Control Entry

2.1.2.1 Function

The file control entry names a file and may specify other file-related information.

2.1.2.2 General Format

```
SELECT [ OPTIONAL ] file-name

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

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

  [ ; ORGANIZATION IS SEQUENTIAL ]

  [ ; ACCESS MODE IS SEQUENTIAL ]

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

2.1.2.3 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-1 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section, the Report Section, or the Communication Section.

(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.

2.1.2.4 General Rules

(1) The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium.

(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. If the RESERVE clause is not specified the number of input-output areas allocated is specified by the implementor.

(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) 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.

(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-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 page IV-1, I-O Status.)

2.1.3 The I-O-CONTROL Paragraph

2.1.3.1 Function

The I-O-CONTROL paragraph specifies the points at which rerun 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.

2.1.3.2 General Format

```
I-O-CONTROL.

[ ; RERUN [ ON { file-name-1       } ] EVERY { { [ END OF ] { REEL } }                    OF file-name-2 } ]...
                { implementor-name }        { { UNIT   }      }
                                            {   integer-1 RECORDS }
                                            { integer-2 CLOCK-UNITS }
                                            { 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 ] ] ... ] ...
```

2.1.3.3 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. The definition of UNIT is determined by each implementor.

(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 RECORDS clauses are specified, no two of them may 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) Only one RERUN clause containing the CLOCK-UNITS clause may be specified.

(7) 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.

(8) The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

2.1.3.4 General Rules

(1) The RERUN clause specifies when and where the rerun information is recorded. Rerun information is recorded in the following ways:

a. If file-name-1 is specified, the rerun information is written on each reel or unit of an output file and the implementor specifies where, on the reel or file, the rerun information is to be recorded.

b. If implementor-name is specified, the rerun information is written as a separate file on a device specified by the implementor.

(2) There are seven forms of the RERUN clause, based on the several conditions under which rerun points can be established. The implementor must provide at least one of the specified forms of the RERUN clause.

a. When either the END OF REEL or END OF UNIT clause is used without the ON clause. In this case, the rerun information is written on file-name-2, which must be an output file.

b. When either the END OF REEL or END OF UNIT clause is used and file-name-1 is specified in the ON clause. In this case, the rerun information is written on file-name-1, which must be an output file. In addition, normal reel, or unit, closing functions for file-name-2 are performed. File-name-2 may either be an input or an output file.

c. When either the END OF REEL or END OF UNIT clause is used and implementor-name is specified in the ON clause. In this case, the rerun information is written on a separate rerun unit defined by the implementor. File-name-2 may be either an input or output file.

d. When the integer-1 RECORDS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever integer-1 records of

file-name-2 have been processed. File-name-2 may be either an input or output file with any organization or access.

e. When the integer-2 CLOCK-UNITS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever an interval of time, calculated by an internal clock, has elapsed.

f. When the condition-name clause is used and implementor-name is specified in the ON clause. In this case, the rerun information is written on the device specified by implementor-name whenever a switch assumes a particular status as specified by condition-name. In this case, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

g. When the condition-name clause is used and file-name-1 is specified in the ON clause. In this case, the rerun information is written on file-name-1, which must be an output file, whenever a switch assumed a particular status as specified by condition-name. In this case, as in paragraph f above, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

(3) 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 shared 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 7c on page IV-7.)

(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.

(5) The MULTIPLE FILE clause is required when more than one file shares the same physical reel of tape. Regardless of the number of files on a single reel, only those files that are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION clause need not be given. If any file in the sequence is not listed, the position relative to the beginning of the tape must be given. Not more than one file on the same tape reel may be open at one time.

3. DATA DIVISION IN THE SEQUENTIAL I-O MODULE

3.1 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 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, and the number of lines to be written on a logical printer page. The entry itself is terminated by a period.

3.2 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 on page I-84 while the elements allowed in a record description are shown in the data description skeleton on page II-12.

3.3 THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

3.3.1 Function

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

3.3.2 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 / RECORDS ARE} {STANDARD / OMITTED}

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

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

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

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

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

    [; CODE-SET IS alphabet-name] .
```

3.3.3 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.

(3) One or more record description entries must follow the file description entry.

3.4 THE BLOCK CONTAINS CLAUSE

3.4.1 Function

The BLOCK CONTAINS clause specifies the size of a physical record.

3.4.2 General Format

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

3.4.3 General Rules

(1) This clause is required except when:

a. A physical record contains one and only one complete logical record.

b. The hardware device assigned to the file has one and only one physical record size.

c. The hardware device assigned to the file has more than one physical record size but the implementor has designated one as standard. In this case, the absence of this clause denotes the standard physical record size.

(2) The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exists, in which case the RECORDS phrase must not be used.

a. In mass storage files, where logical records may extend across physical records.

b. The physical record contains padding (area not contained in a logical record).

c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

(3) When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

(4) If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

(5) If logical records of differing size are grouped into one physical record, the technique for determining the size of each logical record is specified by the implementor.

3.5 THE CODE-SET CLAUSE

3.5.1 Function

The CODE-SET clause specifies the character code set used to represent data on the external media.

3.5.2 General Format

CODE-SET IS alphabet-name

3.5.3 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-mass storage files.

3.5.4 General Rules

(1) If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from/to the native character codes. This code conversion occurs during the execution of an input or output operation. (See page II-8, The SPECIAL-NAMES Paragraph.)

(2) If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media.

3.6 THE DATA RECORDS CLAUSE

3.6.1 Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

3.6.2 General Format

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

3.6.3 Syntax Rules

(1) 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.

3.6.4 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 in no way altered by the presence of more than one type of data record within the file.

3.7 THE LABEL RECORDS CLAUSE

3.7.1 Function

The LABEL RECORDS clause specifies whether labels are present.

3.7.2 General Format

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

3.7.3 Syntax Rules

(1) This clause is required in every file description entry.

3.7.4 General Rules

(1) OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

(2) STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the implementor's label specifications.

3.8 THE LINAGE CLAUSE

3.8.1 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.

3.8.2 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   }]
```

3.8.3 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.

3.8.4 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 or 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-name-1, 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 page IV-34, 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:

1) When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one (1).

2) When the ADVANCING identifier-2 or integer phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer or the value of the data item referenced by identifier-2.

3) When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one (1). (See page IV-34, The WRITE Statement.)

4) The value of LINAGE-COUNTER is automatically reset to one (1) when the device is repositioned to the first line that can be written on for each of the succeeding logical pages. (See page IV-34, The WRITE Statement.)

d. The value of LINAGE-COUNTER is automatically set to one (1) 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.

3.9 THE RECORD CONTAINS CLAUSE

3.9.1 Function

The RECORD CONTAINS clause specifies the size of data records.

3.9.2 General Format

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

3.9.3 General Rules

(1) The size of each data record is completely defined within the record description entry, therefore this clause is never required. When present, however, the following notes apply:

a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record; see page I-85, Selection of Character Representation and Radix; page II-33, The SYNCHRONIZED Clause; and page II-35, The USAGE Clause.

3.10 THE VALUE OF CLAUSE

3.10.1 Function

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

3.10.2 General Format

```
VALUE OF implementor-name-1 IS { data-name-1 }
                               { literal-1   }

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

3.10.3 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.

3.10.4 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.

4. PROCEDURE DIVISION IN THE SEQUENTIAL I-O MODULE

4.1 THE CLOSE STATEMENT

4.1.1 Function

The CLOSE statement terminates the processing of reels/units and files with optional rewind and/or lock or removal where applicable.

4.1.2 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     } ]
```

4.1.3 Syntax Rules

(1) The REEL or UNIT phrase must only be used for sequential files.

(2) The files referenced in the CLOSE statement need not all have the same organization or access.

4.1.4 General Rules

Except where otherwise stated in the general rules below, the terms 'reel' and 'unit' are synonymous and completely interchangeable in the CLOSE statement. Treatment of sequential mass storage files is logically equivalent to the treatment of a file on tape or analogous sequential media.

(1) A CLOSE statement may only be executed for a file in an open mode.

(2) For the purpose of showing the effect of various types of CLOSE statements as applied to various storage media, all files are divided into the following categories:

a. Non-reel/unit. A file whose input or output medium is such that the concepts of rewind and reels/units have no meaning.

b. Sequential single-reel/unit. A sequential file that is entirely contained on one reel/unit.

c. Sequential multi-reel/unit. A sequential file that is contained on more than one reel/unit.

(3) The results of executing each type of CLOSE for each category of file are summarized in Table 1, Relationship of Categories of Files and the Formats of the CLOSE Statement.

CLOSE Statement Format	File Category		
	Non-Reel/Unit	Sequential Single-Reel/Unit	Sequential Multi-Reel/Unit
CLOSE	C	C,G	C,G,A
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A
CLOSE WITH NO REWIND	X	C,B	C,B,A
CLOSE REEL/UNIT	X	X	F,G
CLOSE REEL/UNIT FOR REMOVAL	X	X	F,D,G
CLOSE REEL/UNIT WITH NO REWIND	X	X	F,B

Table 1. Relationship of Categories of Files and the Formats of the CLOSE Statement

The definitions of the symbols in the table are given below. Where the definition depends on whether the file is an input, output or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Previous Reels/Units Unaffected

Input Files and Input-Output Files:

All reels/units in the file prior to the current reel/unit are processed according to the implementor's standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement. If the current reel/unit is not the last in the file, the reels/units in the file following the current one are not processed.

Output Files:

All reels/units in the file prior to the current reel/unit are processed according to the implementor's standard reel/unit swap procedure, except those reels/units controlled by a prior CLOSE REEL/UNIT statement.

B. No Rewind of Current Reel

The current reel/unit is left in its current position.

C. Close File

Input Files and Input-Output Files:

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations specified by the implementor are executed. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed. If the file is positioned other than at its end, the closing operations specified by the implementor are executed, but there is no ending label processing.

Output Files:

If label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations specified by the implementor are executed. If label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed.

D. Reel/Unit Removal

An implementor-defined technique is supplied to ensure that the current reel or unit is rewound when applicable, and that the operating system is notified that the reel or unit is logically removed from this run unit; however, the reel or unit may be accessed again, in its proper order of reels or units within the file, if a CLOSE statement without the REEL or UNIT phrase is subsequently executed for this file followed by the execution of an OPEN statement for the file.

E. File Lock

An implementor-defined technique is supplied to ensure that this file cannot be opened again during this execution of this run unit.

F. Close Reel/Unit

Input Files:

The following operations take place:

1. A reel/unit swap.

2. The standard beginning reel/unit label procedure is executed.

The next executed READ statement for that file makes available the next data record on the new reel/unit.

Output Files and Input-Output Files:

The following operations take place:

1. (For output files only.) The standard ending reel/unit label procedure is executed.

2. A reel/unit swap.

3. The standard beginning reel/unit label procedure is executed.

For input-output files, the next executed READ statement that references that file makes the next logical data record on the next mass storage unit available. For output files, the next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

G. Rewind

The current reel or analogous device is positioned at its physical beginning.

X. Illegal

This is an illegal combination of a CLOSE option and a file category. The results at object time are undefined.

(4) The action taken if the file is in the open mode when a STOP RUN statement is executed is specified by the implementor. 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 also specified by the implementor.

(5) If the OPTIONAL phrase has been specified for the file in the FILE-CONTROL paragraph of the Environment Division and the file is not present, the standard end-of-file processing is not performed for that file.

(6) 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.

(7) The WITH NO REWIND and FOR REMOVAL phrases will have no effect at object time if they do not apply to the storage media on which the file resides.

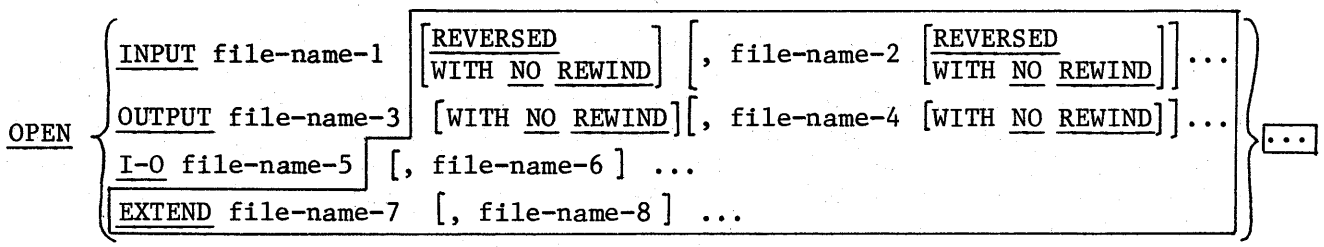
(8) Following the successful execution of a CLOSE statement without the REEL or UNIT phrase, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

4.2 THE OPEN STATEMENT

4.2.1 Function

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

4.2.2 General Format



4.2.3 Syntax Rules

(1) The REVERSED and the NO REWIND phrases can only be used with sequential files. (See The CLOSE Statement on page IV-20.)

(2) The I-O phrase can be used only for mass storage files.

(3) The EXTEND phrase can be used only for sequential files.

(4) The EXTEND phrase must not be specified for multiple file reels. (See The I-O-CONTROL Paragraph on page IV-6.)

(5) The files referenced in the OPEN statement need not all have the same organization or access.

4.2.4 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 statements. In Table 2, Permissible Statements, on page IV-25, '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.

Statement	Open Mode			
	Input	Output	Input-Output	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

Table 2. Permissible Statements

(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, without the REEL, UNIT, [or] [LOCK] phrase, for that file.

(6) Execution of the OPEN statement does not obtain or release the first data record.

(7) If label records are specified for the file, the beginning labels are processed as follows:

a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with the implementor's specified conventions for input label checking.

b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with the implementor's specified conventions for output label writing.

The behavior of the OPEN statement when label records are specified but not present, or when label records are not specified but are present, is undefined.

(8) The file description entry for file-name-1, [file-name-2,] file-name-5, [file-name-6, file-name-7, or 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. (See The READ Statement on page IV-28.)

(10) The REVERSED and NO REWIND phrases can only be used with sequential single reel/unit files. (See The CLOSE Statement on page IV-20.)

(11) The REVERSED and WITH NO REWIND phrases will be ignored if they do not apply to the storage media on which the file resides.

(12) If the storage medium for the file permits rewinding, the following rules apply:

a. When neither the REVERSED, the EXTEND, nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file must be already positioned at its beginning prior to execution of the OPEN statement.

c. When the REVERSED phrase is specified, the file is positioned at its end by execution of the OPEN statement.

(13) When the REVERSED phrase is specified, the subsequent READ statements for the file make the data records of the file available in reversed order; that is, starting with the last record.

(14) 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.

(15) 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.

(16) When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

a. The beginning file labels are processed only in the case of a single reel/unit file.

b. The beginning reel/unit labels on the last existing reel/unit are processed as though the file was being opened with the INPUT phrase.

c. The existing ending file labels are processed as though the file is being opened with the INPUT phrase. These labels are then deleted.

d. Processing then proceeds as though the file had been opened with the OUTPUT phrase.

(17) The I-O phrase permits the opening of a mass storage file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the mass storage file is being initially created.

(18) When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

a. The labels are checked in accordance with the implementor's specified conventions for input-output label checking.

b. The new labels are written in accordance with the implementor's specified conventions for input-output label writing.

(19) 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.

4.3 THE READ STATEMENT

4.3.1 Function

The READ statement makes available the next logical record from a file.

4.3.2 General Format

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

4.3.3 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.

4.3.4 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 on page IV-24.)

(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 page IV-1, I-O Status.)

(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 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 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 execution of a READ statement, and the logical end of the file has not been reached, the following operations are executed:

a. The standard ending reel/unit label procedure.

b. A reel/unit swap.

c. The standard beginning reel/unit label procedure.

d. The first data record of the new reel/unit is made available.

(10) If a file described with the OPTIONAL phrase is not present at the time the file is opened, then at the time of execution of the first READ statement for the file, the AT END condition occurs and the execution of the READ statement is unsuccessful. The standard end-of-file procedures are not performed. (See page IV-4, The FILE-CONTROL Paragraph; page IV-24, The OPEN Statement; page IV-32, The USE Statement; and page IV-1, I-O Status.) Execution of the program then proceeds as specified 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 page IV-1, 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 page IV-1, 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.

4.4 THE REWRITE STATEMENT

4.4.1 Function

The REWRITE statement logically replaces a record existing in a mass storage file.

4.4.2 General Format

REWRITE record-name [FROM identifier]

4.4.3 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.

4.4.4 General Rules

(1) The file associated with record-name must be a mass storage file and must be open in the I-O mode at the time of execution of this statement. (See page IV-24, The OPEN Statement.)

(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 MSCS 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 page IV-1, I-O Status.)

4.5 THE USE STATEMENT

4.5.1 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.

4.5.2 General Format

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

4.5.3 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 referenced in a USE statement need not all have the same organization or access.

4.5.4 General Rules

(1) The designated procedures are executed by the input-output system after completing the standard input-output error routine, or upon recognition of the AT END condition, when the AT END phrase has not been specified in the input-output statement.

(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 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.

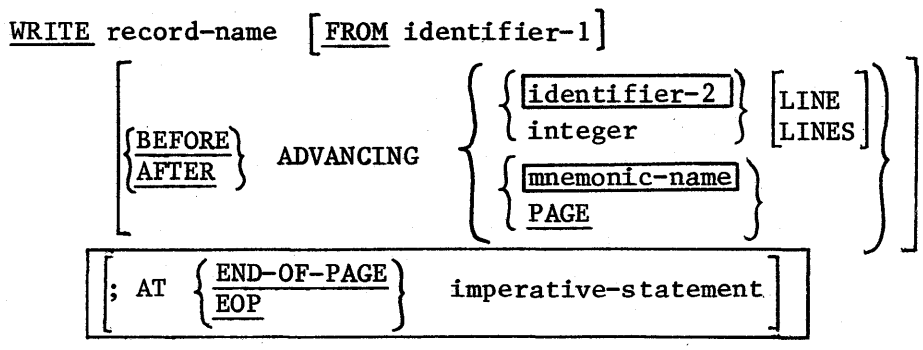
(4) Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

4.6 THE WRITE STATEMENT

4.6.1 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.

4.6.2 General Format



4.6.3 Syntax Rules

(1) Record-name and identifier-1 must not reference the same storage area.

(2) When mnemonic-name is specified, the name is associated with a particular feature specified by the implementor. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the Environment Division.

(3) The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

(4) When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

(5) Integer or 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 mnemonic-name phrase cannot be specified when writing a record to a file whose file description entry contains the LINAGE clause.

4.6.4 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 page IV-24, The OPEN Statement.)

(2) The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful due to a boundary violation. The logical record is also avail-

able 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-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 page IV-1, I-O Status.)

(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) Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided by the implementor to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

a. If identifier-2 is specified, the representation of the printed page is advanced the number of lines equal to the current value associated with identifier-2.

b. If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.

c. If mnemonic-name is specified, the representation of the printed page is advanced according to the rules specified by the implementor for that hardware device.

d. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to rules a, b, and c above.

e. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to rules a, b, and c above.

f. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file whose file description entry contains a LINAGE clause, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause. If the record to be written is associated with a file whose file description entry does not contain a LINAGE clause, the repositioning to the next logical page is accomplished in accordance with an implementor-defined technique. If page has no meaning in conjunction with a specific device, then advancing will be provided by the implementor to act as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

(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 causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and then the imperative statement in the END-OF-PAGE phrase is executed.

An automatic page overflow condition is reached whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. The imperative statement in the END-OF-PAGE clause, if specified, is executed after the record is written and the device has been repositioned.

If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

If integer-2 or data-name-2 of the LINAGE clause is specified, but the execution of a given WRITE statement would cause LINAGE-COUNTER 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 page IV-1, I-O Status.)

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.

(13) After the recognition of an end-of-reel or an end-of-unit of an output file that is contained on more than one physical reel/unit, the WRITE statement performs the following operations:

a. The standard ending reel/unit label procedure.

b. A reel/unit swap.

c. The standard beginning reel/unit label procedure.

1. INTRODUCTION TO THE RELATIVE I-O MODULE

1.1 FUNCTION

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 logical ordinal position in the file.

1.2 LEVEL CHARACTERISTICS

Relative I-O Level 1 does not provide full COBOL facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, the Relative I-O Level 1 provides limited capabilities for the READ and USE statements and full capabilities for the CLOSE, DELETE, OPEN, REWRITE, and WRITE statements, as specified in the formats of this module.

Relative I-O Level 2 provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, the Relative I-O Level 2 provides full capabilities for the CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, and WRITE statements as specified in the formats of this module. The additional features available in Level 2 include: the RESERVE clause, DYNAMIC accessing, SAME RECORD AREA, READ NEXT, and the entire START statement.

1.3 LANGUAGE CONCEPTS

1.3.1 Organization

Relative file organization is permitted only on mass storage 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 is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through the ninth record areas.

1.3.2 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.

1.3.3 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, START, and READ statements.

1.3.4 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.

1.3.4.1 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 Implementor Defined

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 result of an input-output error, such as data check, parity error, or transmission error.

9 - Implementor Defined. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the implementor. 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.

1.3.4.2 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:

1. If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

2. 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 as follows:

   a. A value of '2' in status key 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.

   b. A value of '3' in status key 2 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.

   c. A value of '4' in status key 2 indicates a boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a relative file. The implementor specifies the manner in which these boundaries are defined.

3. When status key 1 contains a value of '9' indicating an implementor-defined condition, the value of status key 2 is defined by the implementor.

1.3.4.3 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 figure. 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)				

1.3.5 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 page V-28, The START Statement; page V-23, The READ Statement; page V-32, The WRITE Statement; page V-26, The REWRITE Statement; and page V-19, The DELETE Statement.

When the INVALID KEY condition is recognized, the MSCS 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 page V-2, 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.

1.3.6 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 page V-23, The READ Statement.

2. ENVIRONMENT DIVISION IN THE RELATIVE I-O MODULE

2.1 INPUT-OUTPUT SECTION

2.1.1 The FILE-CONTROL Paragraph

2.1.1.1 Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

2.1.1.2 General Format

```
FILE-CONTROL. {file-control-entry} ...
```

2.1.2 The File Control Entry

2.1.2.1 Function

The file control entry names a file and may specify other file-related information.

2.1.2.2 General Format

```
SELECT file-name

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

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

    ; ORGANIZATION IS RELATIVE

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

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

2.1.2.3 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-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.

2.1.2.4 General Rules

(1) The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium.

(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. If the RESERVE clause is not specified the number of input-output areas allocated is specified by the implementor.

(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 page V-2, I-O Status.)

(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 one (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 MSCS.

2.1.3 The I-O-CONTROL Paragraph

2.1.3.1 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.

2.1.3.2 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} ... ] ... .
```

2.1.3.3 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:

a. 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] clause need not all have the same organization or access.

2.1.3.4 General Rules

(1) The RERUN clause specifies when and where the rerun information is recorded. Rerun information is recorded in the following ways:

a. If file-name-1 is specified, the rerun information is written on each reel or unit of an output file and the implementor specifies where, on the file, the rerun information is to be recorded.

b. If implementor-name is specified, the rerun information is written as a separate file on a device specified by the implementor.

(2) There are four forms of the RERUN clause, based on the several conditions under which rerun points can be established. The implementor must provide at least one of the specified forms of the RERUN clause.

a. When the integer-1 RECORDS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever integer-1 records or file-name-2 has been processed. File-name-2 may be either an input or output file with any organization or access.

b. When the integer-2 CLOCK-UNITS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever an interval of time, calculated by an internal clock has elapsed.

c. When the condition-name clause is used and implementor-name is specified in the ON clause. In this case, the rerun information is written on the device specified by implementor-name whenever a switch assumes a particular status as specified by condition-name. In this case, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

d. When the condition-name clause is used and file-name-1 is specified in the ON clause. In this case, the rerun information is written on file-name-1, which must be an output file, whenever a switch assumed a particular status as specified by condition-name. In this case, as in paragraph c above, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

(3) 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 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. (See syntax rule 6c on page V-7.)

(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.

3. DATA DIVISION IN THE RELATIVE I-O MODULE

3.1 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 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.

3.2 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 on page I-84 while the elements allowed in a record description are shown in the data description skeleton on page II-12.

3.3 THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

3.3.1 Function

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

3.3.2 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}
```

3.3.3 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.

(3) One or more record description entries must follow the file description entry.

3.4 THE BLOCK CONTAINS CLAUSE

3.4.1 Function

The BLOCK CONTAINS clause specifies the size of a physical record.

3.4.2 General Format

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

3.4.3 General Rules

(1) This clause is required except when:

a. A physical record contains one and only one complete logical record.

b. The hardware device assigned to the file has one and only one physical record size.

c. The hardware device assigned to the file has more than one physical record size but the implementor has designated one as standard. In this case, the absence of this clause denotes the standard physical record size.

(2) The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exists, in which case the RECORDS phrase must not be used

a. Where logical records may extend across physical records.

b. The physical record contains padding (area not contained in a logical record).

c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

(3) When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

(4) If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

(5) If logical records of differing size are grouped into one physical record, the technique for determining the size of each logical record is specified by the implementor.

3.5 THE DATA RECORDS CLAUSE

3.5.1 Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

3.5.2 General Format

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

3.5.3 Syntax Rules

(1) 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.

3.5.4 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 in no way altered by the presence of more than one type of data record within the file.

3.6 THE LABEL RECORDS CLAUSE

3.6.1 Function

The LABEL RECORDS clause specifies whether labels are present.

3.6.2 General Format

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

3.6.3 Syntax Rules

(1) This clause is required in every file description entry.

3.6.4 General Rules

(1) OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

(2) STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the implementor's label specifications.

3.7 THE RECORD CONTAINS CLAUSE

3.7.1 Function

The RECORD CONTAINS clause specifies the size of data records.

3.7.2 General Format

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

3.7.3 General Rules

(1) The size of each data record is completely defined within the record description entry, therefore this clause is never required. When present, however, the following notes apply:

a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record; see page I-85, Selection of Character Representation and Radix; page II-33, The SYNCHRONIZED Clause; and page II-35, The USAGE Clause.

3.8 THE VALUE OF CLAUSE

3.8.1 Function

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

3.8.2 General Format

```
VALUE OF implementor-name-1 IS { data-name-1 }
                               { literal-1   }

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

3.8.3 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.

3.8.4 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.

4. PROCEDURE DIVISION IN THE RELATIVE I-O MODULE

4.1 THE CLOSE STATEMENT

4.1.1 Function

The CLOSE statement terminates the processing of files with optional lock.

4.1.2 General Format

CLOSE file-name-1 [WITH LOCK] [, file-name-2 [WITH LOCK]] ...

4.1.3 Syntax Rules

(1) The files referenced in the CLOSE statement need not all have the same organization or access.

4.1.4. General Rules

(1) A CLOSE statement may only be executed for a file in an open mode.

(2) Relative files are classified as belonging to the category of non-sequential single/multi-reel/unit. The results of executing each type of CLOSE for this category of file are summarized in the following table.

CLOSE Statement Format	File Category = Non-sequential Single/Multi-Reel/Unit
CLOSE	A
CLOSE WITH LOCK	A,B

The definitions of the symbols in the table are given below. Where the definition depends on whether the file is an input, output or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Close File

Input Files and Input-Output Files (Sequential Access Mode):

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations specified by the implementor are executed. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed. If the file is positioned other than at its end, the closing operations specified by the implementor are executed, but there is no ending label processing.

Input Files and Input-Output Files (Random or Dynamic Access Mode); Output Files (Random, Dynamic, or Sequential Access Mode):

If label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations specified by the implementor are executed. If label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed.

B. File Lock

An implementor-defined technique is supplied to insure that this file cannot be opened again during this execution of this run unit.

(3) The action taken if a file is in the open mode when a STOP RUN statement is executed is specified by the implementor. 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 also specified by the implementor.

(4) 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.

(5) 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.

4.2 THE DELETE STATEMENT

4.2.1 Function

The DELETE statement logically removes a record from a mass storage file.

4.2.2 General Format

DELETE file-name RECORD [; INVALID KEY imperative-statement]

4.2.3 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.

4.2.4 General Rules

(1) The associated file must be open in the I-O mode at the time of the execution of this statement. (See page V-20, The OPEN Statement.)

(2) For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The MSCS 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 MSCS 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 page V-4, The INVALID KEY Condition.)

(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 page V-2, I-O Status.)

4.3 THE OPEN STATEMENT

4.3.1 Function

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

4.3.2 General Format

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

4.3.3 Syntax Rules

(1) The files referenced in the OPEN statement need not all have the same organization or access.

4.3.4 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 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 relative file organization and the open mode given at the top of the column.

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

Table 1. Permissible Statements

(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, without the LOCK phrase, for that file.

(6) Execution of the OPEN statement does not obtain or release the first data record.

(7) If label records are specified for the file, the beginning labels are processed as follows:

a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with the implementor's specified conventions for input label checking.

b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with the implementor's specified conventions for output label writing.

The behavior of the OPEN statement when label records are specified but not present, or when label records are not specified but are present, is undefined.

(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. 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.

(10) The I-O phrase permits the opening of a file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the file is being initially created.

(11) When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

a. The labels are checked in accordance with the implementor's specified conventions for input-output label checking.

b. The new labels are written in accordance with the implementor's specified conventions for input-output label writing.

(12) 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.

4.4 THE READ STATEMENT

4.4.1 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.

4.4.2 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]
```

4.4.3 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.

4.4.4 General Rules

(1) The associated files must be open in the INPUT or I-O mode at the time this statement is executed. (See page V-20, The OPEN Statement.)

(2) The record to be made available by a Format 1 READ statement is determined as follows:

a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, 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 page V-2, I-O Status.)

(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 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 page V-2, I-O Status.)

(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 page V-2, 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.

(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 recognized, a Format 1 READ statement for that file must not be executed without first executing one of the following:

a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

b. A successful START statement for that file.

c. A successful Format 2 READ statement for that file.

(13) For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified 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 page V-4, The INVALID KEY Condition.)

4.5 THE REWRITE STATEMENT

4.5.1 Function

The REWRITE statement logically replaces a record existing in a mass storage file.

4.5.2 General Format

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

4.5.3 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 which references a file in sequential access mode.

(4) The INVALID KEY phrase must be specified in the REWRITE statement for files in the random [or dynamic] access mode for which an appropriate USE procedure is not specified.

4.5.4 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 page V-20, The OPEN Statement.)

(2) For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The MSCS 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 page V-2, I-O Status.)

(8) For a file accessed in either random or dynamic access mode, the MSCS 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 page V-3, The INVALID KEY Condition.) The updating operation does not take place and the data in the record area is unaffected.

4.6 THE START STATEMENT

4.6.1 Function

The START statement provides a basis for logical positioning within a relative file, for subsequent sequential retrieval of records.

4.6.2 General Format

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

  [; INVALID KEY imperative-statement]
```

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

4.6.3 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.

4.6.4 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 page V-20, The OPEN Statement.)

(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 V-4, The INVALID KEY Condition.)

(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 page V-2, I-O Status.)

(5) The comparison described in general rule 3 uses the data item referenced by the RELATIVE KEY clause associated with file-name.

4.7 THE USE STATEMENT

4.7.1 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.

4.7.2 General Format

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

4.7.3 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 referenced in a USE statement need not all have the same organization or access.

4.7.4 General Rules

(1) The designated procedures are executed by the input-output system after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions, when the INVALID KEY phrase or AT END phrase, respectively, has not been specified in the input-output statement.

(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 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.

4.8 THE WRITE STATEMENT

4.8.1 Function

The WRITE statement releases a logical record for an output or input-output file.

4.8.2 General Format

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

4.8.3 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.

4.8.4 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 page V-20, The OPEN Statement.)

(2) The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement 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.)

(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 page V-2, I-O Status.)

(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 MSCS. The first record will have a relative record number of one (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 MSCS during execution of the WRITE statement.

b. If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released to the MSCS 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 initialized 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 MSCS.

(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 recognized, the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the FILE STATUS data item, if any, of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated in the INVALID KEY condition on page V-3. (See page V-2, I-O Status.)

1. INTRODUCTION TO THE INDEXED I-O MODULE

1.1 FUNCTION

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.

1.2 LEVEL CHARACTERISTICS

Indexed I-O Level 1 does not provide full COBOL facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats of this module. Within the Procedure Division, the Indexed I-O Level 1 provides limited capabilities for the READ and USE statements and full capabilities for the CLOSE, DELETE, OPEN, REWRITE, and WRITE statements, as specified in the formats for this module.

Indexed I-O Level 2 provides full facilities for the FILE-CONTROL, I-O-CONTROL, and FD entries as specified in the formats for this module. Within the Procedure Division, the Indexed I-O Level 2 provides full capabilities for the CLOSE, DELETE, OPEN, READ, REWRITE, START, USE, and WRITE statements as specified in the formats for this module. The additional features available in Level 2 include: the RESERVE clause, DYNAMIC accessing, ALTERNATE KEYS, SAME RECORD AREA, READ NEXT, and the entire START statement.

1.3 LANGUAGE CONCEPTS

1.3.1 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 alternate record key for that file. The value of an alternate record key may be non-unique if the DUPLICATES phrase is specified for it. These keys provide alternate access paths for retrieval of records from the file.

1.3.2 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 order 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 a 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.

1.3.3 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, START, and READ statements.

1.3.4 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.

1.3.4.1 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 Implementor Defined

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 - Implementor Defined. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the implementor. 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.

1.3.4.2 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:

1. If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

2. 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:

a. 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.

b. 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 is used to designate the cause of that condition as follows:

a. A value of '1' in status key 2 indicates a sequence error for a sequentially accessed indexed file. The ascending sequence requirements of successive record key values have been violated (see The WRITE Statement on page VI-33), 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.

b. A value of '2' in status key 2 indicates a duplicate key value. An attempt has been made to write or rewrite a record that would create a duplicate key in an indexed file.

c. A value of '3' in status key 2 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.

d. A value of '4' in status key 2 indicates a boundary violation. An attempt has been made to write beyond the externally defined boundaries of an indexed file. The implementor specifies the manner in which these boundaries are defined.

4. When status key 1 contains a value of '9' indicating an implementor-defined condition, the value of status key 2 is defined by the implementor.

1.3.4.3 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 figure. 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		X		
At End (1)	X				
Invalid Key (2)		X	X	X	X
Permanent Error (3)	X				
Implementor Defined (9)					

1.3.5 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 page VI-30, The START Statement; page VI-24, The READ Statement; page VI-33, The WRITE Statement; page VI-28, The REWRITE Statement; and page VI-20, The DELETE Statement.

When the INVALID KEY condition is recognized, the MSCS 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 page VI-2, 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.

1.3.6 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 page VI-24, The READ Statement.

2. ENVIRONMENT DIVISION IN THE INDEXED I-O MODULE

2.1 INPUT-OUTPUT SECTION

2.1.1 The FILE-CONTROL Paragraph

2.1.1.1 Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

2.1.1.2 General Format

```
FILE-CONTROL.  {file-control-entry} ...
```

2.1.2 The File Control Entry

2.1.2.1 Function

The file control entry names a file and may specify other file-related information.

2.1.2.2 General Format

```
SELECT file-name

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

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

    ; ORGANIZATION IS INDEXED

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

    ; RECORD KEY IS data-name-1

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

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

2.1.2.3 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, the Report Section, or the Communication 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 page III-2, The OCCURS Clause.)

(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.

2.1.2.4 General Rules

(1) The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium.

(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. If the RESERVE clause is not specified the number of input-output areas allocated is specified by the implementor.

(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 page VI-2, I-O Status.)

(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 alternate record key for the file. This alternate record key provides an alternate access path to records in an indexed file.

(10) The data descriptions of data-name-1 [and data-name-2] as well as their 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.

2.1.3 The I-O-CONTROL Paragraph

2.1.3.1 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.

2.1.3.2 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} ... ] ...  .
```

2.1.3.3 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 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.

2.1.3.4 General Rules

(1) The RERUN clause specifies when and where the rerun information is recorded. Rerun information is recorded in the following ways:

a. If file-name-1 is specified, the rerun information is written on each reel or unit of an output file and the implementor specifies where, on the file, the rerun information is to be recorded.

b. If implementor-name is specified, the rerun information is written as a separate file on a device specified by the implementor.

(2) There are four forms of the RERUN clause, based on the several conditions under which rerun points can be established. The implementor must provide at least one of the specified forms of the RERUN clause.

a. When the integer-1 RECORDS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever integer-1 records of file-name-2 has been processed. File-name-2 may be either an input or output file with any organization or access.

b. When the integer-2 CLOCK-UNITS clause is used. In this case, the rerun information is written on the device specified by implementor-name, which must be specified in the ON clause, whenever an interval of time, calculated by an internal clock, has elapsed.

c. When the condition-name clause is used and implementor-name is specified in the ON clause. In this case, the rerun information is written on the device specified by implementor-name whenever a switch assumes a particular status as specified by condition-name. In this case, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

d. When the condition-name clause is used and file-name-1 is specified in the ON clause. In this case, the rerun information is written on file-name-1, which must be an output file, whenever a switch assumes a particular status as specified by condition-name. In this case, as in paragraph c above, the associated switch must be defined in the SPECIAL-NAMES paragraph of the Configuration Section of the Environment Division. The implementor specifies when the switch status is interrogated.

(3) 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 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 on page VI-8.)

(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.

3. DATA DIVISION IN THE INDEXED I-O MODULE

3.1 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 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.

3.2 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 on page I-84 while the elements allowed in a record description are shown in the data description skeleton on page II-12.

3.3 THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

3.3.1 Function

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

3.3.2 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 / RECORDS ARE} {STANDARD / OMITTED}

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

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

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

3.3.3 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.

(3) One or more record description entries must follow the file description entry.

3.4 THE BLOCK CONTAINS CLAUSE

3.4.1 Function

The BLOCK CONTAINS clause specifies the size of a physical record.

3.4.2 General Format

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

3.4.3 General Rules

(1) This clause is required except when:

a. A physical record contains one and only one complete logical record.

b. The hardware device assigned to the file has one and only one physical record size.

c. The hardware device assigned to the file has more than one physical record size but the implementor has designated one as standard. In this case, the absence of this clause denotes the standard physical record size.

(2) The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exists, in which case the RECORDS phrase must not be used

a. Where logical records may extend across physical records.

b. The physical record contains padding (area not contained in a logical record).

c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

(3) When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

(4) If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

(5) If logical records of differing size are grouped into one physical record, the technique for determining the size of each logical record is specified by the implementor.

3.5 THE DATA RECORDS CLAUSE

3.5.1 Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

3.5.2 General Format

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

3.5.3 Syntax Rules

(1) 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.

3.5.4 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 in no way altered by the presence of more than one type of data record within the file.

3.6 THE LABEL RECORDS CLAUSE

3.6.1 Function

The LABEL RECORDS clause specifies whether labels are present.

3.6.2 General Format

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

3.6.3 Syntax Rules

(1) This clause is required in every file description entry.

3.6.4 General Rules

(1) OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

(2) STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the implementor's label specifications.

3.7 THE RECORD CONTAINS CLAUSE

3.7.1 Function

The RECORD CONTAINS clause specifies the size of data records.

3.7.2 General Format

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

3.7.3 General Rules

(1) The size of each data record is completely defined within the record description entry, therefore this clause is never required. When present, however, the following notes apply:

a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record; see page I-85, Selection of Character Representation and Radix; page II-33, The SYNCHRONIZED Clause; and page II-35, The USAGE Clause.

3.8 THE VALUE OF CLAUSE

3.8.1 Function

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

3.8.2 General Format

```
VALUE OF implementor-name-1 IS { data-name-1 }
                               { literal-1   }

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

3.8.3 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.

3.8.4 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.

4. PROCEDURE DIVISION IN THE INDEXED I-O MODULE

4.1 THE CLOSE STATEMENT

4.1.1 Function

The CLOSE statement terminates the processing of files with optional lock.

4.1.2 General Format

CLOSE file-name-1 [WITH LOCK] [, file-name-2 [WITH LOCK]] ...

4.1.3 Syntax Rules

(1) The files referenced in the CLOSE statement need not all have the same organization or access.

4.1.4 General Rules

(1) A CLOSE statement may only be executed for a file in an open mode.

(2) Indexed files are classified as belonging to the category of non-sequential single/multi-reel/unit. The results of executing each type of CLOSE for this category of file are summarized in the following table.

CLOSE Statement Format	File Category = Non-sequential Single/Multi-Reel/Unit
CLOSE	A
CLOSE WITH LOCK	A,B

The definitions of the symbols in the table are given below. Where the definition depends on whether the file is an input, output, or input-output file, alternate definitions are given; otherwise, a definition applies to input, output, and input-output files.

A. Close File

Input Files and Input-Output Files (Sequential Access Mode):

If the file is positioned at its end and label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations specified by the implementor are executed. If the file is positioned at its end and label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed. If the file is positioned other than at its end, the closing operations specified by the implementor are executed, but there is no ending label processing.

Input Files and Input-Output Files (Random or Dynamic Access Mode); Output Files (Random, Dynamic, or Sequential Access Mode):

If label records are specified for the file, the labels are processed according to the implementor's standard label convention. The behavior of the CLOSE statement when label records are specified but not present, or when label records are not specified but are present, is undefined. Closing operations specified by the implementor are executed. If label records are not specified for the file, label processing does not take place but other closing operations specified by the implementor are executed.

B. File Lock

An implementor-defined technique is supplied to insure that this file cannot be opened again during this execution of this run unit.

(3) The action taken if a file is in the open mode when a STOP RUN statement is executed is specified by the implementor. 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 also specified by the implementor.

(4) 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.

(5) 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.

4.2 THE DELETE STATEMENT

4.2.1 Function

The DELETE statement logically removes a record from a mass storage file.

4.2.2 General Format

DELETE file-name RECORD [; INVALID KEY imperative-statement]

4.2.3 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.

4.2.4 General Rules

(1) The associated file must be open in the I-O mode at the time of the execution of this statement. (See page VI-21, The OPEN Statement.)

(2) For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The MSCS 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 MSCS 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 page VI-4, The INVALID KEY Condition.)

(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 page VI-2, I-O Status.)

4.3 THE OPEN STATEMENT

4.3.1 Function

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

4.3.2 General Format

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

4.3.3 Syntax Rules

(1) The files referenced in the OPEN statement need not all have the same organization or access.

4.3.4 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 2, 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.

File Access Mode	Statement	Open Mode		
		Input	Output	Input-Ouput
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

Table 2. Permissible Statements

(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, without the LOCK phrase, for that file.

(6) Execution of the OPEN statement does not obtain or release the first data record.

(7) If label records are specified for the file, the beginning labels are processed as follows:

a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with the implementor's specified conventions for input label checking.

b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with the implementor's specified conventions for output label writing.

The behavior of the OPEN statement when label records are specified but not present, or when label records are not specified but are present, is undefined.

(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 key 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.

(10) The I-O phrase permits the opening of a file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the file is being initially created.

(11) When the I-O phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following steps:

a. The labels are checked in accordance with the implementor's specified conventions for input-output label checking.

b. The new labels are written in accordance with the implementor's specified conventions for input-output label writing.

(12) 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.

4.4 THE READ STATEMENT

4.4.1 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.

4.4.2 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]
```

4.4.3 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.

4.4.4 General Rules

(1) The associated file must be open in the INPUT or I-O mode at the time this statement is executed. (See page VI-21, The OPEN Statement.)

(2) The record to be made available by a Format 1 READ statement is determined as follows:

a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, 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 page VI-2, I-O Status.)

(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 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 page VI-2, I-O Status.)

(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 page VI-2, 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.

(11) Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined. For indexed files the key of reference is also undefined.

(12) When the AT END condition has been recognized, a Format 1 READ statement for that file must not be executed without first executing one of the following:

a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

b. A successful START statement for that file.

c. A successful Format 2 READ statement for that file.

(13) For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in general rule 2.

(14) For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

(15) For an indexed file if the KEY phrase is specified in a Format 2 READ statement, data-name is established as the key of reference for 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.

(16) If the KEY phrase is not specified in a Format 2 READ statement, the prime record key is established as the key of reference for 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 page VI-4, The INVALID KEY Condition.)

4.5 THE REWRITE STATEMENT

4.5.1 Function

The REWRITE statement logically replaces a record existing in a mass storage file.

4.5.2 General Format

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

4.5.3 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 be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

4.5.4 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 page VI-21, The OPEN Statement.)

(2) For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The MSCS 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 page VI-2, 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 alternate record key data items of the record being rewritten may differ from those in the record being replaced. The MSCS 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 page VI-4, The INVALID KEY Condition.)

4.6 THE START STATEMENT

4.6.1 Function

The START statement provides a basis for logical positioning within an indexed file, for subsequent sequential retrieval of records.

4.6.2 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 '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

4.6.3 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 alphanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of that record key data item.

4.6.4 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 page VI-21, The OPEN Statement.)

(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 page II-42, 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 page VI-4, The INVALID KEY Condition.)

(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 page VI-2, 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 page VI-24, The READ Statement.)

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.

4.7 THE USE STATEMENT

4.7.1 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.

4.7.2 General Format

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

4.7.3 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 referenced in a USE statement need not all have the same organization or access.

4.7.4 General Rules

(1) The designated procedures are executed by the input-output system after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions, when the INVALID KEY phrase or AT END phrase, respectively, has not been specified in the input-output statement.

(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 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.

4.8 THE WRITE STATEMENT

4.8.1 Function

The WRITE statement releases a logical record for an output or input-output file.

4.8.2 General Format

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

4.8.3 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.

4.8.4 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 page VI-21, The OPEN Statement.)

(2) The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement 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.)

(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 page VI-2, I-O Status.)

(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 MSCS utilizes the content of the record keys in such a way that subsequent access of the record key 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. (See general rule 3.)

(12) If sequential access mode is specified for the file, records must be released to the MSCS in ascending order of prime record key values.

(13) If random or dynamic access mode is specified, records may be released to the MSCS 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 MSCS 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 MSCS.

(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 recognized the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected and the FILE STATUS data item, if any, 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 on page VI-4, The INVALID KEY Condition. (See page VI-2, I-O Status.)

1. INTRODUCTION TO THE SORT-MERGE MODULE

1.1 FUNCTION

The Sort-Merge module provides the capability to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each record. Optionally, a user may apply some special processing to each of the individual records by input or output procedures. This special processing may be applied before and/or after the records are ordered by the SORT or after the records have been combined by the MERGE.

1.2 LEVEL CHARACTERISTICS

Sort-Merge Level 1 provides the facility for sorting a single file only once within a given execution of a COBOL program. Procedures for special handling of each record in the file before and/or after it has been sorted are also provided.

Sort-Merge Level 2 provides the facility for sorting one or more files, or combining two or more files, one or more times within a given execution of a COBOL program.

1.3 RELATIONSHIP WITH SEQUENTIAL I-O MODULE

The files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described implicitly or explicitly in the FILE-CONTROL paragraph as having sequential organization. No input-output statement may be executed for the file named in the sort-merge file description.

2. ENVIRONMENT DIVISION IN THE SORT-MERGE MODULE

2.1 INPUT-OUTPUT SECTION

2.1.1 The FILE-CONTROL Paragraph

2.1.1.1 Function

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

2.1.1.2 General Format

FILE-CONTROL. {file-control-entry} ...

2.1.2 The File Control Entry

2.1.2.1 Function

The file control entry names a sort [or merge] file and specifies the association of the file to a storage medium.

2.1.2.2 General Format

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

2.1.2.3 Syntax Rules

(1) Each sort [or merge] file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph. Each sort [or merge] file specified in the file control entry must have a sort-merge file description entry in the Data Division.

(2) Since file-name represents a sort [or merge] file, only the ASSIGN clause is permitted to follow file-name in the FILE-CONTROL paragraph.

2.1.2.4 General Rules

(1) The ASSIGN clause specifies the association of the sort [or merge] file referenced by file-name to a storage medium.

2.1.3 The I-O-CONTROL Paragraph

2.1.3.1 Function

The I-O-CONTROL paragraph specifies the memory area which is to be shared by different files.

2.1.3.2 General Format

```
I-O-CONTROL.
[; SAME { RECORD    } AREA FOR file-name-1 {, file-name-2} ... ] ... .
        { SORT      }
        { SORT-MERGE}
```

2.1.3.3 Syntax Rules

(1) The I-O-CONTROL paragraph is optional.

(2) In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

(3) If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause.

(4) The three formats of the SAME clause (SAME RECORD AREA, SAME SORT AREA, SAME SORT-MERGE AREA) are considered separately in the following:

More than one SAME clause may be included in a program, however:

a. A file-name must not appear in more than one SAME RECORD AREA clause.

b. A file-name that represents a sort or merge file must not appear in more than one SAME SORT AREA or SAME SORT-MERGE AREA clause.

c. If a file-name that does not represent a sort or merge file appears in a SAME AREA clause and one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, all of the files named in that SAME AREA clause must be named in that SAME SORT AREA or SAME SORT-MERGE AREA clause(s). (See page IV-6, Sequential I-O.)

(5) The files referenced in the SAME SORT AREA, SAME SORT-MERGE AREA, or SAME RECORD AREA clause need not all have the same organization or access.

2.1.3.4 General Rules

(1) 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 implicit redefinition of the area, i.e., records are aligned on the leftmost character position.

(2) If the SAME SORT AREA or SAME SORT-MERGE AREA clause is used, at least one of the file-names must represent a sort or merge file. Files that do not represent sort or merge files may also be named in the clause. This clause specifies that storage is shared as follows:

a. The SAME SORT AREA or SAME SORT-MERGE AREA clause specifies a memory area which will be made available for use in sorting or merging each sort or merge file named. Thus any memory area allocated for the sorting or merging of a sort or merge file is available for reuse in sorting or merging any of the other sort or merge files.

b. In addition, storage areas assigned to files that do not represent sort or merge files may be allocated as needed for sorting or merging the sort or merge files named in the SAME SORT AREA or SAME SORT-MERGE AREA clause. The extent of such allocation will be specified by the implementor.

c. Files other than sort or merge files do not share the same storage area with each other. If the user wishes these files to share the same storage area with each other, he must also include in the program a SAME AREA or SAME RECORD AREA clause naming these files.

d. During the execution of a SORT or MERGE statement that refers to a sort or merge file named in this clause, any non sort-merge files named in this clause must not be open.

3. DATA DIVISION IN THE SORT-MERGE MODULE

3.1 FILE SECTION

An SD file description gives information about the size and the names of the data records associated with the file to be sorted or merged. There are no label procedures which the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements.

3.2 THE SORT-MERGE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

3.2.1 Function

The sort-merge file description furnishes information concerning the physical structure, identification, and record names of the file to be sorted or merged.

3.2.2 General Format

```
SD  file-name

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

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

3.2.3 Syntax Rules

(1) The level indicator SD identifies the beginning of the sort-merge file description and must precede the file-name.

(2) The clauses which follow the name of the file are optional and their order of appearance is immaterial.

(3) One or more record description entries must follow the sort-merge file description entry, however, no input-output statements may be executed for this file.

3.3 THE DATA RECORDS CLAUSE

3.3.1 Function

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

3.3.2 General Format

$$\underline{\text{DATA}} \left\{ \begin{matrix} \underline{\text{RECORD}} \text{ IS} \\ \underline{\text{RECORDS}} \text{ ARE} \end{matrix} \right\} \text{ data-name-1 } \left[\text{, data-name-2} \right] \ \ldots$$

3.3.3 Syntax Rules

(1) 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.

3.3.4 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 in no way altered by the presence of more than one type of data record within the file.

3.4 THE RECORD CONTAINS CLAUSE

3.4.1 Function

The RECORD CONTAINS clause specifies the size of data records.

3.4.2 General Format

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

3.4.3 General Rules

(1) The size of each data record is completely defined within the record description entry, therefore this clause is never required. When present, however, the following notes apply:

a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record; see page I-85, Selection of Character Representation and Radix; page II-33, The SYNCHRONIZED Clause; and page II-35, The USAGE Clause.

4. PROCEDURE DIVISION IN THE SORT-MERGE MODULE

4.1 THE MERGE STATEMENT

4.1.1 Function

The MERGE statement combines two or more identically sequenced files on a set of specified keys, and during the process makes records available, in merge order, to an output procedure or to an output file.

4.1.2 General Format

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

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

  [COLLATING SEQUENCE IS alphabet-name]

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

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

4.1.3 Syntax Rules

(1) File-name-1 must be described in a sort-merge file description entry in the Data Division.

(2) Section-name-1 represents the name of an output procedure.

(3) File-name-2, file-name-3, file-name-4, and file-name-5 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, file-name-3, file-name-4, and file-name-5 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause an equal number of character positions to be allocated for the corresponding records.

(4) The words THRU and THROUGH are equivalent.

(5) Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

a. The data items identified by KEY data-names must be described in records associated with file-name-1.

b. KEY data-names may be qualified.

c. The data items identified by KEY data-names must not be variable length items.

d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

(6) No more than one file-name from a multiple file reel can appear in the MERGE statement.

(7) File-names must not be repeated within the MERGE statement.

(8) MERGE statements may appear anywhere except in the declaratives portion of the Procedure Division or in an input or output procedure associated with a SORT or MERGE statement.

4.1.4 General Rules

(1) The MERGE statement will merge all records contained on file-name-2, file-name-3, and file-name-4. The files referenced in the MERGE statement must not be open at the time the MERGE statement is executed. These files are automatically opened and closed by the merge operation with all implicit functions performed, such as the execution of any associated USE procedures. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file.

(2) The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, etc.

a. When the ASCENDING phrase is specified, the merged sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the merged sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

(3) The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

b. Second, the collating sequence established as the program collating sequence.

(4) The output procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any other procedure. In order to make merged records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time in merged order, from file-name-1. The restrictions on the procedural statements within the output procedure are as follows:

a. The output procedure must not contain any transfers of control to points outside the output procedure; ALTER, GO TO and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

b. The output procedures must not contain any SORT or MERGE statements.

c. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedures; ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedures.

(5) If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the merge, and then passes control to the next executable statement after the MERGE statement. Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statements in the output procedure are the requests for the next record.

(6) Segmentation, as defined in Section IX, can be applied to programs containing the MERGE statement. However, the following restrictions apply:

a. If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

1) Totally within non-independent segments, or

2) Wholly contained in a single independent segment.

b. If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

1) Totally within non-independent segments, or

2) Wholly within the same independent segment as that MERGE statement.

(7) If the GIVING phrase is specified, all the merged records in file-name-1 are automatically written on file-name-5 as the implied output procedure for this MERGE statement.

(8) In the case of an equal compare, according to the rules for comparison of operands in a relation condition, on the contents of the data items identified by all the KEY data-names between records from two or more input files (file-name-2, file-name-3, file-name-4, ...), the records are written on file-name-5 or returned to the output procedure, depending on the phrase specified, in the order that the associated input files are specified in the MERGE statement.

(9) The results of the merge operation are predictable only when the records in the files referenced by file-name-2, file-name-3, ..., are ordered as described in the ASCENDING or DESCENDING KEY clause associated with the MERGE statement.

4.2 THE RELEASE STATEMENT

4.2.1 Function

The RELEASE statement transfers records to the initial phase of a SORT operation.

4.2.2 General Format

RELEASE record-name [FROM identifier]

4.2.3 Syntax Rules

(1) A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement for a file whose sort-merge file description entry contains record-name. (See page VII-14, The SORT Statement.)

(2) Record-name must be the name of a logical record in the associated sort-merge file description entry and may be qualified.

(3) Record-name and identifier must not refer to the same storage area.

4.2.4 General Rules

(1) The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.

(2) If the FROM phrase is used, the contents of the identifier data area are moved to record-name, then the contents of record-name are released to the sort file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase. The information in the record area is no longer available, but the information in the data area associated with identifier is available.

(3) After the execution of the RELEASE statement, the logical record is no longer available in the record area unless the associated sort-merge file is named in a SAME RECORD AREA clause. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated sort-merge file, as well as to the file associated with record-name. When control passes from the input procedure, the file consists of all those records which were placed in it by the execution of RELEASE statements.

4.3 THE RETURN STATEMENT

4.3.1 Function

The RETURN statement obtains [either] sorted records from the final phase of a SORT operation [or merged records during a MERGE operation.]

4.3.2 General Format

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

4.3.3 Syntax Rules

(1) File-name must be described by a sort-merge file description entry in the Data Division.

(2) A RETURN statement may only be used within the range of an output procedure associated with a SORT [or MERGE] statement for file-name.

(3) The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

4.3.4 General Rules

(1) When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the RETURN statement.

(2) The execution of the RETURN statement causes the next record, in the order specified by the keys listed in the SORT [or MERGE] statement, to be made available for processing in the record areas associated with the sort [or merge] file.

(3) If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

(4) When the INTO phrase is used, the data is available in both the input record area and the data area associated with identifier.

(5) If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs. The contents of the record areas associated with the file when the AT END condition occurs are undefined. After the execution of the imperative-statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

4.4 THE SORT STATEMENT

4.4.1 Function

The SORT statement creates a sort file by executing input procedures or by transferring records from another file, sorts the records in the sort file on a set of specified keys, and in the final phase of the sort operation, makes available each record from the sort file, in sorted order, to some output procedures or to an output file.

4.4.2 General Format

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1  [, data-name-2 ] ...
                    { DESCENDING }
                 [ ON { ASCENDING  } KEY data-name-3  [, data-name-4] ... ] ...
                      { DESCENDING }
[COLLATING SEQUENCE IS alphabet-name]
{ INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
{                                     { THRU    }                 }
{ USING file-name-2 [, file-name-3] ...                           }
{ OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
{                                      { THRU    }                 }
{ GIVING file-name-4                                              }
```

4.4.3 Syntax Rules

(1) File-name-1 must be described in a sort-merge file description entry in the Data Division.

(2) Section-name-1 represents the name of an input procedure. Section-name-3 represents the name of an output procedure.

(3) File-name-2, file-name-3 and file-name-4 must be described in a file description entry, not in a sort-merge file description entry, in the Data Division. The actual size of the logical record(s) described for file-name-2, file-name-3 and file-name-4 must be equal to the actual size of the logical record(s) described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner so as to cause equal amounts of character positions to be allocated for the corresponding records.

(4) Data-name-1, data-name-2, data-name-3, and data-name-4 are KEY data-names and are subject to the following rules:

a. The data items identified by KEY data-names must be described in records associated with file-name-1.

b. KEY data-names may be qualified.

c. The data items identified by KEY data-names must not be variable length items.

d. If file-name-1 has more than one record description, then the data items identified by KEY data-names need be described in only one of the record descriptions.

e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

(5) The words THRU and THROUGH are equivalent.

(6) SORT statements may appear anywhere except in the declaratives portion of the Procedure Division or in an input or output procedure associated with a SORT [or MERGE] statement.

(7) No more than one file-name from a multiple file reel can appear in the SORT statement.

4.4.4 General Rules

(1) In Level 1, the Procedure Division of a program contains one SORT statement and a STOP RUN statement in the first non-declarative portion. Other sections consist of only the input and output procedures associated with the SORT statement.

(2) In Level 2, the Procedure Division may contain more than one SORT statement appearing anywhere except:

a. in the declaratives portion, or

b. in the input and output procedures associated with a SORT or MERGE statement.

(3) The data-names following the word KEY are listed from left to right in the SORT statement in order of decreasing significance without regard to how they are divided into KEY phrases. In the format, data-name-1 is the major key, data-name-2 is the next most significant key, etc.

a. When the ASCENDING phrase is specified, the sorted sequence will be from the lowest value of the contents of the data items identified by the KEY data-names to the highest value, according to the rules for comparison of operands in a relation condition.

b. When the DESCENDING phrase is specified, the sorted sequence will be from the highest value of the contents of the data items identified by the KEY data-names to the lowest value, according to the rules for comparison of operands in a relation condition.

(4) The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

a. First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in the SORT statement.

b. Second, the collating sequence established as the program collating sequence.

(5) The input procedure must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. In order to transfer records to the file referenced by file-name-1, the input procedure must include the execution of at least one RELEASE statement. Control must not be passed to the input procedure except when a related SORT statement is being executed. The input procedure can include any procedures needed to select, create, or modify records. The restrictions on the procedural statements within the input procedure are as follows:

a. The input procedure must not contain any SORT [or MERGE] statements.

b. The input procedure must not contain any explicit transfers of control to points outside the input procedure; ALTER, GO TO, and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

c. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

(6) If an input procedure is specified, control is passed to the input procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

(7) The output procedure must consist of one or more sections that appear contiguously in a source program and do not form part of any input procedure. In order to make sorted records available for processing, the output procedure must include the execution of at least one RETURN statement. Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure may consist of any procedures needed to select, modify or copy the records that are being returned, one at a time in sorted order, from the sort file. The restrictions on the procedural statements within the output procedure are as follows:

a. The output procedure must not contain any SORT [or MERGE] statements.

b. The output procedure must not contain any explicit transfers of control to points outside the output procedure; ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to declaratives.

c. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure; ALTER, GO TO and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

(8) If an output procedure is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the output procedure and when control passes the last statement in the output procedure, the return mechanism provides for termination of the sort and then passes control to the next executable statement after the SORT statement. Before entering the output procedure, the sort procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

(9) Segmentation as defined in Section IX can be applied to programs containing the SORT statement. However, the following restrictions apply:

a. If a SORT statement appears in a section that is not in an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

1) Totally within non-independent segments, or

2) Wholly contained in a single independent segment.

b. If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

1) Totally within non-independent segments, or

2) Wholly within the same independent segment as that SORT statement.

(10) If the USING phrase is specified, all the records in file-name-2 [and file-name-3] are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 [and file-name-3] must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2 [and file-name-3.] These implicit functions are performed such that any associated USE procedures are executed. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, had been executed for each file. The SORT statement also automatically performs the implicit functions of moving the records from the file area of file-name-2 [and file-name-3] to the file area for file-name-1 and the release of records to the initial phase of the sort operation.

(11) If the GIVING phrase is specified, all the sorted records in file-name-1 are automatically written on file-name-4 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement file-name-4 must not be open. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of file-name-4. These implicit functions are performed such that any associated USE procedures are executed. The terminating function is performed as if a CLOSE statement, without optional phrases, had been executed for the file. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and the moving of the records from the file area for file-name-1 to the file area for file-name-4.

1. INTRODUCTION TO THE REPORT WRITER MODULE

1.1 FUNCTION

The Report Writer module provides the facility for producing reports by specifying the physical appearance of a report rather than requiring specification of the detailed procedures necessary to produce that report.

A hierarchy of levels is used in defining the logical organization of a report. Each report is divided into report groups, which in turn are divided into sequences of items. Such a hierarchical structure permits explicit reference to a report group with implicit reference to other levels in the hierarchy. A report group contains one or more items to be presented on one or more lines.

1.2 LANGUAGE CONCEPTS

1.2.1 LINE-COUNTER

The reserved word LINE-COUNTER is a name for a special register that is generated for each report description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 0 through 999999. The usage is defined by the implementor. The value in LINE-COUNTER is maintained by the Report Writer Control System, and is used to determine the vertical positioning of a report. The value in LINE-COUNTER may be accessed by Procedure Division statements; however, only the RWCS may change the value of LINE-COUNTER.

1.2.2 PAGE-COUNTER

The reserved word PAGE-COUNTER is a name for a special register that is generated for each report description entry in the Report Section of the Data Division. The implicit description is that of an unsigned integer that must be capable of representing a range of values from 1 through 999999. The usage is defined by the implementor. The value in PAGE-COUNTER is maintained by the Report Writer Control System and is used by the program to number the pages of a report. The value in PAGE-COUNTER may be altered by Procedure Division statements.

1.2.3 SUBSCRIPTING

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

1.3 RELATIONSHIP WITH SEQUENTIAL I-O MODULE

A report file is a sequential file as described in the Sequential I-O module and is subject to the restrictions in the following paragraph.

An OPEN statement, specifying either the OUTPUT or EXTEND phrase, must have been executed prior to the execution of the INITIATE statement, and a CLOSE, without the REEL or UNIT phrase, must be executed for this file subsequent to the execution of the TERMINATE statement. No other input-output statement may be executed for this file.

2. DATA DIVISION IN THE REPORT WRITER MODULE

2.1 FILE SECTION

A REPORT clause is required in the FD entry to list the names of the reports to be produced.

2.2 REPORT SECTION

In the Report Section the description of each report must begin with a report description entry (RD entry) and be followed by the entries that describe the report groups within the report.

2.2.1 Report Description Entry

In addition to naming the report, the RD entry defines the format of each page of the report by specifying the vertical boundaries of the region within which each type of report group may be printed. The RD entry also specifies the control data items. When the report is produced, changes in the values of the control data items cause the detail information of the report to be processed in groups called control groups.

Each report named in the REPORTS clause of an RD entry in the File Section must be the subject of an RD entry in the Report Section. Furthermore each report in the Report Section must be named in one and only one FD entry.

2.2.2 Report Group Description Entry

The report groups that will comprise the report are described following the RD entry. The description of each report group begins with a report group description entry; that is an entry that has a 01 level-number and a TYPE clause. Subordinate to the report group description entry, there may appear group and elementary entries that further describe the characteristics of the report group.

2.3 THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON

2.3.1 Function

The file description furnishes information concerning the physical structure, identification and report names pertaining to a given report file.

2.3.2 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  }
    [; CODE-SET IS alphabet-name
     ;{REPORT IS  } report-name-1 [, report-name-2] ... .
      {REPORTS ARE}
```

2.3.3 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.

(3) The file referenced by file-name must be defined, implicitly or explicitly in the FILE-CONTROL paragraph of the Environment Division, as a sequential file. Further, each report named in the REPORT clause must be the subject of a report description entry in the Report Section.

(4) No record description entries are permitted for file-name and no input-output statements, except the OPEN with either the OUTPUT or EXTEND phrase and the CLOSE without either the REEL or UNIT phrase, may be executed for this file.

2.4 THE REPORT DESCRIPTION - COMPLETE ENTRY SKELETON

2.4.1 Function

The report description entry names a report, specifies any identifying characters to be appended to each print line, and describes the physical structure and organization of that report.

2.4.2 General Format

```
RD  report-name

    [; CODE literal-1 ]

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

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

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

         [, FOOTING integer-5 ] .
```

2.4.3 Syntax Rules

(1) The report-name must appear in one and only one REPORT clause.

(2) The order of appearance of the clauses following the report-name is immaterial.

(3) Report-name is the highest permissible qualifier that may be specified for LINE-COUNTER, PAGE-COUNTER and all data-names defined within the Report Section.

(4) One or more report group description entries must follow the report description entry.

2.4.4 PAGE-COUNTER Rules

(1) PAGE-COUNTER is the reserved word to reference a special register that is automatically created for each report specified in the Report Section. (See page VIII-1, PAGE-COUNTER.)

(2) In the Report Section, a reference to PAGE-COUNTER can only appear in a SOURCE clause. Outside of the Report Section, PAGE-COUNTER may be used in any context in which a data-name of integral value can appear.

(3) If more than one PAGE-COUNTER exists in a program, PAGE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

In the Report Section an unqualified reference to PAGE-COUNTER is implicitly qualified by the name of the report in which the reference is made. Whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be explicitly qualified by that report-name. (See page II-1, Name Characteristics, for constraints that apply when Report Writer is associated with Nucleus, Level 1.)

(4) Execution of the INITIATE statement causes the Report Writer Control System to set the PAGE-COUNTER of the referenced report to one (1).

(5) PAGE-COUNTER is automatically incremented by one (1) each time the Report Writer Control System executes a page advance.

(6) PAGE-COUNTER may be altered by Procedure Division statements.

2.4.5 LINE-COUNTER Rules

(1) LINE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the Report Section. (See page VIII-1, LINE-COUNTER.)

(2) In the Report Section a reference to LINE-COUNTER can only appear in a SOURCE clause. Outside the Report Section, LINE-COUNTER may be used in any context in which a data-name of integral value may appear. However, only the Report Writer Control System can change the contents of LINE-COUNTER.

(3) If more than one LINE-COUNTER exists in a program, LINE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division.

In the Report Section an unqualified reference to LINE-COUNTER is implicitly qualified by the name of the report in which the reference is made. Whenever the LINE-COUNTER of a different report is referenced, LINE-COUNTER must be explicitly qualified by that report-name. (See page II-1, Name Characteristics, for constraints that apply when Report Writer is associated with Nucleus, Level 1.)

(4) Execution of an INITIATE statement causes the Report Writer Control System to set the LINE-COUNTER of the referenced report to zero (0). The Report Writer Control System also automatically resets LINE-COUNTER to zero each time it executes a page advance.

(5) The value of LINE-COUNTER is not affected by the processing of non-printable report groups nor by the processing of a printable report group whose printing is suppressed by means of the SUPPRESS statement.

(6) At the time each print line is presented, the value of LINE-COUNTER represents the line number on which the print line is presented. The value of LINE-COUNTER after the presentation of a report group is governed by the presentation rules for the report group. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

2.5 THE REPORT GROUP DESCRIPTION - COMPLETE SKELETON

2.5.1 Function

The report group description entry specifies the characteristics of a report group and of the individual items within a report group.

2.5.2 General Format

Format 1

```
01 [data-name-1]
   [; LINE NUMBER IS {integer-1   ON NEXT PAGE}]
                     {PLUS integer-2           }
   [; NEXT GROUP IS {integer-3  }]
                    {PLUS integer-4}
                    {NEXT PAGE     }
   ; TYPE IS {{REPORT HEADING}                                    }
             {{RH           }                                     }
             {{PAGE HEADING }                                     }
             {{PH           }                                     }
             {{CONTROL HEADING} {data-name-2}                     }
             {{CH             } {FINAL      }                     }
             {{DETAIL }                                           }
             {{DE     }                                           }
             {{CONTROL FOOTING} {data-name-3}                     }
             {{CF             } {FINAL      }                     }
             {{PAGE FOOTING }                                     }
             {{PF           }                                     }
             {{REPORT FOOTING}                                    }
             {{RF            }                                    }
   [; [USAGE IS] DISPLAY ] .
```

Format 2

```
level-number [data-name-1]
   [; LINE NUMBER IS {integer-1 [ON NEXT PAGE]}]
                     {PLUS integer-2          }
   [; [USAGE IS] DISPLAY ] .
```

Format 3

```
level-number [ data-name-1]
   [; BLANK WHEN ZERO]
   [; GROUP INDICATE]
   [; {JUSTIFIED} RIGHT]
      {JUST     }
   [; LINE NUMBER IS {integer-1 [ON NEXT PAGE]}]
                     {PLUS integer-2          }
   [; COLUMN NUMBER IS integer-3]
    ; {PICTURE} IS character-string
      {PIC    }
  {; SOURCE IS identifier-1                                    }
  {; VALUE IS literal                                          }
  {; SUM identifier-2 [, identifier-3] ...                     }
  {     [UPON data-name-2 [, data-name-3] ... ]} ...           }
  {     [RESET ON {data-name-4}]               }               }
  {               {FINAL     }                                 }
   [; [USAGE IS] DISPLAY ] .
```

2.5.3 Syntax Rules

(1) The report group description entry can appear only in the Report Section.

(2) Except for the data-name clause, which when present must immediately follow the level-number, the clauses may be written in any sequence.

(3) In Format 2 the level-number may be any integer from 02 to 48 inclusive. In Format 3 the level-number may be any integer from 02 to 49 inclusive.

(4) The description of a report group may consist of one, two or three hierarchic levels:

a. The first entry that describes a report group must be a Format 1 entry.

b. Both Format 2 and Format 3 entries may be immediately subordinate to a Format 1 entry.

c. At least one Format 3 entry must be immediately subordinate to a Format 2 entry.

d. Format 3 entries must be elementary.

(5) In a Format 1 entry, data-name-1 is required only when:

a. A DETAIL report group is referenced by a GENERATE statement,

b. A DETAIL report group is referenced by the UPON phrase of a SUM clause,

c. A report group is referenced in a USE BEFORE REPORTING sentence,

d. The name of a CONTROL FOOTING report group is used to qualify a reference to a sum counter.

(6) A Format 2 entry must contain at least one optional clause.

(7) In a Format 2 entry, data-name-1 is optional. If present it may be used only to qualify a sum counter reference.

(8) In the Report Section, the USAGE clause is used only to declare the usage of printable items.

a. If the USAGE clause appears in a Format 3 entry, that entry must define a printable item.

b. If the USAGE clause appears in a Format 1 or Format 2 entry, at least one subordinate entry must define a printable item.

(9) An entry that contains a LINE NUMBER clause must not have a subordinate entry that also contains a LINE NUMBER clause.

(10) In Format 3:

a. A GROUP INDICATE clause may appear only in a TYPE DETAIL report group.

b. A SUM clause may appear only in a TYPE CONTROL FOOTING report group.

c. An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.

d. Data-name-1 is optional but may be specified in any entry. Data-name-1, however, may be referenced only if the entry defines a sum counter.

e. A LINE NUMBER clause must not be the only clause specified.

f. An entry that contains a VALUE clause must also have a COLUMN NUMBER clause.

(11) The following table shows all permissible clause combinations for a Format 3 entry. The table is read from left to right along the selected row.

An 'M' indicates that the presence of the clause is mandatory.

A 'P' indicates that the presence of the clause is permitted, but not required.

A blank indicates that the clause is not permitted.

CLAUSES

PIC	COLUMN	SOURCE	SUM	VALUE	JUST	BLANK WHEN ZERO	GROUP INDICATE	USAGE	LINE
M			M						P
M	M		M			P		P	P
M	P	M			P		P	P	P
M	P	M				P	P	P	P
M	M			M	P		P	P	P

Permissible Clause Combinations in Format 3 Entries

2.5.4 General Rules

(1) Format 1 is the report group entry. The report group is defined by the contents of this entry and all of its subordinate entries.

2.5.5 Presentation Rules Tables

2.5.5.1 Description

The tables and rules on the following pages specify:

(1) The permissible combinations of LINE NUMBER and NEXT GROUP clauses for each type of report group,

(2) The requirements that are placed on the use of these clauses, and

(3) The interpretation that the RWCS gives to these clauses.

2.5.5.2 Organization

There is an individual presentation rules table for each of the following types of report groups: REPORT HEADING, PAGE HEADING, PAGE FOOTING, REPORT FOOTING. In addition, DETAIL report groups, CONTROL HEADING report groups, and CONTROL FOOTING report groups are treated jointly in the Body Group Presentation Rules Table. (See paragraph 2.5.5.8, The Body Group Presentation Rules Table, beginning on page VIII-15.)

Columns 1 and 2 of a presentation rules table list all of the permissible combinations of LINE NUMBER and NEXT GROUP clauses for the designated report group TYPE. Consequently, for the purpose of identifying the set of presen-

tation rules that apply to a particular combination of LINE NUMBER and NEXT GROUP clauses, a presentation rules table is read from left to right, along the selected row.

The applicable rules columns of a presentation rules table are partitioned into two parts. The first part specifies the rules that apply if the report description contains a PAGE clause, and the second part specifies the rules that apply if the PAGE clause is omitted. The purpose of the rules named in the applicable rules columns is discussed below:

(1) Upper Limit Rules and Lower Limit Rules. These rules specify the vertical subdivisions of the page within which the specified report group may be presented.

In the absence of a PAGE clause the printed report is not considered to be partitioned into vertical subdivisions. Consequently, within the tables no upper limit rule and lower limit rule is specified for a report description in which the PAGE clause is omitted.

(2) Fit Test Rules. The fit test rules are applicable only to body groups, and hence fit test rules are specified only within the Body Group Presentation Rules Table. At object time the RWCS applies the fit test rules to determine whether the designated body group can be presented on the page to which the report is currently positioned.

However, even for body groups there are no fit test rules when the PAGE clause is omitted from the report description entry.

(3) First Print Line Position Rules. The first print line position rules specify where on the report medium the RWCS shall present the first print line of the given report group.

The presentation rule tables do not specify where on the report medium the RWCS shall present the second and subsequent print lines (if any) of a report group. Certain general rules determine where the second and subsequent print lines of a report group shall be presented. Refer to the LINE NUMBER clause general rules for this information. (See page VIII-33, The LINE NUMBER Clause.)

(4) Next Group Rules. The next group rules relate to the proper use of the NEXT GROUP clause.

(5) Final LINE-COUNTER Setting Rules. The terminal values that the RWCS places in LINE-COUNTER after presenting report groups are specified by the final LINE-COUNTER setting rules.

2.5.5.3 LINE NUMBER Clause Notation

Column 1 of the presentation rules table uses a shorthand notation to describe the sequence of LINE NUMBER clauses that may appear in the description of a report group. The meaning of the abbreviations used in column 1 is as follows:

(1) The letter 'A' represents one or more absolute LINE NUMBER clauses, none of which have the NEXT PAGE phrase, that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry.

(2) The letter 'R' represents one or more relative LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry.

(3) The letters 'NP' represent one or more absolute LINE NUMBER clauses that appear in consecutive order within the sequence of LINE NUMBER clauses in the report group description entry, with the phrase NEXT PAGE appearing in the first, and only in the first, LINE NUMBER clause.

When two abbreviations appear together, they refer to a sequence of LINE NUMBER clauses that consists of the two specified consecutive sequences. For example 'AR' refers to a report group description entry within which the 'A' sequence (defined in rule 1 above) is immediately followed by the 'R' sequence (defined in rule 2 above).

2.5.5.4 LINE NUMBER Clause Sequence Substitutions

Where 'AR' is shown to be a permissible sequence in the presentation rules table, 'A' is also permissible and the same presentation rules are applicable.

When 'NP R' is shown to be a permissible sequence in the presentation rules table, 'NP' is also permissible and the same presentation rules are applicable.

2.5.5.5 Saved Next Group Integer Description

Saved next group integer is a data item that is addressable only by the RWCS. When an absolute NEXT GROUP clause specifies a vertical positioning value which cannot be accommodated on the current page, the RWCS stores that value in saved next group integer. After page advance processing, the RWCS positions the next body group using the value stored in saved next group integer.

2.5.5.6 Table 1 - REPORT HEADING Group Presentation Rules Table

The table on page VIII-12 points to the appropriate presentation rules for all permissible combinations of LINE-NUMBER and NEXT GROUP clauses in a REPORT HEADING report group.

2.5.5.6.1 Table 1 Presentation Rules

(1) Upper Limit Rule. The first line number on which the REPORT HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

(2) Lower Limit Rules

a. The last line number on which the REPORT HEADING report group can be presented is the line number that is obtained by subtracting 1 from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

Table 1 - REPORT HEADING Group Presentation Rules Table

**		Applicable Rules ***						
		If the PAGE clause is specified.					If the PAGE clause is omitted.	
Sequence of LINE NUMBER clauses*	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R	Absolute	1	2a	3a	4a	5a	Illegal Combination+	
A R	Relative	1	2a	3a	4b	5b	Illegal Combination+	
A R	NEXT PAGE	1	2b	3a	4c	5c	Illegal Combination+	
A R		1	2a	3a		5d	Illegal Combination+	
R	Absolute	1	2a	3b	4a	5a	Illegal Combination++	
R	Relative	1	2a	3b	4b	5b	3d	5b
R	NEXT PAGE	1	2b	3b	4c	5c	Illegal Combination++	
R		1	2a	3b		5d	3d	5d
				3c		5e	3c	5e

* See page VIII-10, LINE NUMBER Clause Notation, for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

\+ See page VIII-33, The LINE NUMBER Clause.

++ See page VIII-35, The NEXT GROUP Clause.

b. The last line number on which the REPORT HEADING report group can be presented is the line number specified by integer-1 of the PAGE clause.

(3) First Print Line Position Rules

a. The first print line of the REPORT HEADING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

b. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the integer of the first LINE NUMBER clause and the value obtained by subtracting 1 from the value of integer-2 of the HEADING phrase of the PAGE clause.

c. The REPORT HEADING report group is not presented.

d. The first print line of the REPORT HEADING report group is presented on the line number obtained by adding the contents of its LINE-COUNTER (in this case, zero) to the integer of the first LINE NUMBER clause.

(4) Next Group Rules

a. The NEXT GROUP integer must be greater than the line number on which the final print line of the REPORT HEADING report group is presented. In addition, the NEXT GROUP integer must be less than the line number specified by the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

b. The sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group is presented must be less than the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

c. NEXT GROUP NEXT PAGE signifies that the REPORT HEADING report group is to be presented entirely by itself on the first page of the report. The RWCS processes no other report group while positioned to the first page of the report.

(5) Final LINE-COUNTER Setting Rules

a. After the REPORT HEADING report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

b. After the REPORT HEADING report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the REPORT HEADING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

c. After the REPORT HEADING report group is presented, the RWCS places zero into LINE-COUNTER as the final LINE-COUNTER setting.

d. After the REPORT HEADING report group is presented, the final LINE-COUNTER setting is the line number on which the final print line of the REPORT HEADING report group was presented.

e. LINE-COUNTER is unaffected by the processing of a non-printable report group.

2.5.5.7 Table 2 - PAGE HEADING Group Presentation Rules Table

The following table points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE HEADING report group.

Table 2 - PAGE HEADING Group Presentation Rules Table

**		Applicable Rules ***				
		If the PAGE clause is specified ****				
Sequence of LINE NUMBER clauses*	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting
A R		1	2	3a		4a
R		1	2	3b		4a
				3c		4b

* See page VIII-10, LINE NUMBER Clause Notation, for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

**** If the PAGE clause is omitted from the report description entry, then a PAGE HEADING report group may not be defined. (See page VIII-45, The TYPE Clause.)

2.5.5.7.1 Table 2 Presentation Rules

(1) Upper Limit Rules. If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the first line number on which the PAGE HEADING report group can be presented is one greater than the final LINE-COUNTER setting established by the REPORT HEADING.

Otherwise the first line number on which the PAGE HEADING report group can be presented is the line number specified by the HEADING phrase of the PAGE clause.

(2) Lower Limit Rule. The last line number on which the PAGE HEADING report group can be presented is the line number that is obtained by subtracting one (1) from the value of integer-3 of the FIRST DETAIL phrase of the PAGE clause.

(3) First Print Line Position Rules.

a. The first print line of the PAGE HEADING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

b. If a REPORT HEADING report group has been presented on the page on which the PAGE HEADING report group is to be presented, then the sum of the final LINE-COUNTER setting established by the REPORT HEADING report group and the integer of the first LINE NUMBER clause of the PAGE HEADING report group defines the line number on which the first print line of the PAGE HEADING report group is presented.

Otherwise the sum of the integer of the first LINE NUMBER clause of the PAGE HEADING report group and the value obtained by subtracting one (1) from the value of integer-2 of the HEADING phrase of the PAGE clause defines the line number on which the first print line of the PAGE HEADING report group is presented.

c. The PAGE HEADING report group is not presented.

(4) Final LINE-COUNTER Setting Rules

a. The final LINE-COUNTER setting is the line number on which the final print line of the PAGE HEADING report group was presented.

b. LINE-COUNTER is unaffected by the processing of a non-printable report group.

2.5.5.8 Table 3 - Body Group Presentation Rules Table

The table on page VIII-16 points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in CONTROL HEADING, DETAIL and CONTROL FOOTING report groups.

2.5.5.8.1 Table 3 Presentation Rules

(1) Upper Limit Rule. The first line number on which a body group can be presented is the line number specified by the FIRST DETAIL phrase of the PAGE clause.

(2) Lower Limit Rules. The last line number on which a CONTROL HEADING report group or DETAIL report group can be presented is the line number specified by the LAST DETAIL phrase of the PAGE clause.

The last line number on which a CONTROL FOOTING report group can be presented is the line number specified by the FOOTING phrase of the PAGE clause.

(3) Fit Test Rules.

a. If the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, then the body group shall be presented on the page to which the report is currently positioned.

Otherwise the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS determines

Table 3 - Body Group Presentation Rules Table

**		Applicable Rules ***							
		If the PAGE clause is specified.						If the PAGE clause is omitted.	
Sequence of LINE NUMBER clauses*	NEXT GROUP clause	Upper Limit	Lower Limit	Fit Test	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R	Absolute	1	2	3a	4a	5	6a	Illegal Combination +	
A R	Relative	1	2	3a	4a		6b	Illegal Combination +	
A R	NEXT PAGE	1	2	3a	4a		6c	Illegal Combination +	
A R		1	2	3a	4a		6d	Illegal Combination +	
R	Absolute	1	2	3b	4b	5	6a	Illegal Combination ++	
R	Relative	1	2	3b	4b		6b	4d	6f
R	NEXT PAGE	1	2	3b	4b		6c	Illegal Combination ++	
R		1	2	3b	4b		6d	4d	6d
NP R	Absolute	1	2	3c	4a	5	6a	Illegal Combination +	
NP R	Relative	1	2	3c	4a		6b	Illegal Combination +	
NP R	NEXT PAGE	1	2	3c	4a		6c	Illegal Combination +	
NP R		1	2	3c	4a		6d	Illegal Combination +	
					4c		6e	4c	6e

* See page VIII-10, LINE NUMBER Clause Notation, for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

\+ See page VIII-33, The LINE NUMBER Clause.

++ See page VIII-35, The NEXT GROUP Clause.

whether the saved next group integer location was set when the final body group was presented on the preceding page. (See final LINE-COUNTER setting rule 6a on page VIII-18.) If the saved next group integer was not so set, the body group shall be presented on the page to which the report is currently positioned. If the saved next group integer was so set, the RWCS moves the saved next group integer into LINE-COUNTER, resets the saved next group integer to zero, and reapplies fit test rule 3a.

b. If a body group has been presented on the page to which the report is currently positioned, the RWCS computes a trial sum in a work location. The trial sum is computed by adding the contents of LINE-COUNTER to the integers of all LINE NUMBER clauses of the report group. If the trial sum is not greater than the body group's lower limit integer, then the report group is presented on the current page. If the trial sum exceeds the body group's lower limit integer, then the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS reapplies fit test rule 3b.

If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. (See final LINE-COUNTER setting rule 6a on page VIII-18.) If the saved next group integer was not so set, the body group shall be presented on the page to which the report is currently positioned. If the saved next group integer was so set, the RWCS moves the saved next group integer into LINE-COUNTER, resets the saved next group integer to zero, and computes a trial sum in a work location.

The trial sum is computed by adding the contents of LINE-COUNTER to the integer one (1) and the integers of all but the first LINE NUMBER clause of the body group. If the trial sum is not greater than the body group's lower limit integer, then the body group is presented on the current page. If the trial sum exceeds the body group's lower limit integer, then the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

c. If a body group has been presented on the page to which the report is currently positioned, the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS reapplies fit test rule 3c.

If no body group has yet been presented on the page to which the report is currently positioned, the RWCS determines whether the saved next group integer location was set when the final body group was presented on the preceding page. (See final LINE-COUNTER setting rule 6a on page VIII-18.) If the saved next group integer was not so set, the body group shall be presented on the page to which the report is currently positioned. If the saved next group integer was so set, the RWCS moves the saved next group integer into LINE-COUNTER and resets the saved next group integer to zero. If then the value in LINE-COUNTER is less than the integer of the first absolute LINE NUMBER clause, the body group shall be presented on the page to which the report is currently positioned. Otherwise the RWCS executes page advance processing. After the PAGE HEADING report group (if defined) has been processed, the RWCS presents the body group on that page.

(4) First Print Line Position Rules

a. The first print line of the body group is presented on the line number specified by the integer of its LINE NUMBER clause.

b. If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if no body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line immediately following the line indicated by the value contained in LINE-COUNTER.

If the value in LINE-COUNTER is equal to or greater than the line number specified by the FIRST DETAIL phrase of the PAGE clause, and if a body group has previously been presented on the page to which the report is currently positioned, then the first print line of the current body group is presented on the line that is obtained by adding the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause of the current body group.

If the value in LINE-COUNTER is less than the line number specified by the FIRST DETAIL phrase of the PAGE clause, then the first print line of the body group is presented on the line specified by the FIRST DETAIL phrase.

c. The body group is not presented.

d. The sum of the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause defines the line number on which the first print line is presented.

(5) Next Group Rule. The integer of the absolute NEXT GROUP clause must specify a line number that is not less than that specified in the FIRST DETAIL phrase of the PAGE clause, and that is not greater than that specified in the FOOTING phrase of the PAGE clause.

(6) Final LINE-COUNTER Setting Rules

a. If the body group that has just been presented is a CONTROL FOOTING report group and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases the RWCS makes a comparison of the line number on which the final print line of the body group was presented and the integer of the NEXT GROUP clause. If the former is less than the latter, then the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting. If the former is equal to or greater than the latter, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting; in addition the RWCS places the NEXT GROUP integer into the saved next group integer location.

b. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final

LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases the RWCS computes a trial sum in a work location. The trial sum is computed by adding the integer of the NEXT GROUP clause to the line number on which the final print line of the body group was presented. If the sum is less than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places that sum into LINE-COUNTER as the final LINE-COUNTER setting. If the sum is equal to or greater than the line number specified by the FOOTING phrase of the PAGE clause, then the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

c. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases the RWCS places the line number specified by the FOOTING phrase of the PAGE clause into LINE-COUNTER as the final LINE-COUNTER setting.

d. The final LINE-COUNTER setting is the line number on which the final print line of the body group was presented.

e. LINE-COUNTER is unaffected by the processing of a non-printable body group.

f. If the body group that has just been presented is a CONTROL FOOTING report group, and if the CONTROL FOOTING report group is not associated with the highest level at which the RWCS detected a control break, then the final LINE-COUNTER setting is the line number on which the final print line of the CONTROL FOOTING report group was presented.

For all other cases the RWCS places the sum of the line number on which the final print line was presented and the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

2.5.5.9 Table 4 - PAGE FOOTING Presentation Rules

The following table points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a PAGE FOOTING report group.

Table 4 - PAGE FOOTING Presentation Rules Table

**		Applicable Rules *** If the PAGE clause is specified ****				
Sequence of LINE NUMBER clauses*	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting
A R	Absolute	1	2	3a	4a	5a
A R	Relative	1	2	3a	4b	5b
A R		1	2	3a		5c
				3b		5d

* See page VIII-10, LINE NUMBER Clause Notation, for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

**** If the PAGE clause is omitted from the report description entry, then a PAGE FOOTING report group may not be defined. (See page VIII-45, The TYPE Clause.)

2.5.5.9.1 Table 4 Presentation Rules

(1) Upper Limit Rule. The first line number on which the PAGE FOOTING report group can be presented, is the line number obtained by adding one to the value of integer-5 of the FOOTING phrase of the PAGE clause.

(2) Lower Limit Rule. The last line number on which the PAGE FOOTING report group can be presented is the line number specified by integer-1 of the PAGE clause.

(3) First Print Line Position Rules

a. The first print line of the PAGE FOOTING report group is presented on the line specified by the integer of its LINE NUMBER clause.

b. The PAGE FOOTING report group is not presented.

(4) NEXT GROUP Rules

a. The NEXT GROUP integer must be greater than the line number on which the final print line of the PAGE FOOTING report group is presented. In addition, the NEXT GROUP integer must not be greater than the line number specified by integer-1 of the PAGE clause.

b. The sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group is presented must not be greater than the line number specified by integer-1 of the PAGE clause.

(5) Final LINE-COUNTER Setting Rules

a. After the PAGE FOOTING report group is presented, the RWCS places the NEXT GROUP integer into LINE-COUNTER as the final LINE-COUNTER setting.

b. After the PAGE FOOTING report group is presented, the RWCS places the sum of the NEXT GROUP integer and the line number on which the final print line of the PAGE FOOTING report group was presented into LINE-COUNTER as the final LINE-COUNTER setting.

c. After the PAGE FOOTING report group is presented the final LINE-COUNTER setting is the line number on which the final print line of the PAGE FOOTING report group was presented.

d. LINE-COUNTER is unaffected by the processing of a non-printable report group.

2.5.5.10 Table 5 - REPORT FOOTING Presentation Rules Table

The table on page VIII-22 points to the appropriate presentation rules for all permissible combinations of LINE NUMBER and NEXT GROUP clauses in a REPORT FOOTING report group.

2.5.5.10.1 Table 5 Presentation Rules

(1) Upper Limit Rules

a. If a PAGE FOOTING report group has been presented on the page to which the report is currently positioned, then the first line number on which the REPORT FOOTING report group can be presented is one greater than the final LINE-COUNTER setting established by the PAGE FOOTING report group.

Otherwise the first line number on which the REPORT FOOTING report group can be presented is the line number obtained by adding one and the value of integer-5 of the PAGE clause.

b. The first line number on which the REPORT FOOTING report group can be presented, is the line number specified by the HEADING phrase of the PAGE clause.

(2) Lower Limit Rule. The last line number on which the REPORT FOOTING report group can be presented is the line number specified by integer-1 of the PAGE clause.

Table 5 - REPORT FOOTING Presentation Rules Table

**		Applicable Rules ***						
		If the PAGE clause is specified.					If the PAGE clause is omitted.	
Sequence of LINE NUMBER clauses*	NEXT GROUP clause	Upper Limit	Lower Limit	First Print Line Position	Next Group	Final LINE-COUNTER Setting	First Print Line Position	Final LINE-COUNTER Setting
A R		1a	2	3a		4a	Illegal Combination+	
R		1a	2	3b		4a	3d	4a
NP R		1b	2	3c		4a	Illegal Combination+	
				3e		4b	3e	4b

* See page VIII-10, LINE NUMBER Clause Notation, for a description of the abbreviations used in column 1.

** A blank entry in column 1 or column 2 indicates that the named clause is totally absent from the report group description entry.

*** A blank entry in an applicable rules column indicates the absence of the named rule for the given combination of LINE NUMBER and NEXT GROUP clauses.

\+ See page VIII-33, The LINE NUMBER Clause.

(3) First Print Line Position Rules

a. The first print line of the REPORT FOOTING report group is presented on the line specified by the integer of its LINE NUMBER clause.

b. If a PAGE FOOTING report group has been presented on the page to which the report is currently positioned, then the sum of the final LINE-COUNTER setting established by the PAGE FOOTING report group and the integer of the first LINE NUMBER clause of the REPORT FOOTING report group defines the line number on which the first print line of the REPORT FOOTING report group is presented. Otherwise the sum of the integer of the first LINE NUMBER clause of the REPORT FOOTING report group, and the line number specified by the value of integer-5 of the FOOTING phrase of the PAGE clause defines the line number on which the first print line of the REPORT FOOTING report group is presented.

c. The NEXT PAGE phrase in the first absolute LINE NUMBER clause directs that the REPORT FOOTING report group is presented on a page on which no other report group has been presented. The first print line of the REPORT FOOTING report group is presented on the line number specified by the integer of its LINE NUMBER clause.

d. The sum of the contents of LINE-COUNTER and the integer of the first LINE NUMBER clause defines the line number on which the first print line is presented.

e. The REPORT FOOTING report group is not presented.

(4) Final LINE-COUNTER Setting Rules.

a. The final LINE-COUNTER setting is the line number on which the final print line of the REPORT FOOTING report group is presented.

b. LINE-COUNTER is unaffected by the processing of a non-printable report group.

2.6 THE BLOCK CONTAINS CLAUSE

2.6.1 Function

The BLOCK CONTAINS clause specifies the size of a physical record.

2.6.2 General Format

$$\underline{\text{BLOCK}}\ \text{CONTAINS}\ \left[\text{integer-1}\ \underline{\text{TO}}\right]\ \text{integer-2}\ \left\{\begin{matrix}\text{RECORDS}\\ \text{CHARACTERS}\end{matrix}\right\}$$

2.6.3 General Rules

(1) This clause is required except when:

a. A physical record contains one and only one complete logical record.

b. The hardware device assigned to the file has one and only one physical record size.

c. The hardware device assigned to the file has more than one physical record size but the implementor has designated one as standard. In this case, the absence of this clause denotes the standard physical record size.

(2) The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exists, in which case the RECORDS phrase must not be used

a. In mass storage files, where logical records may extend across physical records.

b. The physical record contains padding (area not contained in a logical record).

c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

(3) When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

(4) If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

(5) If logical records of differing size are grouped into one physical record, the technique for determining the size of each logical record is specified by the implementor.

2.7 THE CODE CLAUSE

2.7.1 Function

The CODE clause specifies a two character literal that identifies each print line as belonging to a specific report.

2.7.2 General Format

CODE literal-1

2.7.3 Syntax Rules

(1) Literal-1 is a two character nonnumeric literal.

(2) If the CODE clause is specified for any report in a file, then it must be specified for all reports in the same file.

2.7.4 General Rules

(1) When the CODE clause is specified, literal-1 is automatically placed in the first two character positions of each Report Writer logical record.

(2) The positions occupied by literal-1 are not included in the description of the print line, but are included in the logical record size.

2.8 THE CODE-SET CLAUSE

2.8.1 Function

The CODE-SET clause specifies the character code set used to represent data on the external media.

2.8.2 General Format

CODE-SET IS alphabet-name

2.8.3 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-mass storage files.

2.8.4 General Rules

(1) If the CODE-SET clause is specified, alphabet-name specifies the character code convention used to represent data on the external media. It also specifies the algorithm for converting the character codes on the external media from the native character codes. This code conversion occurs during the execution of an output operation. (See page II-8, The SPECIAL-NAMES Paragraph.)

(2) If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media.

2.9 THE COLUMN NUMBER CLAUSE

2.9.1 Function

The COLUMN NUMBER clause identifies a printable item and specifies the column number position of the item on a print line.

2.9.2 General Format

COLUMN NUMBER IS integer-1

2.9.3 Syntax Rules

(1) The COLUMN NUMBER clause can only be specified at the elementary level within a report group. The COLUMN NUMBER clause, if present, must appear in or be subordinate to an entry that contains a LINE NUMBER clause.

(2) Within a given print line, the printable items must be defined in ascending column number order such that each character defined occupies a unique position.

2.9.4 General Rules

(1) The COLUMN NUMBER clause indicates that the object of a SOURCE clause or the object of a VALUE clause or the sum counter defined by a SUM clause is to be presented on the print line. The absence of a COLUMN NUMBER clause indicates that the entry is not to be presented on a print line.

(2) Integer-1 specifies the column number of the leftmost character position of the printable item.

(3) The Report Writer Control System supplies space character for all positions of a print line that are not occupied by printable items.

(4) The first position of the print line is considered to be column number 1.

2.10 THE CONTROL CLAUSE

2.10.1 Function

The CONTROL clause establishes the levels of the control hierarchy for the report.

2.10.2 General Format

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

2.10.3 Syntax Rules

(1) Data-name-1 and data-name-2 must not be defined in the Report Section. Data-name-1 and data-name-2 may be qualified but must not be subscripted or indexed.

(2) Each data-name must identify a different data item.

(3) Data-name-1, data-name-2, ..., must not have subordinate to it a data item whose size is variable as defined in the OCCURS clause. (See page III-2, The OCCURS Clause.)

2.10.4 General Rules

(1) The data-names and the word FINAL specify the levels of the control hierarchy. FINAL, if specified, is the highest control, data-name-1 is the major control, data-name-2 is an intermediate control, etc. The last data-name specified is the minor control.

(2) The execution of the chronologically first GENERATE statement for a given report causes the RWCS to save the values of all control data items associated with that report. On subsequent executions of all GENERATE statements for that report, control data items are tested by the RWCS for a change of value. A change of value in any control data item causes a control break to occur. The control break is associated with the highest level for which a change of value is noted. (See page VIII-51, The GENERATE Statement.)

(3) The Report Writer Control System tests for a control break by comparing the contents of each control data item with the prior contents saved from the execution of the previous GENERATE statement for the same report. The RWCS applies the inequality relation test described on page II-41, The Relation Condition, as follows:

a. If the control data item is a numeric data item, the relation test is for the comparison of two numeric operands.

b. If the control data item is an index data item, the relation test is for the comparison of two index data items.

c. If the control data item is a data item other than as described in paragraph 3a and 3b, the relation test is for the comparison of two nonnumeric operands.

See page II-6, PROGRAM COLLATING SEQUENCE clause.

(4) FINAL is used when the most inclusive control group in the report is not associated with a control data-name.

2.11 THE DATA-NAME CLAUSE

2.11.1 Function

A data-name specifies the name of the data being described.

2.11.2 General Format

data-name

2.11.3 Syntax Rules

(1) In the Report Section a data-name need not appear in a data description entry and FILLER must not be used.

2.11.4 General Rules

(1) In the Report Section, data-name must be given in the following cases:

a. When the data-name represents a report group to be referred to by a GENERATE or a USE statement in the Procedure Division.

b. When reference is to be made to the sum counter in the Procedure Division or Report Section.

c. When a DETAIL report group is referenced in the UPON phrase of the SUM clause.

d. When the data-name is required to provide sum counter qualification.

2.12 THE GROUP INDICATE CLAUSE

2.12.1 Function

The GROUP INDICATE clause specifies that the associated printable item is presented only on the first occurrence of its report group after a control break or page advance.

2.12.2 General Format

GROUP INDICATE

2.12.3 Syntax Rules

(1) The GROUP INDICATE clause may only appear in a DETAIL report group entry that defines a printable item.

2.12.4 General Rules

(1) If a GROUP INDICATE clause is specified, it causes the SOURCE or VALUE clause to be ignored and spaces supplied, except:

a. On the first presentation of the DETAIL report group in the report, or

b. On the first presentation of the DETAIL report group after every page advance, or

c. On the first presentation of the DETAIL report group after every control break.

(2) If the report description entry specifies neither a PAGE clause nor a CONTROL clause, then a GROUP INDICATE printable item is presented the first time its DETAIL is presented after the INITIATE statement is executed. Thereafter spaces are supplied for indicated items with SOURCE or VALUE clauses.

2.13 THE LABEL RECORDS CLAUSE

2.13.1 Function

The LABEL RECORDS clause specifies whether labels are present.

2.13.2 General Format

$$\underline{\text{LABEL}} \left\{ \begin{matrix} \underline{\text{RECORD}} \text{ IS} \\ \underline{\text{RECORDS}} \text{ ARE} \end{matrix} \right\} \left\{ \begin{matrix} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \end{matrix} \right\}$$

2.13.3 Syntax Rules

(1) This clause is required in every File Description entry.

2.13.4 General Rules

(1) OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

(2) STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the implementor's label specifications.

2.14 THE LINE NUMBER CLAUSE

2.14.1 Function

The LINE NUMBER clause specifies vertical positioning information for its report group.

2.14.2 General Format

$$\underline{\text{LINE}}\ \text{NUMBER IS}\ \left\{\begin{array}{l}\text{integer-1}\ \left[\text{ON}\ \underline{\text{NEXT}}\ \underline{\text{PAGE}}\right]\\ \underline{\text{PLUS}}\ \text{integer-2}\end{array}\right\}$$

2.14.3 Syntax Rules

(1) Integer-1 and integer-2 must not exceed three significant digits in length.

Neither integer-1 nor integer-2 may be specified in such a way as to cause any line of a report group to be presented outside of the vertical subdivision of the page designated for the report group type, as defined by the PAGE clause. (See page VIII-36, The PAGE Clause.)

(2) Within a given report group description entry, an entry that contains a LINE NUMBER clause must not contain a subordinate entry that also contains a LINE NUMBER clause.

(3) Within a given report group description entry, all absolute LINE NUMBER clauses must precede all relative LINE NUMBER clauses.

(4) Within a given report group description entry, successive absolute LINE NUMBER clauses must specify integers that are in ascending order. The integers need not be consecutive.

(5) If the PAGE clause is omitted from a given report group description entry, only relative LINE NUMBER clauses can be specified in any report group description entry within that report.

(6) Within a given report group description entry a NEXT PAGE phrase can appear only once and, if present, must be in the first LINE NUMBER clause in that report group description entry.

A LINE NUMBER clause with the NEXT PAGE phrase can appear only in the description of body groups and in a REPORT FOOTING report group.

(7) Every entry that defines a printable item (see page VIII-27, The COLUMN NUMBER Clause) must either contain a LINE NUMBER clause, or be subordinate to an entry that contains a LINE NUMBER clause.

(8) The first LINE NUMBER clause specified within a PAGE FOOTING report group must be an absolute LINE NUMBER clause.

2.14.4 General Rules

(1) A LINE NUMBER clause must be specified to establish each print line of a report group.

(2) The RWCS effects the vertical positioning specified by a LINE NUMBER clause, before presenting the print line established by that LINE NUMBER clause.

(3) Integer-1 specifies an absolute line number. An absolute line number specifies the line number on which the print line is presented.

(4) Integer-2 specifies a relative line number. If a relative LINE NUMBER clause is not the first LINE NUMBER clause in the report group description entry, then the line number on which its print line is presented is determined by calculating the sum of the line number on which the previous print line of the report group was presented and integer-2 of the relative LINE NUMBER clause.

If a relative LINE NUMBER clause is the first LINE NUMBER clause in the report group description entry, then the line number on which its print line is presented is determined by the rules stated in paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.

(5) The NEXT PAGE phrase specifies that the report group is to be presented beginning on the indicated line number on a new page. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

2.15 THE NEXT GROUP CLAUSE

2.15.1 Function

The NEXT GROUP clause specifies information for vertical positioning of a page following the presentation of the last line of a report group.

2.15.2 General Format

```
NEXT GROUP IS { integer-1          }
              { PLUS integer-2     }
              { NEXT PAGE          }
```

2.15.3 Syntax Rules

(1) A report group entry must not contain a NEXT GROUP clause unless the description of that report group contains at least one LINE NUMBER clause.

(2) Integer-1 and integer-2 must not exceed three significant digits in length.

(3) If the PAGE clause is omitted from the report description entry only a relative NEXT GROUP clause may be specified in any report group description entry within that report.

(4) The NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a PAGE FOOTING report group.

(5) The NEXT GROUP clause must not be specified in a REPORT FOOTING report group or in a PAGE HEADING report group.

2.15.4 General Rules

(1) Any positioning of the page specified by the NEXT GROUP clause takes place after the presentation of the report group in which the clause appears. (See paragraph 2.5.5, Presentation Rules Table, beginning on page VIII-9.)

(2) The vertical positioning information supplied by the NEXT GROUP clause is interpreted by the RWCS along with information from the TYPE and PAGE clauses, and the value in LINE-COUNTER, to determine a new value for LINE-COUNTER. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

(3) The NEXT GROUP clause is ignored by the RWCS when it is specified on a CONTROL FOOTING report group that is at a level other than the highest level at which a control break is detected.

(4) The NEXT GROUP clause of a body group refers to the next body group to be presented, and therefore can affect the location at which the next body group is presented. The NEXT GROUP clause of a REPORT HEADING report group can affect the location at which the PAGE HEADING report group is presented. The NEXT GROUP clause of a PAGE FOOTING report group can affect the location at which the REPORT FOOTING report group is presented. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

2.16 THE PAGE CLAUSE

2.16.1 Function

The PAGE clause defines the length of a page and the vertical subdivisions within which report groups are presented.

2.16.2 General Format

```
PAGE [LIMIT IS  ]  integer-1 [LINE ]
     [LIMITS ARE]            [LINES]

   [, HEADING integer-2 ] [, FIRST DETAIL integer-3 ]
   [, LAST DETAIL integer-4 ] [, FOOTING integer-5 ]
```

2.16.3 Syntax Rules

(1) The HEADING, FIRST DETAIL, LAST DETAIL and FOOTING phrases may be written in any order.

(2) Integer-1 must not exceed three (3) significant digits in length.

(3) Integer-2 must be greater than or equal to one (1).

(4) Integer-3 must be greater than or equal to integer-2.

(5) Integer-4 must be greater than or equal to integer-3.

(6) Integer-5 must be greater than or equal to integer-4.

(7) Integer-1 must be greater than or equal to integer-5.

(8) The following rules indicate the vertical subdivision of the page in which each TYPE of report group may appear when the PAGE clause is specified. (See page VIII-38, Page Regions Table.)

a. A REPORT HEADING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

A REPORT HEADING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

b. A PAGE HEADING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

c. A CONTROL HEADING or DETAIL report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-4, inclusive.

d. A CONTROL FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-5, inclusive.

e. A PAGE FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

f. A REPORT FOOTING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

A REPORT FOOTING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

(9) All report groups must be described such that they can be presented on one page. The RWCS never splits a multi-line report group across page boundaries.

2.16.4 General Rules

(1) The vertical format of a report page is established using the integer values specified in the PAGE clause.

a. Integer-1 defines the size of a report page by specifying the number of lines available on each page.

b. HEADING integer-2 defines the first line number on which a REPORT HEADING or PAGE HEADING report group may be presented.

c. FIRST DETAIL integer-3 defines the first line number on which a body group may be presented. REPORT HEADING and PAGE HEADING report groups may not be presented on or beyond the line number specified by integer-3.

d. LAST DETAIL integer-4 defines the last line number on which a CONTROL HEADING or DETAIL report group may be presented.

e. FOOTING integer-5 defines the last line number on which a CONTROL FOOTING report group may be presented. PAGE FOOTING and REPORT FOOTING report groups must follow the line number specified by integer-5.

(2) If the PAGE clause is specified the following implicit values are assumed for any omitted phrases:

a. If the HEADING phrase is omitted, a value of one (1) is assumed for integer-2.

b. If the FIRST DETAIL phrase is omitted, a value equal to integer-2 is given to integer-3.

c. If the LAST DETAIL and the FOOTING phrases are both omitted, the value of integer-1 is given to both integer-4 and integer-5.

d. If the FOOTING phrase is specified and the LAST DETAIL phrase is omitted, the value of integer-5 is given to integer-4.

e. If the LAST DETAIL phrase is specified and the FOOTING phrase is omitted, the value of integer-4 is given to integer-5.

(3) If the PAGE clause is omitted, the report consists of a single page of indefinite length.

(4) The presentation rules for each TYPE of report group are specified in paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.

2.16.5 Page Regions Table

Page regions that are established by the PAGE clause are described below.

Report Groups That May Be Presented In The Region	First Line Number Of The Region	Last Line Number Of The Region
REPORT HEADING described with NEXT GROUP NEXT PAGE REPORT FOOTING described with LINE integer-1 NEXT PAGE	integer-2	integer-1
REPORT HEADING not described with NEXT GROUP NEXT PAGE PAGE HEADING	integer-2	integer-3 minus 1
CONTROL HEADING DETAIL	integer-3	integer-4
CONTROL FOOTING	integer-3	integer-5
PAGE FOOTING REPORT FOOTING not described with LINE integer-1 NEXT PAGE	integer-5 plus 1	integer-1

2.17 THE RECORD CONTAINS CLAUSE

2.17.1 Function

The RECORD CONTAINS clause specifies the size of data records.

2.17.2 General Format

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

2.17.3 General Rules

(1) The size of each data record is completely defined within the record description entry, therefore this clause is never required. When present, however, the following notes apply:

a. Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

b. The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in any variable length item subordinate to the record. This sum may be different from the actual size of the record; see page I-85, Selection of Character Representation and Radix; page II-33, The SYNCHRONIZED Clause; and page II-35, The USAGE Clause.

2.18 THE REPORT CLAUSE

2.18.1 Function

The REPORT clause specifies the names of reports that comprise a report file.

2.18.2 General Format

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

2.18.3 Syntax Rules

(1) Each report-name specified in a REPORT clause must be the subject of a report description entry in the Report Section. The order of appearance of the report-names is not significant.

(2) A report-name must appear in only one REPORT clause.

(3) The subject of a file description entry that specifies a REPORT clause may only be referred to by the OPEN OUTPUT, OPEN EXTEND, and CLOSE statements.

2.18.4 General Rules

(1) The presence of more than one report-name in a REPORT clause indicates that the file contains more than one report.

2.19 THE SOURCE CLAUSE

2.19.1 Function

The SOURCE clause identifies the sending data item that is moved to an associated printable item defined within a report group description entry.

2.19.2 General Format

SOURCE IS identifier-1

2.19.3 Syntax Rules

(1) Identifier-1 may be defined in any section of the Data Division. If identifier-1 is a Report Section item it can only be:

a. PAGE-COUNTER, or

b. LINE-COUNTER, or

c. A sum counter of the report within which the SOURCE clause appears.

(2) Identifier-1 specifies the sending data item of the implicit MOVE statement that the RWCS will execute to move identifier-1 to the printable item. Identifier-1 must be defined such that it conforms to the rules for sending items in the MOVE statement. (See page II-74, The MOVE Statement.)

2.19.4 General Rules

(1) The RWCS formats the print lines of a report group just prior to presenting the report group. (See page VIII-45, The TYPE Clause.) It is at this time that the implicit MOVE statements specified by SOURCE clauses are executed by the RWCS.

2.20 THE SUM CLAUSE

2.20.1 Function

The SUM clause establishes a sum counter and names the data items to be summed.

2.20.2 General Format

```
{SUM identifier-1  [, identifier-2 ] ...
  [UPON data-name-1  [, data-name-2 ] ... ]} ...
  [RESET ON {data-name-3}]
            {FINAL     }
```

2.20.3 Syntax Rules

(1) Identifier-1 and identifier-2 must be defined as numeric data items. When defined in the Report Section, identifier-1 and identifier-2 must be the names of sum counters.

If the UPON phrase is omitted, any identifiers in the associated SUM clause which are themselves sum counters must be defined either in the same report group that contains this SUM clause or in a report group which is at a lower level in the control hierarchy of this report.

If the UPON phrase is specified, any identifiers in the associated SUM clause must not be sum counters.

(2) Data-name-1 and data-name-2 must be the names of DETAIL report groups described in the same report as the CONTROL FOOTING report group in which the SUM clause appears. Data-name-1 and data-name-2 may be qualified by a report-name.

(3) A SUM clause can appear only in the description of a CONTROL FOOTING report group.

(4) Data-name-3 must be one of the data-names specified in the CONTROL clause for this report. Data-name-3 must not be a lower level control than the associated control for the report group in which the RESET phrase appears.

FINAL, if specified in the RESET phrase, must also appear in the CONTROL clause for this report.

(5) The highest permissible qualifier of a sum counter is the report-name.

2.20.4 General Rules

(1) The SUM clause establishes a sum counter. The sum counter is a numeric data item with an optional sign. At object time the RWCS adds directly into the sum counter each of the values contained in identifier-1 and identifier-2. This addition is performed under the rules of the ADD statement. (See page II-55, The ADD Statement.)

(2) The size of the sum counter is equal to the number of receiving character positions specified by the PICTURE clause that accompanies the SUM clause in the description of the elementary item.

(3) Only one sum counter exists for an elementary report entry regardless of the number of SUM clauses specified in the elementary report entry.

(4) If the elementary report entry for a printable item contains a SUM clause, the sum counter serves as a source data item. The RWCS moves the data contained in the sum counter, according to the rules of the MOVE statement, to the printable item for presentation.

(5) If a data-name appears as the subject of an elementary report entry that contains a SUM clause, the data-name is the name of the sum counter; the data-name is not the name of the printable item that the entry may also define.

It is permissible for Procedure Division statements to alter the contents of sum counters.

(6) Addition of the identifiers into sum counters is performed by the RWCS during the execution of GENERATE and TERMINATE statements. There are three categories of sum counter incrementing called subtotalling, crossfooting, and rolling forward. Subtotalling is accomplished during execution of GENERATE statements only, after any control break processing but before processing of the DETAIL report group. (See page VIII-51, The GENERATE Statement.) Crossfooting and rolling forward are accomplished during the processing of CONTROL FOOTING report groups. (See page VIII-45, The TYPE Clause.)

(7) The UPON phrase provides the capability to accomplish selective subtotalling for the DETAIL report groups named in the phrase.

(8) The RWCS adds each individual addend into the sum counter at a time that depends upon the characteristics of the addend.

a. When the addend is a sum counter defined in the same CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed crossfooting.

Crossfooting occurs when a control break takes place and at the time the CONTROL FOOTING report group is processed.

Crossfooting is performed according to the sequence in which sum counters are defined within the CONTROL FOOTING report group. That is, all crossfooting into the first sum counter defined in the CONTROL FOOTING report group is completed, and then all crossfooting into the second sum counter defined in the CONTROL FOOTING report group is completed. This procedure is repeated until all crossfooting operations are completed.

b. When the addend is a sum counter defined in a lower level CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed rolling forward. A sum counter in a lower level CONTROL FOOTING report group is rolled forward when a control break occurs and at the time that the lower level CONTROL FOOTING report group is processed.

c. When the addend is not a sum counter the accumulation into a sum counter of such an addend is called subtotalling. If the SUM clause contains the UPON phrase, the addends are subtotalled when a GENERATE statement for the designated DETAIL report group is executed. If the SUM clause does not contain the UPON phrase, the addends which are not sum counters are subtotalled when any GENERATE data-name statement is executed for the report in which the SUM clause appears.

(9) If two or more of the identifiers specify the same addend, then the addend is added into the sum counter as many times as the addend is referenced in the SUM clause. It is permissible for two or more of the data-names to specify the same DETAIL report group. When a GENERATE data-name statement for such a DETAIL report group is given, the incrementing occurs repeatedly, as many times as data-name appears in the UPON phrase.

(10) For the subtotalling that occurs when a GENERATE report-name statement is executed, see page VIII-51, The GENERATE Statement.

(11) In the absence of an explicit RESET phrase, the RWCS will set a sum counter to zero at the time that the RWCS is processing the CONTROL FOOTING report group within which the sum counter is defined. If an explicit RESET phrase is specified, then the RWCS will set the sum counter to zero at the time that the RWCS is processing the designated level of the control hierarchy. (See page VIII-45, The TYPE Clause.)

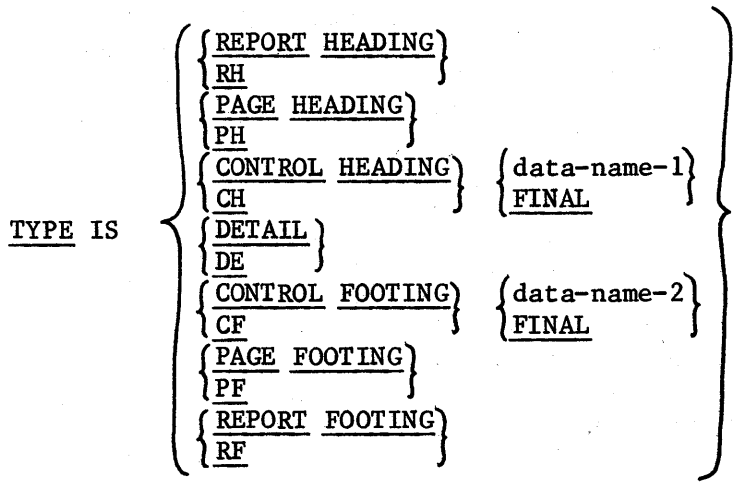
Sum counters are initially set to zero by the RWCS during the execution of the INITIATE statement for the report containing the sum counter.

2.21 THE TYPE CLAUSE

2.21.1 Function

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be processed by the Report Writer Control System.

2.21.2 General Format



2.21.3 Syntax Rules

(1) RH is an abbreviation for REPORT HEADING.
PH is an abbreviation for PAGE HEADING.
CH is an abbreviation for CONTROL HEADING.
DE is an abbreviation for DETAIL.
CF is an abbreviation for CONTROL FOOTING.
PF is an abbreviation for PAGE FOOTING.
RF is an abbreviation for REPORT FOOTING.

(2) REPORT HEADING, PAGE HEADING, CONTROL HEADING FINAL, CONTROL FOOTING FINAL, PAGE FOOTING, and REPORT FOOTING report groups may each appear no more than once in the description of a report.

(3) PAGE HEADING and PAGE FOOTING report groups may be specified only if a PAGE clause is specified in the corresponding report description entry.

(4) Data-name-1, data-name-2 and FINAL, if present, must be specified in the CONTROL clause of the corresponding report description entry. At most, one CONTROL HEADING report group and one CONTROL FOOTING report group can be specified for each data-name or FINAL in the CONTROL clause of the report description entry. However, neither a CONTROL HEADING report group nor a CONTROL FOOTING report group is required for a data-name or FINAL specified in the CONTROL clause of the report description entry.

(5) In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE clauses and USE statements must not reference any of the following:

a. Group data items containing a control data item.

b. Data items subordinate to a control data item.

c. A redefinition or renaming of any part of a control data item.

In PAGE HEADING and PAGE FOOTING report groups, SOURCE clauses and USE statements must not reference control data-names.

(6) When a GENERATE report-name statement is specified in the Procedure Division, the corresponding report description entry must include no more than one DETAIL report group. If no GENERATE data-name statements are specified for such a report, a DETAIL report group is not required.

(7) The description of a report must include at least one body group.

2.21.4 General Rules

(1) DETAIL report groups are processed by the RWCS as a direct result of GENERATE statements. If a report group is other than TYPE DETAIL, its processing is an automatic RWCS function.

(2) The REPORT HEADING phrase specifies a report group that is processed by the RWCS only once, per report, as the first report group of that report. The REPORT HEADING report group is processed during the execution of the chronologically first GENERATE statement for that report.

(3) The PAGE HEADING phrase specifies a report group that is processed by the RWCS as the first report group on each page of that report except under the following conditions:

a. A PAGE HEADING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.

b. A PAGE HEADING report group is processed as the second report group on a page when it is preceded by a REPORT HEADING report group that is not to be presented on a page by itself.

See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9, for further information.

(4) The CONTROL HEADING phrase specifies a report group that is processed by the RWCS at the beginning of a control group for a designated control data-name or, in the case of FINAL, is processed during the execution of the chronologically first GENERATE statement for that report. During the execution of any GENERATE statement at which the RWCS detects a control break, any CONTROL HEADING report groups associated with the highest control level of the break and lower levels are processed.

(5) The DETAIL phrase specifies a report group that is processed by the RWCS when a corresponding GENERATE statement is executed.

(6) The CONTROL FOOTING phrase specifies a report group that is processed by the RWCS at the end of a control group for a designated control data-name.

In the case of FINAL, the CONTROL FOOTING report group is processed only once per report as the last body group of that report. During the execution of any GENERATE statement in which the RWCS detects a control break, any CONTROL FOOTING report group associated with the highest level of the control break or more minor levels is presented. All CONTROL FOOTING report groups are presented during the execution of the TERMINATE statement if there has been at least one GENERATE statement executed for the report. (See page VIII-55, The TERMINATE Statement.)

(7) The PAGE FOOTING phrase specifies a report group that is processed by the RWCS as the last report group on each page except under the following conditions:

a. A PAGE FOOTING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.

b. A PAGE FOOTING report group is processed as the second to last report group on a page when it is followed by a REPORT FOOTING report group that is not to be processed on a page by itself.

See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9, for further information.

(8) The REPORT FOOTING phrase specifies a report group that is processed by the RWCS only once per report and as the last report group of that report. The REPORT FOOTING report group is processed during the execution of a corresponding TERMINATE statement, if there has been at least one GENERATE statement executed for the report. (See page VIII-55, The TERMINATE Statement.)

(9) The sequence of steps that the RWCS executes when it processes a REPORT HEADING, PAGE HEADING, CONTROL HEADING, PAGE FOOTING, or REPORT FOOTING report group is described below.

a. If there is a USE BEFORE REPORTING procedure that references the data-name of the report group, the USE procedure is executed.

b. If a SUPPRESS statement has been executed or if the report group is not printable, there is no further processing to be done for the report group.

c. Otherwise, the RWCS formats the print lines and presents the report group according to the presentation rules for that type of report group. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

(10) The sequence of steps that the RWCS executes when it processes a CONTROL FOOTING report group is described below.

The GENERATE rules specify that when a control break occurs, the RWCS produces the CONTROL FOOTING report groups beginning at the minor level, and proceeding upwards, through the level at which the highest control break was sensed. In this regard, it should be noted that even though no CONTROL FOOTING report group has been defined for a given control data-name, the RWCS will still have to execute the step described in paragraph 10f below if a RESET phrase within the report description specifies that control data-name.

a. Sum counters are crossfooted, i.e., all sum counters defined in this report group that are operands of SUM clauses in the same report group are added to their sum counters. (See page VIII-42, The SUM Clause.)

b. Sum counters are rolled forward, i.e., all sum counters defined in the report group that are operands of SUM clauses in higher level CONTROL FOOTING report groups are added to the higher level sum counters. (See page VIII-42, The SUM Clause.)

c. If there is a USE BEFORE REPORTING procedure that references the data-name of the report group the USE procedure is executed.

d. If a SUPPRESS statement has been executed or if the report group is not printable, the RWCS next executes the step described in paragraph 10f below.

e. Otherwise the RWCS formats the print lines and presents the report group according to the presentation rules for CONTROL FOOTING report groups.

f. Then the RWCS resets those sum counters that are to be reset when the RWCS processes this level in the control hierarchy. (See page VIII-42, The SUM Clause.)

(11) The DETAIL report group processing that the RWCS executes in response to a GENERATE data-name statement is described in paragraphs 11a through 11e below.

When the description of a report includes exactly one DETAIL report group, the detail-related processing that the RWCS executes in response to a GENERATE report-name statement is described in paragraph 11a through paragraph 11d below. These steps are performed as though a GENERATE data-name statement were being executed.

When the description of a report includes no DETAIL report groups, the detail-related processing that the RWCS executes in response to a GENERATE report-name statement is described in paragraph 11a below. This step is performed as though the description of the report included exactly one DETAIL report group, and a GENERATE data-name statement were being executed.

a. The RWCS performs any subtotalling that has been designated for the DETAIL report group. (See page VIII-42, The SUM Clause.)

b. If there is a USE BEFORE REPORTING procedure that refers to the data-name of the report group, the USE procedure is executed.

c. If a SUPPRESS statement has been executed or if the report group is not printable there is no further processing done for the report group.

d. If the DETAIL report group is being processed as a consequence of a GENERATE report-name statement, there is no further processing done for the report group.

e. Otherwise the RWCS formats the print lines and presents the report group according to the presentation rules for DETAIL report groups. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

(12) When the RWCS is processing a CONTROL HEADING, CONTROL FOOTING, or DETAIL report group, as described in general rules 9, 10, and 11, the RWCS may have to interrupt the processing of that body group after determining that the body group is to be presented, and execute a page advance (and process PAGE FOOTING and PAGE HEADING report groups) before actually presenting the body group.

(13) During control break processing, the values of control data items that the RWCS used to detect a given control break are referred to as prior values.

a. During control break processing of a CONTROL FOOTING report group, any references to control data items in a USE procedure or SOURCE clause associated with that CONTROL FOOTING report group are supplied with prior values.

b. When a TERMINATE statement is executed, the RWCS makes the prior control data item values available to SOURCE clause or USE procedure references in CONTROL FOOTING and REPORT FOOTING report groups as though a control break had been detected in the highest control data-name.

c. All other data item references within report groups and their USE procedures access the current values that are contained within the data items at the time the report group is processed.

2.22 THE VALUE OF CLAUSE

2.22.1 Function

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

2.22.2 General Format

```
VALUE OF implementor-name-1 IS  { data-name-1 }
                                { literal-1   }

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

2.22.3 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.

(3) See page IV-19, The VALUE OF Clause, for constraints that apply when Report Writer is associated with Sequential I-O, Level 1.

2.22.4 General Rules

(1) 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 data-name-1, whichever has been specified.

(2) A figurative constant may be substituted in the format above wherever a literal is specified.

3. PROCEDURE DIVISION IN THE REPORT WRITER MODULE

3.1 THE GENERATE STATEMENT

3.1.1 Function

The GENERATE statement directs the RWCS to produce a report in accordance with the report description that was specified in the Report Section of the Data Division.

3.1.2 General Format

GENERATE { data-name / report-name }

3.1.3 Syntax Rules

(1) Data-name must name a TYPE DETAIL report group and may be qualified by a report-name.

(2) Report-name may be used only if the referenced report description contains:

a. A CONTROL clause, and

b. Not more than one DETAIL report group, and

c. At least one body group.

3.1.4 General Rules

(1) In response to a GENERATE report-name statement, the RWCS performs summary processing. If all of the GENERATE statements that are executed for a report are of the form GENERATE report-name, then the report that is produced is called a summary report. A summary report is one in which no DETAIL report group is presented.

(2) In response to a GENERATE data-name statement, the RWCS performs detail processing that includes certain processing that is specific for the DETAIL report group designated by the GENERATE statement. Normally, the execution of a GENERATE data-name statement causes the RWCS to present the designated DETAIL report group.

(3) During the execution of the chronologically first GENERATE statement for a given report, the RWCS saves the values within the control data items. During the execution of the second and subsequent GENERATE statements for the same report, and until a control break is detected, the RWCS utilizes this set of control values to determine whether a control break has occurred. When a control break occurs, the RWCS saves the new set of control values, which it thereafter uses to sense for a control break until another control break occurs.

(4) During report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS

must advance the report to a new page for the purpose of presenting a body group. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

(5) When the chronologically first GENERATE statement for a given report is executed, the RWCS processes, in order, the report groups that are named below, provided that such report groups are defined within the report description. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in general rule 4. See page VIII-45, The TYPE Clause, for the actions that the RWCS takes when it processes each type of report group.

a. The REPORT HEADING report group is processed.

b. The PAGE HEADING report group is processed.

c. All CONTROL HEADING report groups are processed from major to minor.

d. If a GENERATE data-name statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name statement is being executed, certain of the steps that are involved in the processing of a DETAIL report group are performed. (See page VIII-45, The TYPE Clause.)

(6) When a GENERATE statement other than the chronologically first is executed for a given report, the RWCS performs the steps enumerated below, as applicable. The RWCS also processes PAGE HEADING and PAGE FOOTING report groups as described in general rule 4. See page VIII-45, The TYPE Clause, for the actions that the RWCS takes when it processes each type of report group.

a. Sense for control break. The rules for determining the equality of control data items are the same as those specified for relation conditions. If a control break has occurred then:

1) Enable the CONTROL FOOTING USE procedures and CONTROL FOOTING SOURCE clauses to access the control data item values that are described on page VIII-45, The TYPE Clause.

2) Process the CONTROL FOOTING report groups in the order minor to major. Only CONTROL FOOTING report groups that are not more major than the highest level at which a control break occurred are processed.

3) Process the CONTROL HEADING report groups in the order major to minor. Only the CONTROL HEADING report groups that are not more major than the highest level at which a control break occurred are processed.

b. If a GENERATE data-name statement is being executed, the processing for the designated DETAIL report group is performed. If a GENERATE report-name statement is being executed, certain of the steps that are involved in the processing of a DETAIL report group are performed. (See page VIII-45, The TYPE Clause.)

(7) GENERATE statements for a report can be executed only after an INITIATE statement for the report has been executed and before a TERMINATE statement for the report has been executed.

3.2 THE INITIATE STATEMENT

3.2.1 Function

The INITIATE statement causes the Report Writer Control System to begin the processing of a report.

3.2.2 General Format

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

3.2.3 Syntax Rules

(1) Each report-name must be defined by a report description entry in the Report Section of the Data Division.

3.2.4 General Rules

(1) The INITIATE statement performs the following initialization functions for each named report:

a. All sum counters are set to zero.

b. LINE-COUNTER is set to zero.

c. PAGE-COUNTER is set to one (1).

(2) The INITIATE statement does not open the file with which the report is associated, therefore an OPEN statement with either the OUTPUT phrase or the EXTEND phrase for the file must be executed prior to the execution of the INITIATE statement.

(3) A subsequent INITIATE statement for a particular report-name must not be executed unless an intervening TERMINATE statement has been executed for that report-name.

3.3 THE SUPPRESS STATEMENT

3.3.1 Function

The SUPPRESS statement causes the Report Writer Control System to inhibit the presentation of a report group.

3.3.2 General Format

SUPPRESS PRINTING

3.3.3 Syntax Rules

(1) The SUPPRESS statement may only appear in a USE BEFORE REPORTING procedure.

3.3.4 General Rules

(1) The SUPPRESS statement inhibits presentation only for the report group named in the USE procedure within which the SUPPRESS statement appears.

(2) The SUPPRESS statement must be executed each time the presentation of the report group is to be inhibited.

(3) When the SUPPRESS statement is executed, the RWCS is instructed to inhibit the processing of the following report group functions:

a. The presentation of the print lines of the report group,

b. The processing of all LINE clauses in the report group,

c. The processing of the NEXT GROUP clause in the report group,

d. The adjustment of LINE-COUNTER.

3.4 THE TERMINATE STATEMENT

3.4.1 Function

The TERMINATE statement causes the Report Writer Control System to complete the processing of the specified reports.

3.4.2 General Format

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

3.4.3 Syntax Rules

(1) Each report-name given in a TERMINATE statement must be defined by an RD entry in the Report Section of the Data Division.

3.4.4 General Rules

(1) The TERMINATE statement causes the RWCS to produce all the CONTROL FOOTING report groups beginning with the minor CONTROL FOOTING report group. Then the REPORT FOOTING report group is produced. The RWCS makes the prior set of control data item values available to the CONTROL FOOTING and REPORT FOOTING SOURCE clauses and USE procedures, as though a control break has been sensed in the most major control data-name.

(2) If no GENERATE statements have been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement, for that report, the TERMINATE statement does not cause the RWCS to produce any report groups or perform any of the related processing.

(3) During report presentation, an automatic function of the RWCS is to process PAGE HEADING and PAGE FOOTING report groups, if defined, when the RWCS must advance the report to a new page for the purpose of presenting a body group. (See paragraph 2.5.5, Presentation Rules Tables, beginning on page VIII-9.)

(4) The TERMINATE statement cannot be executed for a report unless the TERMINATE statement was chronologically preceded by an INITIATE statement for that report and for which no TERMINATE statement has yet been executed.

(5) The TERMINATE statement does not close the file with which the report is associated; a CLOSE statement for the file must be executed. Every report in a file that is in an initiated condition must be terminated before a CLOSE statement is executed for that file.

3.5 THE USE STATEMENT

3.5.1 Function

The USE statement specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

3.5.2 General Format

USE BEFORE REPORTING identifier.

3.5.3 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) Identifier represents a report group. Identifier must not appear in more than one USE statement.

The GENERATE, INITIATE or TERMINATE statements must not appear in a paragraph within a USE BEFORE REPORTING procedure.

A USE BEFORE REPORTING procedure must not alter the value of any control data item.

(3) The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

3.5.4 General Rules

(1) The designated procedures are executed by the Report Writer Control System just before the named report group is produced. (See page VIII-45, The TYPE Clause.)

(2) 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 BEFORE REPORTING statement or to the procedures associated with such a USE statement.

1. INTRODUCTION TO THE SEGMENTATION MODULE

1.1 FUNCTION

The Segmentation module provides a capability to specify object program overlay requirements.

1.2 LEVEL CHARACTERISTICS

Segmentation Level 1 provides a facility for specifying permanent and independent segments (see paragraph 2.2.1 on page IX-2). All sections with the same segment-number must be contiguous in the source program. All segments specified as permanent segments must be contiguous in the source program.

Segmentation Level 2 provides the facility for intermixing sections with different segment-numbers and allows the fixed portion of the source program to contain segments that may be overlaid (see paragraph 2.2.2 on page IX-2).

2. GENERAL DESCRIPTION OF SEGMENTATION

COBOL segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements.

2.1 SCOPE

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division and the Environment Division are considered in determining segmentation requirements for an object program.

2.2 ORGANIZATION

2.2.1 Program Segments

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to insure uniqueness.

2.2.2 Fixed Portion

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of two types of segments: fixed permanent segments and fixed overlayable segments.

A fixed permanent segment is a segment in the fixed portion which cannot be overlaid by any other part of the program. A fixed overlayable segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlaid by another segment to optimize memory utilization. Variation of the number of fixed permanent segments in the fixed portion can be accomplished by using a special facility called the SEGMENT-LIMIT clause (see page IX-5, SEGMENT-LIMIT). Such a segment, if called for by the program, is always made available in its last used state.

2.2.3 Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, either a fixed overlayable segment or another independent segment. An independent segment is in its initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program. On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

(1) Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

(2) Control is transferred to that segment as the result of the implicit transfer of control between a SORT or MERGE statement, in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

(3) Control is transferred explicitly to that segment from a segment with a different segment-number (with the exception noted in paragraph 2 below).

On subsequent transfer of control to the segment, an independent segment is in its last-used state when:

(1) Control is transferred implicitly to that segment from a segment with a different segment-number (except as noted in paragraphs 1 and 2 above).

(2) Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

See paragraph 3.4.2, Explicit and Implicit Transfers of Control, page I-92.

2.3 SEGMENTATION CLASSIFICATION

Sections which are to be segmented are classified, using a system of segment-numbers (see paragraph 3.1 on page IX-4) and the following criteria:

(1) Logic Requirements - Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections which are used less frequently are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.

(2) Frequency of Use - Generally, the more frequently a section is referred to, the lower its segment-number, the less frequently it is referred to, the higher its segment-number.

(3) Relationship to Other Sections - Sections which frequently communicate with one another should be given the same segment-numbers.

2.4 SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. If any reordering of the object program is required to handle the flow from segment to segment, according to the rules in paragraph 3.1 on page IX-4, the implementor must provide control transfers to maintain the logical flow specified in the source program. The implementor must also provide all controls necessary for a segment to operate whenever the segment is used. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

3. STRUCTURE OF PROGRAM SEGMENTS

3.1 SEGMENT-NUMBERS

Section classification is accomplished by means of a system of segment-numbers. The segment-number is included in the section header.

3.1.1 General Format

section-name SECTION [segment-number] .

3.1.2 Syntax Rules

(1) The segment-number must be an integer ranging in value from 0 through 99.

(2) If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

(3) Sections in the declaratives must contain segment-numbers less than 50.

3.1.3 General Rules

(1) All sections which have the same segment-number constitute a program segment. In Level 1 all sections which have the same segment-number must be together in the source program. In Level 2 sections with the same segment-numbers need not be physically contiguous in the source program.

(2) Segments with segment-number 0 through 49 belong to the fixed portion of the object program. In Level 1 all sections with segment-number 0 through 49 must be together in the source program.

(3) Segments with segment-number 50 through 99 are independent segments.

3.2 SEGMENT-LIMIT

Ideally, all program segments having segment-numbers ranging from 0 through 49 should be specified as permanent segments. However, when insufficient memory is available to contain all permanent segments plus the largest overlayable segment, it becomes necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means by which he can reduce the number of permanent segments in his program, while still retaining the logical properties of fixed portion segments (segment-numbers 0 through 49).

3.2.1 General Format

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph and has the following format:

[, SEGMENT-LIMIT IS segment-number]

3.2.2 Syntax Rules

(1) Segment-number must be an integer ranging in value from 1 through 49.

3.2.3 General Rules

(1) When the SEGMENT-LIMIT clause is specified, only those segments having segment-numbers from 0 up to, but not including, the segment-number designated as the segment-limit, are considered as permanent segments of the object program.

(2) Those segments having segment-numbers from the segment-limit through 49 are considered as overlayable fixed segments.

(3) When the SEGMENT-LIMIT clause is omitted, all segments having segment-numbers from 0 through 49 are considered as permanent segments of the object program.

4. RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER, PERFORM, MERGE, and SORT statements.

4.1 THE ALTER STATEMENT

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 the GO TO to which the ALTER refers is in a fixed overlayable segment.

4.2 THE PERFORM STATEMENT

A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

a. Sections and/or paragraphs wholly contained in one or more non-independent segments.

b. Sections and/or paragraph wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

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

4.3 THE MERGE STATEMENT

If the MERGE statement appears in a section that is not in an independent segment, then any output procedure referenced by that MERGE statement must appear:

a. Totally within non-independent segments, or

b. Wholly contained in a single independent segment.

If a MERGE statement appears in an independent segment, then any output procedure referenced by that MERGE statement must be contained:

a. Totally within non-independent segments, or

b. Wholly within the same independent segment as that MERGE statement.

4.4 THE SORT STATEMENT

If a SORT statement appears in a section that is not an independent segment, then any input procedures or output procedures referenced by that SORT statement must appear:

a. Totally within non-independent segments, or

b. Wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, then any input procedures or output procedures referenced by that SORT statement must be contained:

a. Totally within non-independent segments, or

b. Wholly within the same independent segment as that SORT statement.

1. INTRODUCTION TO THE LIBRARY MODULE

1.1 FUNCTION

The Library module provides a capability for specifying text that is to be copied from a library.

COBOL libraries contain library texts that are available to the compiler for copying at compile time. The effect of the interpretation of the COPY statement is to insert text into the source program, where it will be treated by the compiler as part of the source program.

COBOL library text is placed on the COBOL library as a function independent of the COBOL program and according to implementor-defined techniques.

1.2 LEVEL CHARACTERISTICS

Library Level 1 provides the facility for copying text from a single library into the source program. Text is copied from the library without change.

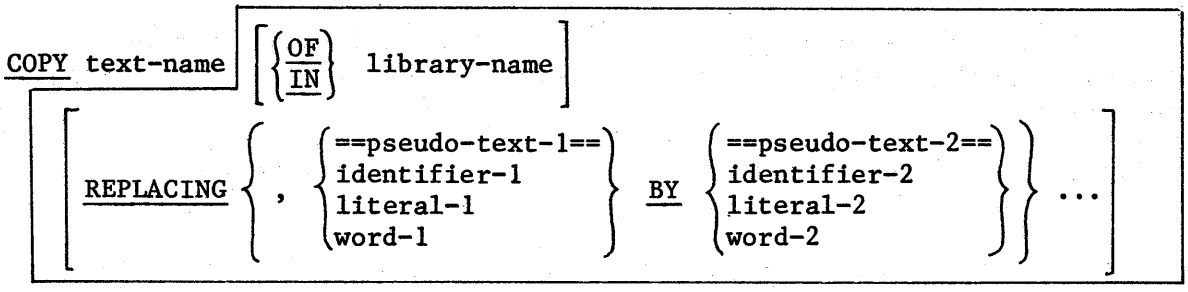
Library Level 2 provides the additional capability of replacing all occurrences of a given literal, identifier, word or group of words in the library text, with alternate text, during the copying process. Level 2 also provides for the availability of more than one COBOL library at compile time.

2. THE COPY STATEMENT

2.1 FUNCTION

The COPY statement incorporates text into a COBOL source program.

2.2 GENERAL FORMAT



2.3 SYNTAX RULES

(1) If more than one COBOL library is available during compilation, text-name must be qualified by the library-name identifying the COBOL library in which the text associated with text-name resides. (See page II-1, Name Characteristics, for constraints that apply when Library is associated with Nucleus, Level 1.)

Within one COBOL library, each text-name must be unique.

(2) The COPY statement must be preceded by a space and terminated by the separator period.

(3) Pseudo-text-1 must not be null, nor may it consist solely of the character space(s), nor may it consist solely of comment lines.

(4) Pseudo-text-2 may be null.

(5) Character-strings within pseudo-text-1 and pseudo-text-2 may be continued. However, both characters of a pseudo-text delimiter must be on the same line. (See page I-106, Continuation of Lines.)

(6) Word-1 or word-2 may be any single COBOL word.

(7) A COPY statement may occur in the source program anywhere a character-string or a separator may occur except that a COPY statement must not occur within a COPY statement.

2.4 GENERAL RULES

(1) The compilation of a source program containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of the resulting source program.

(2) The effect of processing a COPY statement is that the library text associated with text-name is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and ending with the punctuation character period, inclusive.

(3) If the REPLACING phrase is not specified, the library text is copied unchanged.

If the REPLACING phrase is specified, the library text is copied and each properly matched occurrence of pseudo-text-1, identifier-1, word-1, and literal-1 in the library text is replaced by the corresponding pseudo-text-2, identifier-2, word-2, or literal-2.

(4) For purposes of matching, identifier-1, word-1, and literal-1 are treated as pseudo-text containing only identifier-1, word-1, or literal-1, respectively.

(5) The comparison operation to determine text replacement occurs in the following manner:

Any separator comma, semicolon and/or space(s) preceding the leftmost library text-word is copied into the source program. Starting with the leftmost library text-word **and** the first pseudo-text-1, identifier-1, word-1, or literal-1 that was specified in the REPLACING phrase, the entire REPLACING phrase operand that precedes the reserved word BY is compared to an equivalent number of contiguous library text-words.

Pseudo-text-1, identifier-1, word-1, or literal-1 match the library text if, and only if, the ordered sequence of text-words that forms pseudo-text-1, identifier-1, word-1, or literal-1 is equal, character for character, to the ordered sequence of library text-words. For purposes of matching, each occurrence of a separator comma or semicolon in pseudo-text-1 or in the library text is considered to be a single space except when pseudo-text-1 consists solely of either a separator comma or semicolon, in which case it participates in the match as a text-word. Each sequence of one or more space separators is considered to be a single space.

If no match occurs, the comparison is repeated with each next successive pseudo-text-1, identifier-1, word-1, or literal-1, if any, in the REPLACING phrase until either a match is found or there is no next successive REPLACING operand.

When all the REPLACING phrase operands have been compared and no match has occurred, the leftmost library text-word is copied into the source program. The next successive library text-word is then considered as the leftmost library text-word, and the comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, or literal-1 specified in the REPLACING phrase.

Whenever a match occurs between pseudo-text-1, identifier-1, word-1, or literal-1 and the library text, the corresponding pseudo-text-2, identifier-2, word-2, or literal-2 is placed into the source program. The library text-word immediately following the rightmost text-word that participated in the match is then considered as the leftmost library text-word. The comparison cycle starts again with the first pseudo-text-1, identifier-1, word-1, or literal-1 specified in the REPLACING phrase.

The comparison operation continues until the rightmost text-word in the library text has either participated in a match or been considered as a leftmost library text-word and participated in a complete comparison cycle.

(6) A comment line occurring in the library text and pseudo-text-1 is interpreted, for purposes of matching, as a single space. Comment lines appearing in pseudo-text-2 and library text are copied into the source program unchanged.

(7) Debugging lines are permitted within library text and pseudo-text-2. Debugging lines are not permitted within pseudo-text-1; text-words within a debugging line participate in the matching rules as if the 'D' did not appear in the indicator area. If a COPY statement is specified on a debugging line, then the text that is the result of the processing of the COPY statement will appear as though it were specified on debugging lines with the following exception: comment lines in library text will appear as comment lines in the resultant source program.

(8) The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

(9) The syntactic correctness of the library text cannot be independently determined. The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed.

(10) Library text must conform to the rules for COBOL reference format.

(11) For purposes of compilation, text-words after replacement are placed in the source program according to the rules for reference format. (See page I-105, Reference Format.)

1. INTRODUCTION TO THE DEBUG MODULE

1.1 FUNCTION

The Debug module provides a means by which the user can describe his debugging algorithm including the conditions under which data items or procedures are to be monitored during the execution of the object program.

The decisions of what to monitor and what information to display on the output device are explicitly in the domain of the user. The COBOL debug facility simply provides a convenient access to pertinent information.

1.2 LEVEL CHARACTERISTICS

Debug Level 1 provides a basic debugging capability, including the ability to specify: (a) selective or full procedure monitoring, and (b) optionally compiled debugging statements.

Debug Level 2 provides the full COBOL debugging facility.

1.3 LANGUAGE CONCEPTS

The features of the COBOL language that support the Debug module are:

a. A compile time switch -- WITH DEBUGGING MODE.

b. An object time switch.

c. A USE FOR DEBUGGING statement.

d. A special register -- DEBUG-ITEM.

e. Debugging lines.

1.3.1 DEBUG-ITEM

The reserved word DEBUG-ITEM is the name for a special register generated automatically by the implementor's code that supports the debugging facility. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

1.3.2 A Compile Time Switch

The WITH DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph. It serves as a compile time switch over the debugging statements written in the program.

When the WITH DEBUGGING MODE clause is specified in a program, all debugging sections and all debugging lines are compiled as specified in this section of the document. When the WITH DEBUGGING MODE clause is not specified, all debugging lines and all debugging sections are compiled as if they were comment lines.

1.3.3 An Object Time Switch

An object time switch dynamically activates the debugging code inserted by the compiler. This switch cannot be addressed in the program; it is controlled outside the COBOL environment. If the switch is 'on', all the effects of the debugging language written in the source program are permitted. If the switch is 'off', all the effects described in paragraph 3.1 on page XI-4, The USE FOR DEBUGGING Statement, are inhibited. Recompilation of the source program is not required to provide or take away this facility.

The object time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program at compile time.

2. ENVIRONMENT DIVISION IN THE DEBUG MODULE

2.1 THE WITH DEBUGGING MODE CLAUSE

2.1.1 Function

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be compiled. If this clause is not specified, all debugging lines and sections are compiled as if they were comment lines.

2.1.2 General Format

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

2.1.3 General Rules

(1) If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, all USE FOR DEBUGGING statements and all debugging lines are compiled.

(2) If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, any USE FOR DEBUGGING statements and all associated debugging sections, and any debugging lines are compiled as if they were comment lines.

3. PROCEDURE DIVISION IN THE DEBUG MODULE

3.1 THE USE FOR DEBUGGING STATEMENT

3.1.1 Function

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

3.1.2 General Format

```
section-name SECTION [segment-number] .

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

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

3.1.3 Syntax Rules

(1) Debugging section(s), if specified, must appear together immediately after the DECLARATIVES header.

(2) Except in the USE FOR DEBUGGING statement itself, there must be no reference to any non-declarative procedure within the debugging section.

(3) Statements appearing outside of the set of debugging sections must not reference procedure-names defined within the set of debugging sections.

(4) Except for the USE FOR DEBUGGING statement itself, statements appearing within a given debugging section may reference procedure-names defined within a different USE procedure only with a PERFORM statement.

(5) Procedure-names defined within debugging sections must not appear within USE FOR DEBUGGING statements.

(6) Any given identifier, cd-name, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.

(7) The ALL PROCEDURES phrase can appear only once in a program.

(8) When the ALL PROCEDURES phrase is specified, procedure-name-1, procedure-name-2, ... must not be specified in any USE FOR DEBUGGING statement.

(9) Identifier-1, identifier-2, ..., must not reference any data item defined in the Report Section except sum counters.

(10) If the data description entry of the data item referenced by identifier-1, identifier-2, ..., contains an OCCURS clause or is subordinate to a data description entry that contains an OCCURS clause, identifier-1, identifier-2, ..., must be specified without the subscripting or indexing normally required.

(11) References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

3.1.4 General Rules

(1) In the following general rules all references to cd-name-1, identifier-1, procedure-name-1, and file-name-1 apply equally to cd-name-2, identifier-2, procedure-name-2, and file-name-2, respectively.

(2) Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

(3) When file-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

a. After the execution of any OPEN or CLOSE statement that references file-name-1, and

b. After the execution of any READ statement (after any other specified USE procedure) not resulting in the execution of an associated AT END or INVALID KEY imperative statement, and

c. After the execution of any DELETE or START statement that references file-name-1.

(4) When procedure-name-1 is specified in a USE FOR DEBUGGING statement that debugging section is executed:

a. Immediately before each execution of the named procedure;

b. Immediately after the execution of an ALTER statement which references procedure-name-1.

(5) The ALL PROCEDURES phrase causes the effects described in general rule 4 to occur for every procedure-name in the program, except those appearing within a debugging section.

(6) When the ALL REFERENCES OF identifier-1 phrase is specified, that debugging section is executed for every statement that explicitly references identifier-1 at each of the following times:

a. In the case of a WRITE or REWRITE statement immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

b. In the case of a GO TO statement with a DEPENDING ON phrase, immediately before control is transferred and prior to the execution of any debugging section associated with the procedure-name to which control is to be transferred.

c. In the case of a PERFORM statement in which a VARYING, AFTER, or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.

d. In the case of any other COBOL statement, immediately after execution of that statement.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

(7) When identifier-1 is specified without the ALL REFERENCES OF phrase, that debugging section is executed at each of the following times:

a. In the case of a WRITE or REWRITE statement that explicitly references identifier-1, immediately before the execution of that WRITE or REWRITE statement and after the execution of any implicit move resulting from the presence of the FROM phrase.

b. In the case of a PERFORM statement in which a VARYING, AFTER or UNTIL phrase references identifier-1, immediately after each initialization, modification or evaluation of the contents of the data item referenced by identifier-1.

c. Immediately after the execution of any other COBOL statement that explicitly references and causes the contents of the data item referenced by identifier-1 to be changed.

If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

(8) The associated debugging section is not executed for a specific operand more than once as a result of the execution of a single statement, regardless of the number of times that operand is explicitly specified. In the case of a PERFORM statement which causes iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

(9) When cd-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

a. After the execution of any ENABLE, DISABLE, and SEND statement that references cd-name-1,

b. After the execution of a RECEIVE statement referencing cd-name-1 that does not result in the execution of the NO DATA imperative-statement, and

c. After the execution of an ACCEPT MESSAGE COUNT statement that references cd-name-1.

(10) A reference to file-name-1, identifier-1, procedure-name-1 or cd-name-1 as a qualifier does not constitute reference to that item for the debugging described in the general rules above.

(11) Associated with each execution of a debugging section is the special register DEBUG-ITEM, which provides information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```
01  DEBUG-ITEM.
    02  DEBUG-LINE       PICTURE IS X(6).
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-NAME       PICTURE IS X(30).
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-1      PICTURE IS S9999  SIGN IS LEADING SEPARATE CHARACTER.
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-2      PICTURE IS S9999  SIGN IS LEADING SEPARATE CHARACTER.
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-SUB-3      PICTURE IS S9999  SIGN IS LEADING SEPARATE CHARACTER.
    02  FILLER           PICTURE IS X  VALUE SPACE.
    02  DEBUG-CONTENTS   PICTURE IS X(n).
```

(12) Prior to each execution of a debugging section, the contents of the data item referenced by DEBUG-ITEM are space-filled. The contents of data items subordinate to DEBUG-ITEM are then updated, according to the following general rules, immediately before control is passed to that debugging section. The contents of any data item not specified in the following general rules remains spaces.

Updating is accomplished in accordance with the rules for the MOVE statement, the sole exception being the move to DEBUG-CONTENTS when the move is treated exactly as if it was an alphanumeric to alphanumeric elementary move with no conversion of data from one form of internal representation to another.

(13) The contents of DEBUG-LINE is the implementor-defined means of identifying a particular source statement.

(14) DEBUG-NAME contains the first 30 characters of the name that caused the debugging section to be executed.

All qualifiers of the name are separated in DEBUG-NAME by the word 'IN' or 'OF'. Subscripts/indices, if any, are not entered into DEBUG-NAME.

(15) If the reference to a data item that causes the debugging section to be executed is subscripted or indexed, the occurrence number of each level is entered in DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3 respectively as necessary.

(16) DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following general rules.

(17) If the first execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:

a. DEBUG-LINE identifies the first statement of that procedure.

b. DEBUG-NAME contains the name of that procedure.

c. DEBUG-CONTENTS contains 'START PROGRAM'.

(18) If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:

a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.

(19) If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:

a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.

b. DEBUG-NAME contains procedure-name-1.

(20) If reference to procedure-name-1 in the INPUT or OUTPUT phrase of a SORT or MERGE statement causes the debugging section to be executed, the following conditions exist:

a. DEBUG-LINE identifies the SORT or MERGE statement that references procedure-name-1.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains:

1. If the reference to procedure-name-1 is in the INPUT phrase of a SORT statement, 'SORT INPUT'.

2. If the reference to procedure-name-1 is in the OUTPUT phrase of a SORT statement, 'SORT OUTPUT'.

3. If the reference to procedure-name-1 is in the OUTPUT phrase of a MERGE statement,'MERGE OUTPUT'.

(21) If the transfer to control from the control mechanism associated with a PERFORM statement caused the debugging section associated with procedure-name-1 to be executed, the following conditions exist:

a. DEBUG-LINE identifies the PERFORM statement that references procedure-name-1.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains 'PERFORM LOOP'.

(22) If procedure-name-1 is a USE procedure that is to be executed, the following conditions exist:

a. DEBUG-LINE identifies the statement that causes execution of the USE procedure.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains 'USE PROCEDURE'.

(23) If an implicit transfer of control from the previous sequential paragraph to procedure-name-1 causes the debugging section to be executed, the following conditions exist:

a. DEBUG-LINE identifies the previous statement.

b. DEBUG-NAME contains procedure-name-1.

c. DEBUG-CONTENTS contains 'FALL THROUGH'.

(24) If references to file-name-1, cd-name-1 causes the debugging section to be executed, then:

a. DEBUG-LINE identifies the source statement that references file-name-1, cd-name-1.

b. DEBUG-NAME contains the name of file-name-1, cd-name-1.

c. For READ, DEBUG-CONTENTS contains the entire record read.

d. For all other references to file-name-1, DEBUG-CONTENTS contains spaces.

e. For any reference to cd-name-1, DEBUG-CONTENTS contains the contents of the area associated with the cd-name.

(25) If a reference to identifier-1 causes the debugging section to be executed, then:

a. DEBUG-LINE identifies the source statement that references identifier-1,

b. DEBUG-NAME contains the name of identifier-1, and

c. DEBUG-CONTENTS contains the contents of the data item referenced by identifier-1 at the time that control passes to the debugging section (see general rules 6 and 7).

3.2 DEBUGGING LINES

A debugging line is any line with a 'D' in the indicator area of the line. Any debugging line that consists solely of spaces from margin A to margin R is considered the same as a blank line.

The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

A debugging line will be considered to have all the characteristics of a comment line, if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain a 'D' in the indicator area, and character-strings may not be broken across two lines.

A debugging line is only permitted in the program after the OBJECT-COMPUTER paragraph.

1. INTRODUCTION TO THE INTER-PROGRAM COMMUNICATION MODULE

1.1 FUNCTION

The Inter-Program Communication module provides a facility by which a program can communicate with one or more programs. This communication is provided by: (a) the ability to transfer control from one program to another within a run unit and (b) the ability for both programs to have access to the same data items.

1.2 LEVEL CHARACTERISTICS

Inter-Program Communication Level 1 provides a capability to transfer control to one or more programs whose names are known at compile time and for the sharing of data among such programs.

Additionally Inter-Program Communication Level 2 provides the capability to transfer control to one or more programs whose names are not known at compile time as well as the ability to determine the availability of object time memory for the program to which control is being passed.

2. DATA DIVISION IN THE INTER-PROGRAM COMMUNICATION MODULE

2.1 LINKAGE SECTION

The Linkage Section in a program is meaningful if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. No space is allocated in the program for data items referenced by data-names in the Linkage Section of that program. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of the called program only if they are specified as operands of the USING phrase of the Procedure Division header or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous data items and/or record description entries.

Each Linkage Section record-name and noncontiguous item name must be unique within the called program since it cannot be qualified. Data items defined in the Linkage Section of the called program must not be associated with data items defined in the Report Section of the calling program.

Of those items defined in the Linkage Section only data-name-1, data-name-2, ... in the USING phrase of the Procedure Division header, data items subordinate to these data-names, and condition-names and/or index-names associated with such data-names and/or subordinate data items, may be referenced in the Procedure Division.

2.1.1 Noncontiguous Linkage Storage

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these data items is defined in a separate data description entry which begins with the special level-number 77.

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

a. level-number 77
b. data-name
c. 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.

2.1.2 Linkage Records

Data elements in the Linkage Section which bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause which is used in an input or output record description can be used in a Linkage Section.

2.1.3 Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

3. PROCEDURE DIVISION IN THE INTER-PROGRAM COMMUNICATION MODULE

3.1 THE 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] ...] .

The USING phrase is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

Each of the operands in the USING phrase of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

Within a called program, Linkage Section data items are processed according to their data descriptions given in the called program.

When the USING phrase is present, the object program operates as if data-name-1 of the Procedure Division header in the called program and data-name-1 in the USING phrase of the CALL statement in the calling program refer to a single set of data that is equally available to both the called and calling programs. Their descriptions must define an equal number of character positions; however, they need not be the same name. In like manner, there is an equivalent relationship between data-name-2, ..., in the USING phrase of the called program and data-name-2, ..., in the USING phrase of the CALL statement in the calling program. A data-name must not appear more than once in the USING phrase in the Procedure Division header of the called program; however, a given data-name may appear more than once in the same USING phrase of a CALL statement.

If the USING phrase is specified, the INITIAL clause must not be present in any CD entry. (See syntax rule 2 of the communication description entry on page XIII-4.)

3.2 THE CALL STATEMENT

3.2.1 Function

The CALL statement causes control to be transferred from one object program to another, within the run unit.

3.2.2 General Format

```
CALL { identifier-1 } [ USING data-name-1 [, data-name-2] ... ]
     { literal-1    }

[; ON OVERFLOW imperative-statement]
```

3.2.3 Syntax Rules

(1) Literal-1 must be a nonnumeric literal.

(2) Identifier-1 must be defined as an alphanumeric data item such that its value can be a program name.

(3) The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program and the number of operands in each USING phrase must be identical.

(4) Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, Communication Section, or Linkage Section, and must have a level-number of 01 or 77. Data-name-1, data-name-2, ..., may be qualified when they reference data items defined in the File Section or the Communication Section.

3.2.4 General Rules

(1) The program whose name is specified by the value of literal-1 or identifier-1 is the called program; the program in which the CALL statement appears is the calling program.

(2) The execution of a CALL statement causes control to pass to the called program.

(3) A called program is in its initial state the first time it is called within a run unit and the first time it is called after a CANCEL to the called program.

On all other entries into the called program, the state of the program remains unchanged from its state when last exited. This includes all data fields, the status and positioning of all files, and all alterable switch settings.

(4) If during the execution of a CALL statement, it is determined that the available portion of object time memory is incapable of accommodating the program specified in the CALL statement and the ON OVERFLOW phrase is specified, no action is taken and the imperative-statement is executed.

If the above condition exists and the ON OVERFLOW phrase is not specified, the effects of the CALL statement are defined by the implementor.

(5) Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

(6) The data-names, specified by the USING phrase of the CALL statement, indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the data-names in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Corresponding data-names refer to a single set of data which is available to the called and calling program. The correspondence is positional, not by name. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

(7) The CALL statement may appear anywhere within a segmented program. The implementor must provide all controls necessary to insure that the proper logic flow is maintained. Therefore, when a CALL statement appears in a section with a segment-number greater than or equal to 50, that segment is in its last used state when the EXIT PROGRAM statement returns control to the calling program.

3.3 THE CANCEL STATEMENT

3.3.1 Function

The CANCEL statement releases the memory areas occupied by the referred to program.

3.3.2 General Format

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

3.3.3 Syntax Rules

(1) Literal-1, literal-2, ..., must each be a nonnumeric literal.

(2) Identifier-1, identifier-2, ..., must each be defined as an alphanumeric data item such that its value can be a program name.

3.3.4 General Rules

(1) Subsequent to the execution of a CANCEL statement, the program referred to therein ceases to have any logical relationship to the run unit in which the CANCEL statement appears. A subsequently executed CALL statement naming the same program will result in that program being initiated in its initial state. The memory areas associated with the named programs are released so as to be made available for disposition by the operating system.

(2) A program named in the CANCEL statement must not refer to any program that has been called and has not yet executed an EXIT PROGRAM statement.

(3) A logical relationship to a cancelled subprogram is established only by execution of a subsequent CALL statement.

(4) A called program is cancelled either by being referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

(5) No action is taken when a CANCEL statement is executed naming a program that has not been called in this run unit or has been called and is at present cancelled. Control passes to the next statement.

3.4 THE EXIT PROGRAM STATEMENT

3.4.1 Function

The EXIT PROGRAM statement marks the logical end of a called program.

3.4.2 General Format

EXIT PROGRAM.

3.4.3 Syntax Rules

(1) The EXIT PROGRAM statement must appear in a sentence by itself.

(2) The EXIT PROGRAM sentence must be the only sentence in the paragraph.

3.4.4 General Rules

(1) An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program which is not called behaves as if the statement were an EXIT statement. (See page II-64, The EXIT Statement.)

1. INTRODUCTION TO THE COMMUNICATION MODULE

1.1 FUNCTION

The Communication module provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System with local and remote communication devices.

1.2 LEVEL CHARACTERISTICS

Communication Level 1 does not provide the full COBOL facility for the CD entry as specified in the formats for this module. In the Procedure Division, Level 1 provides limited capabilities for the ENABLE, DISABLE, RECEIVE and SEND statements, as specified in the formats of this module. There is also a provision for determining the number of messages in an input queue.

Communication Level 2 provides full facility for the CD entry as specified in the formats of this module. Within the Procedure Division, full capabilities are provided for the ENABLE, DISABLE, RECEIVE and SEND statements, as specified in the formats for this module. The additional features available in Level 2 include: partial messages, segmented messages, multiple destination message processing, and program invocation by the MCS as specified by the INITIAL CD.

2. DATA DIVISION IN THE COMMUNICATION MODULE

2.1 COMMUNICATION SECTION

In a COBOL program the communication description entries (CD) represent the highest level of organization in the Communication Section. The Communication Section header is followed by a communication description entry consisting of a level indicator (CD), a data-name and a series of independent clauses. These clauses indicate the queues and sub-queues, the message date and time, the source, the text length, the status and end keys, and message count of input. These clauses specify the destination count, the text length, the status and error keys, and destinations for output. The entry itself is terminated by a period. These record areas may be implicitly redefined by user-specified record description entries following the various communication description clauses.

2.2 THE COMMUNICATION DESCRIPTION - COMPLETE ENTRY SKELETON

2.2.1 Function

The communication description specifies the interface area between the MCS and a COBOL program.

2.2.2 General Format

Format 1

```
CD  cd-name;
                                  [ [ [; SYMBOLIC QUEUE IS data-name-1]
                                        [; SYMBOLIC SUB-QUEUE-1 IS data-name-2]
                                        [; SYMBOLIC SUB-QUEUE-2 IS data-name-3]
                                        [; SYMBOLIC SUB-QUEUE-3 IS data-name-4]
                                        [; MESSAGE DATE IS data-name-5 ]
                                        [; MESSAGE TIME IS data-name-6 ]
                                        [; SYMBOLIC SOURCE IS data-name-7]
                                        [; TEXT LENGTH IS data-name-8]
                                        [; END KEY IS data-name-9]
                                        [; STATUS KEY IS data-name-10]
                                        [; MESSAGE COUNT IS data-name-11]]
                                    [ data-name-1, data-name-2, ..., data-name-11 ] ]
   FOR [INITIAL] INPUT                                                              .
```

Format 2

```
CD  cd-name; FOR OUTPUT
    [; DESTINATION COUNT IS data-name-1]
    [; TEXT LENGTH IS data-name-2]
    [; STATUS KEY IS data-name-3]
    [; DESTINATION TABLE OCCURS integer-2 TIMES
         [; INDEXED BY index-name-1  [, index-name-2]...] ]
    [; ERROR KEY IS data-name-4 ]
    [; SYMBOLIC DESTINATION IS data-name-5 ] .
```

2.2.3 Syntax Rules

FORMAT 1

(1) A CD must appear only in the Communication Section.

(2) Within a single program, the INITIAL clause may be specified in only one CD. The INITIAL clause must not be used in a program that specifies the USING phrase of the Procedure Division Header. (See page XII-4, The Procedure Division Header.)

(3) Except for the INITIAL clause, the optional clauses may be written in any order.

(4) If neither option in the format is specified, a level 01 data description entry must follow the CD description entry. Either option may be followed by a level 01 data description entry.

(5) For each input CD, a record area of 87 contiguous standard data format characters is allocated. This record area is defined to the MCS as follows:

a. The SYMBOLIC QUEUE clause defines data-name-1 as the name of an elementary alphanumeric data item of 12 characters occupying positions 1-12 in the record.

b. The SYMBOLIC SUB-QUEUE-1 clause defines data-name-2 as the name of an elementary alphanumeric data item of 12 characters occupying positions 13-24 in the record.

c. The SYMBOLIC SUB-QUEUE-2 clause defines data-name-3 as the name of an elementary alphanumeric data item of 12 characters occupying positions 25-36 in the record.

d. The SYMBOLIC SUB-QUEUE-3 clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 37-48 in the record.

e. The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 49-54 in the record.

f. The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of 8 digits without an operational sign occupying character positions 55-62 in the record.

g. The SYMBOLIC SOURCE clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63-74 in the record.

h. The TEXT LENGTH clause defines data-name-8 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 75-78 in the record.

i. The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of 1 character occupying position 79 in the record.

j. The STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of 2 characters occupying positions 80-81 in the record.

k. The MESSAGE COUNT clause defines data-name-11 as the name of an elementary data item whose implicit description is that of an integer of 6 digits without an operational sign occupying character positions 82-87 in the record.

The second option may be used to replace the above clauses by a series of data-names which, taken in order, correspond to the data-names defined by these clauses.

Use of either option results in a record whose implicit description is equivalent to the following:

```
          IMPLICIT DESCRIPTION                          COMMENT

01  data-name-0.

    02  data-name-1    PICTURE X(12).          SYMBOLIC QUEUE
    02  data-name-2    PICTURE X(12).          SYMBOLIC SUB-QUEUE-1
    02  data-name-3    PICTURE X(12).          SYMBOLIC SUB-QUEUE-2
    02  data-name-4    PICTURE X(12).          SYMBOLIC SUB-QUEUE-3
    02  data-name-5    PICTURE 9(06).          MESSAGE DATE
    02  data-name-6    PICTURE 9(08).          MESSAGE TIME
    02  data-name-7    PICTURE X(12).          SYMBOLIC SOURCE
    02  data-name-8    PICTURE 9(04).          TEXT LENGTH
    02  data-name-9    PICTURE X.              END KEY
    02  data-name-10   PICTURE XX.             STATUS KEY
    02  data-name-11   PICTURE 9(06).          MESSAGE COUNT
```

NOTE: In the above, the information under 'COMMENT' is for clarification and is not part of the description.

(6) Record description entries following an input CD implicitly redefine this record and must describe a record of exactly 87 characters. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. However, the MCS will always reference the record according to the data descriptions defined in syntax rule 5.

(7) Data-name-1, data-name-2, ..., data-name-11 must be unique within the CD. Within this series, any data-name may be replaced by the reserved word FILLER.

FORMAT 2

(8) A CD must appear only in the Communication Section.

(9) If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.

(10) For each output CD, a record area of contiguous standard data format characters is allocated according to the following formula: (10 plus 13 times integer-2).

a. The DESTINATION COUNT clause defines data-name-1 as the name of a data item whose implicit description is that of an integer without an operational sign occupying character positions 1-4 in the record.

b. The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign occupying character positions 5-8 in the record.

c. The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of 2 characters occupying positions 9-10 in the record.

d. Character positions 11-23 and every set of 13 characters thereafter will form table items of the following description:

1) The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of 1 character.

2) The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

Use of the above clauses results in a record whose implicit description is equivalent to the following:

```
          IMPLICIT DESCRIPTION                    COMMENT

01  data-name-0.
    02  data-name-1  PICTURE 9(04).              DESTINATION COUNT
    02  data-name-2  PICTURE 9(04).              TEXT LENGTH
    02  data-name-3  PICTURE XX.                 STATUS KEY
    02  data-name  OCCURS integer-2 TIMES.       DESTINATION TABLE
        03  data-name-4  PICTURE X.              ERROR KEY
        03  data-name-5  PICTURE X(12).          SYMBOLIC DESTINATION
```

NOTE: In the above, the information under 'COMMENT' is for clarification and is not part of the description.

(11) Record descriptions following an output CD implicitly redefine this record. Multiple redefinitions of this record are permitted; however, only the first redefinition may contain VALUE clauses. However, the MCS will always reference the record according to the data descriptions defined in syntax rule 10.

(12) Data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.

(13) If the DESTINATION TABLE OCCURS clause is not specified, one (1) ERROR KEY and one (1) SYMBOLIC DESTINATION area is assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.

(14) If the DESTINATION TABLE OCCURS clause is specified, data-name-4 and data-name-5 may only be referred to by subscripting or indexing.

(15) In Level 1, the value of the data item referenced by data-name-1 and integer-2 must be 1. In Level 2, there is no restriction on the value of the data item referenced by data-name-1 and integer-2.

2.2.4 General Rules

FORMAT 1

(1) The input CD information constitutes the communication between the MCS and the program as information about the message being handled. This information does not come from the terminal as part of the message.

(2) The contents of the data items referenced by data-name-2, data-name-3, and data-name-4, when not being used must contain spaces.

(3) The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 contain symbolic names designating queues, sub-queues, ... respectively. All symbolic names must follow the rules for the formation of system-names, and must have been previously defined to the MCS.

(4) A RECEIVE statement causes the serial return of the 'next' message or a portion of a message from the queue as specified by the entries in the CD.

If during the execution of a RECEIVE statement, a message from a more specific source is needed, the contents of the data item referenced by data-name-1 can be made more specific by the use of the contents of the data items referenced by data-name-2, data-name-3, and in turn data-name-4. When a given level of the queue structure is specified, all higher levels must also be specified.

If less than all the levels of the queue hierarchy are specified, the MCS determines the 'next' message or portion of a message to be accessed.

After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 will contain the symbolic names of all the levels of the queue structure.

(5) Whenever a program is scheduled by the MCS to process a message, the symbolic names of the queue structure that demanded this activity will be placed in the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause as applicable. In all other cases, the contents of the data items referenced by data-name-1 through data-name-4 of the CD associated with the INITIAL clause are initialized to spaces.

The symbolic names are inserted or the initialization to spaces is completed prior to the execution of the first Procedure Division statement.

The execution of a subsequent RECEIVE statement naming the same contents of the data items referenced by data-name-1 through data-name-4 will return the actual message that caused the program to be scheduled. Only at that time will the remainder of the CD be updated.

(6) If the MCS attempts to schedule a program lacking an INITIAL clause, the results are undefined.

(7) Data-name-5 has the format 'YYMMDD' (year, month, day). Its contents represent the date on which the MCS recognizes that the message is complete.

The contents of the data item referenced by data-name-5 are only updated by the MCS as part of the execution of a RECEIVE statement.

(8) The contents of data-name-6 has the format 'HHMMSSTT' (hours, minutes, seconds, hundredths of a second) and its contents represent the time at which the MCS recognizes that the message is complete.

The contents of the data item referenced by data-name-6 are only updated by the MCS as part of the execution of the RECEIVE statement.

(9) During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by data-name-7, the symbolic name of the communications terminal that is the source of the message being transferred. However, if the symbolic name of the communication terminal is not known to the MCS, the contents of the data item referenced by data-name-7 will contain spaces.

(10) The MCS indicates via the contents of the data item referenced by data-name-8 the number of character positions filled as a result of the execution of the RECEIVE statement. (See page XIII-17.)

(11) The contents of the data item referenced by data-name-9 are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

a. When the RECEIVE MESSAGE phrase is specified, then:

1. If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;

2. If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

3. If less than a message is transferred, the contents of the data item referenced by data-name-9 are set to 0.

b. When the RECEIVE SEGMENT phrase is specified, then:

1. If an end of group has been detected, the contents of the data item referenced by data-name-9 are set to 3;

2. If an end of message has been detected, the contents of the data item referenced by data-name-9 are set to 2;

3. If an end of segment has been detected, the contents of the data item referenced by data-name-9 are set to 1;

4. If less than a message segment is transferred, the contents of the data item referenced by data-name-9 are set to 0.

c. When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of the data item referenced by data-name-9.

(12) The contents of the data item referenced by data-name-10 indicate the status condition of the previously executed RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT, or DISABLE INPUT statements.

The actual association between the contents of the data item referenced by data-name-10 and the status condition itself is defined in the table on page XIII-10.

(13) The contents of the data item referenced by data-name-11 indicate the number of messages that exist in a queue, sub-queue-1, ... The MCS updates the contents of the data item referenced by data-name-11 only as part of the execution of an ACCEPT statement with the COUNT phrase.

FORMAT 2

(14) The nature of the output CD information is such that it is not sent to the terminal, but constitutes the communication between the program and the MCS as information about the message being handled.

(15) During the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the contents of the data item referenced by data-name-1 will indicate to the MCS the number of symbolic destinations that are to be used from the area referenced by data-name-5.

The MCS finds the first symbolic destination in the first occurrence of the area referenced by data-name-5, the second symbolic destination in the second occurrence of the area referenced by data-name-5 ..., up to and including the occurrence of the area referenced by data-name-5 indicated by the contents of data-name-1.

If during the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement the value of the data item referenced by data-name-1 is outside the range of 1 through integer-2, an error condition is indicated and the execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is terminated.

(16) It is the responsibility of the user to insure that the value of the data item referenced by data-name-1 is valid at the time of execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

(17) As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 to be the user's indication of the number of leftmost character positions of the data item referenced by the associated SEND identifier from which data is to be transferred. (See page XIII-20.)

(18) Each occurrence of the data item referenced by data-name-5 contains a symbolic destination previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.

(19) The contents of the data item referenced by data-name-3 indicate the status condition of the previously executed SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

The actual association between the contents of the data item referenced by data-name-3 and the status condition itself is defined in the table on page XIII-10.

(20) If, during the execution of a SEND, an ENABLE OUTPUT, or a DISABLE OUTPUT statement, the MCS determines that any specified destination is unknown, the contents of the data item referenced by data-name-3 and all occurrences of the data items referenced by data-name-4 are updated.

The contents of the data item referenced by data-name-4 when equal to 1 indicate that the associated value in the area referenced by data-name-5 has not been previously defined to the MCS. Otherwise, the contents of the data item referenced by data-name-4 are set to zero (0).

ALL FORMATS

(21) For Level 1, the table below indicates the possible contents of the data items referenced by data-name-10 for Format 1 and by data-name-3 for Format 2 at the completion of each statement shown. An 'X' on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

RECEIVE	SEND	ACCEPT MESSAGE COUNT	ENABLE INPUT (without TERMINAL)	ENABLE OUTPUT	DISABLE INPUT (without TERMINAL)	DISABLE OUTPUT	STATUS KEY CODE	
X	X	X	X	X	X	X	00	No error detected. Action completed.
	X						10	Destination is disabled. Action completed.
	X			X		X	20	Destination unknown. No action taken for unknown destination. Data-name-4 (ERROR KEY) indicates unknown.
X		X	X		X		20	One or more queues or sub-queues unknown. No action taken.
	X			X		X	30	Content of DESTINATION COUNT invalid. No action taken.
			X	X	X	X	40	Password invalid. No enabling/disabling action taken.
	X						50	Character count greater than length of sending field. No action taken.

Communication Status Key Condition in Level 1

(22) For Level 2, the table below indicates the possible contents of the data items referenced by data-name-10 for Format 1 and by data-name-3 for Format 2 at the completion of each statement shown. An 'X' on a line in a statement column indicates that the associated code shown for that line is possible for that statement.

RECEIVE	SEND	ACCEPT MESSAGE COUNT	ENABLE INPUT (without TERMINAL)	ENABLE INPUT (with TERMINAL)	ENABLE OUTPUT	DISABLE INPUT (without TERMINAL)	DISABLE INPUT (with TERMINAL)	DISABLE OUTPUT	STATUS KEY CODE	
X	X	X	X	X	X	X	X	X	00	No error detected. Action completed.
	X								10	One or more destinations are disabled. Action completed.
	X				X			X	20	One or more destinations unknown. Action completed for known destinations. No action taken for unknown destinations. Data-name-4 (ERROR KEY) indicates known or unknown.
X		X	X			X			20	One or more queues or sub-queues unknown. No action taken.
				X			X		20	The source is unknown. No action taken.
	X				X			X	30	Content of DESTINATION COUNT invalid. No action taken.
			X	X	X	X	X	X	40	Password invalid. No enabling/disabling action taken.
	X								50	Character count greater than length of sending field. No action taken.
	X								60	Partial segment with either zero character count or no sending area specified. No action taken.

Communication Status Key Condition in Level 2

3. PROCEDURE DIVISION IN THE COMMUNICATION MODULE

3.1 THE ACCEPT MESSAGE COUNT STATEMENT

3.1.1 Function

The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available.

3.1.2 General Format

ACCEPT cd-name MESSAGE COUNT

3.1.3 Syntax Rules

(1) Cd-name must reference an input CD.

3.1.4 General Rules

(1) The ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in a queue, sub-queue-1, ...

(2) Upon execution of the ACCEPT MESSAGE COUNT statement, the contents of the area specified by a communication description entry must contain at least the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-10 (STATUS KEY) and data-name-11 (MESSAGE COUNT) of the area associated with the communication entry to be appropriately updated. (See page XIII-3, The Communication Description - Complete Entry Skeleton.)

3.2 THE DISABLE STATEMENT

3.2.1 Function

The DISABLE statement notifies the MCS to inhibit data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

3.2.2 General Format

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

3.2.3 Syntax Rules

(1) Cd-name must reference an input CD when the INPUT phrase is specified.

(2) Cd-name must reference an output CD when the OUTPUT phrase is specified.

(3) Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric.

3.2.4 General Rules

(1) The DISABLE statement provides a logical disconnection between the MCS and the specified sources or destinations. When this logical disconnection is already in existence, or is to be handled by some other means external to this program, the DISABLE statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the DISABLE statement.

(2) When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all queues and sub-queues is deactivated. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful.

(3) When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queues and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are deactivated.

(4) When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are deactivated.

(5) Literal-1 or the contents of the data-name referenced by identifier-1 will be matched with a password built into the system. The DISABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 matches the system password. When literal-1 or the contents of the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

The MCS must be capable of handling a password of from one to ten characters inclusive.

(6) The MCS will insure that the execution of a DISABLE statement will cause the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement will never cause the remaining portion of the message to be terminated during transmission to or from a terminal.

3.3 THE ENABLE STATEMENT

3.3.1 Function

The ENABLE statement notifies the MCS to allow data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

3.3.2 General Format

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

3.3.3 Syntax Rules

(1) Cd-name must reference an input CD when the INPUT phrase is specified.

(2) Cd-name must reference an output CD when the OUTPUT phrase is specified.

(3) Literal-1 or the contents of the data item referenced by identifier-1 must be defined as alphanumeric.

3.3.4 General Rules

(1) The ENABLE statement provides a logical connection between the MCS and the specified sources or destinations. When this logical connection is already in existence, or is to be handled by some other means external to this program, the ENABLE statement is not required in this program. The logical path for the transfer of data between the COBOL programs and the MCS is not affected by the ENABLE statement.

(2) When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all associated queues and sub-queues which are already enabled is activated. Only the contents of the data item referenced by data-name-7 (SYMBOLIC SOURCE) of the area referenced by cd-name are meaningful to the MCS.

(3) When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queue and sub-queues specified by the contents of data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name are activated.

(4) When the OUTPUT phrase is specified, the logical path for destination, or the logical paths for all destinations, specified by the contents of the data item referenced by data-name-5 (SYMBOLIC DESTINATION) of the area referenced by cd-name are activated.

(5) Literal-1 or the contents of the data item referenced by identifier-1 will be matched with a password built into the system. The ENABLE statement will be honored only if literal-1 or the contents of the data item referenced by identifier-1 match the system password. When literal-1 or the contents of

the data item referenced by identifier-1 do not match the system password, the value of the STATUS KEY item in the area referenced by cd-name is updated.

The MCS must be capable of handling a password of from one to ten characters inclusive.

3.4 THE RECEIVE STATEMENT

3.4.1 Function

The RECEIVE statement makes available to the COBOL program a message, message segment, or a portion of a message or segment, and pertinent information about that data from a queue maintained by the Message Control System. The RECEIVE statement allows for a specific imperative statement when no data is available.

3.4.2 General Format

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

3.4.3 Syntax Rules

(1) Cd-name must reference an input CD.

3.4.4 General Rules

(1) The contents of the data items specified by data-name-1 (SYMBOLIC QUEUE) through data-name-4 (SYMBOLIC SUB-QUEUE-3) of the area referenced by cd-name designate the queue structure containing the message. (See page XIII-3, The CD Entry.)

(2) The message, message segment, or portion of a message or segment is transferred to the receiving character positions of the area referenced by identifier-1 aligned to the left without space fill.

(3) When during the execution of a RECEIVE statement, the MCS makes data available in the data item referenced by identifier-1, control is transferred to the next executable statement, whether or not the NO DATA phrase is specified.

(4) When, during the execution of a RECEIVE statement, the MCS does not make data available in the data item referenced by identifier-1:

a. If the NO DATA phrase is specified, the RECEIVE operation is terminated with the indication that action is complete (see general rule 5), and the imperative statement in the NO DATA phrase is executed.

b. If the NO DATA phrase is not specified, execution of the object program is suspended until data is made available in the data item referenced by identifier-1.

c. If one or more queues or sub-queues is unknown to the MCS, control passes to the next executable statement, whether or not the NO DATA phrase is specified. (See page XIII-10 and XIII-11, Communication Status Key Condition.)

(5) The data items identified by the input CD are appropriately updated by the Message Control System at each execution of a RECEIVE statement. (See page XIII-3, The CD Entry.)

(6) A single execution of a RECEIVE statement never returns to the data item referenced by identifier-1 more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used). However, the MCS does not pass any portion of a message to the object program until the entire message is available in the input queue, even if the SEGMENT phrase of the RECEIVE statement is specified.

(7) When the MESSAGE phrase is used, end of segment indicators are ignored, and the following rules apply to the data transfer:

a. If a message is the same size as the area referenced by identifier-1, the message is stored in the area referenced by identifier-1.

b. If a message size is less than the area referenced by identifier-1, the message is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

c. If a message size is greater than the area referenced by identifier-1, the message fills the area referenced by identifier-1 left to right starting with the leftmost character of the message. In Level 1, the disposition of the remainder of the message is undefined. In Level 2, the remainder of the message can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue, sub-queue ... . The remainder of the message, for the purposes of applying rules 7a, 7b, and 7c, is treated as a new message.

(8) When the SEGMENT phrase is used, the following rules apply:

a. If a segment is the same size as the area referenced by identifier-1, the segment is stored in the area referenced by identifier-1.

b. If the segment size is less than the area referenced by identifier-1, the segment is aligned to the leftmost character position of the area referenced by identifier-1 with no space fill.

c. If a segment size is greater than the area referenced by identifier-1, the segment fills the area referenced by identifier-1 left to right starting with the leftmost character of the segment. The remainder of the segment can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements calling out the same queue, sub-queue ... . The remainder of the segment, for the purposes of applying rules 8a, 8b, and 8c, is treated as a new segment.

d. If the text to be accessed by the RECEIVE statement has associated with it an end of message indicator or end of group indicator, the existence of an end of segment indicator associated with the test is implied and the text is treated as a message segment according to general rule 8.

(9) Once the execution of a RECEIVE statement has returned a portion of a message, only subsequent execution of RECEIVE statements in that run unit can cause the remaining portion of the message to be returned.

(10) After the execution of a STOP RUN statement, the disposition of a remaining portion of a message partially obtained in that run unit is defined by the implementor. (See page II-85, The STOP Statement.)

3.5 THE SEND STATEMENT

3.5.1 Function

The SEND statement causes a message, a message segment, or a portion of a message or segment to be released to one or more output queues maintained by the Message Control System.

3.5.2 General Format

Format 1

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

Format 2

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

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

3.5.3 Syntax Rules

(1) Cd-name must reference an output CD.

(2) Identifier-2 must reference a one character integer without an operational sign.

(3) When identifier-3 is used in the ADVANCING phrase, it must be the name of an elementary integer item.

(4) When the mnemonic-name phrase is used, the name is identified with a particular feature specified by the implementor. The mnemonic-name is defined in the SPECIAL-NAMES paragraph of the Environment Division.

(5) Integer or the value of the data item referenced by identifier-3 may be zero.

3.5.4 General Rules

ALL FORMATS

(1) When a receiving communication device (printer, display screen, card punch, etc.) is oriented to a fixed line size:

a. Each message or message segment will begin at the leftmost character position of the physical line.

b. A message or message segment that is smaller than the physical line size is released so as to appear space-filled to the right.

c. Excess characters of a message or message segment will not be truncated. Characters will be packed to a size equal to that of the physical line and then outputted to the device. The process continues on the next line with the excess characters.

(2) When a receiving communication device (paper tape punch, another computer, etc.) is oriented to handle variable length messages, each message or message segment will begin on the next available character position of the communications device.

(3) As part of the execution of a SEND statement, the MCS will interpret the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name to be the user's indication of the number of left-most character positions of the data item referenced by identifier-1 from which data is to be transferred.

If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are zero, no characters of the data item referenced by identifier-1 are transferred.

If the contents of the data item referenced by data-name-2 (TEXT LENGTH) of the area referenced by cd-name are outside the range of zero through the size of the data item referenced by identifier-1 inclusive, an error is indicated by the value of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred. (See pages XIII-10 and XIII-11, Communication Status Key Condition.)

(4) As part of the execution of a SEND statement, the contents of the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name is updated by the MCS. (See page XIII-3, The CD Entry.)

(5) The effect of having special control characters within the contents of the data item referenced by identifier-1 is undefined.

(6) A single execution of a SEND statement for Format 1 releases only a single portion of a message or of a message segment to the MCS.

A single execution of a SEND statement of Format 2 never releases to the MCS more than a single message or a single message segment as indicated by the contents of the data item referenced by identifier-2 or by the specified indicator ESI, EMI or EGI.

However, the MCS will not transmit any portion of a message to a communications device until the entire message is placed in the output queue.

(7) During the execution of the run unit, the disposition of a portion of a message not terminated by an EMI or EGI is undefined. However, the message does not logically exist for the MCS and hence cannot be sent to a destination.

After the execution of a STOP RUN statement, any portion of a message transferred from the run unit via a SEND statement, but not terminated by an EMI or EGI, is purged from the system. Thus no portion of the message is sent.

(8) Once the execution of a SEND statement has released a portion of a message to the MCS, only subsequent execution of SEND statements in the same run unit can cause the remaining portion of the message to be released.

FORMAT 2

(9) The contents of the data item referenced by identifier-2 indicate that the contents of the data item referenced by identifier-1 are to have associated with it an end of segment indicator, an end of message indicator or an end of transmission indicator according to the following schedule:

If the content of the data item referenced by identifier-2 is	then the contents of data item referenced by identifier-1 have associated with it	which means
'0'	no indicator	no indicator
'1'	ESI	an end of segment indicator
'2'	EMI	an end of message indicator
'3'	EGI	an end of group indicator

Any character other than '1', '2', or '3' will be interpreted as '0'.

If the content of the data item referenced by identifier-2 is other than '1', '2', or '3', and identifier-1 is not specified, then an error is indicated by the value in the data item referenced by data-name-3 (STATUS KEY) of the area referenced by cd-name, and no data is transferred.

(10) The ESI indicates to the MCS that the message segment is complete. The EMI indicates to the MCS that the message is complete.

The EGI indicates to the MCS that the group of messages is complete. The implementor will specify the interpretation that is given to the EGI by the MCS.

The MCS will recognize these indications and establish whatever is necessary to maintain group, message, and segment control.

(11) The hierarchy of ending indicators is EGI, EMI, and ESI. An EGI need not be preceded by an ESI or EMI. An EMI need not be preceded by an ESI.

(12) The ADVANCING phrase allows control of the vertical positioning of each message or message segment on a communication device where vertical positioning is applicable. If vertical positioning is not applicable on the device, the MCS will ignore the vertical positioning specified or implied.

(13) If identifier-2 is specified and the content of the data item referenced by identifier-2 is zero, the ADVANCING phrase is ignored by the MCS.

(14) On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing will be provided by the implementor to act as if the user had specified AFTER ADVANCING 1 LINE.

(15) If the ADVANCING phrase is implicitly or explicitly specified and vertical positioning is applicable, the following rules apply:

a. If identifier-3 or integer is specified, characters transmitted to the communication device will be repositioned vertically downward the number of lines equal to the value associated with the data item referenced by identifier-3 or integer.

b. If mnemonic-name is specified, characters transmitted to the communication device will be positioned according to the rules specified by the implementor for that communication device.

c. If the BEFORE phrase is used, the message [or message segment] is represented on the communication device before vertical repositioning according to general rules 15a and 15b above.

d. If the AFTER phrase is used, the message [or message segment] is represented on the communication device after vertical repositioning according to general rules 15a and 15b above.

e. If PAGE is specified, characters transmitted to the communication device will be represented on the device before or after (depending upon the phrase used) the device is repositioned to the next page. If PAGE is specified but page has no meaning in conjunction with a specific device, then advancing will be provided by the implementor to act as if the user had specified BEFORE or AFTER (depending upon the phrase used) ADVANCING 1 LINE.

1. APPENDIX A: THE HISTORY OF COBOL

1.1 ORGANIZATION OF COBOL EFFORT

On May 28 and 29, 1959, a meeting was held in the Pentagon for the purpose of considering both the desirability and the feasibility of establishing a common language for the programming of electronic computers for business-type applications. Representatives from users, both in private industry and in government, computer manufacturers, and other interested parties were present. The group agreed that the project should be undertaken. The COnference on DAta SYstems Languages (CODASYL) developed out of this meeting.

The original COBOL specification resulted from the work of a committee of CODASYL. By September, 1959, this committee had specified a language which they considered superior to existing language-compiler systems. This language specification was further modified and by December, 1959, COBOL existed as a language that was not identified with any manufacturer and therefore presented advantages for both government and private industry users.

1.1.1 Initial Organization

The product of phase I of COBOL development was a report published in April of 1960 by the Government Printing Office entitled "COBOL--A Report to the Conference on Data Systems Languages, including Initial Specifications for a Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers". The language described in this report has since become known as COBOL-60.

1.1.2 The COBOL Maintenance Committee

The Executive Committee of CODASYL recognized that the task of defining COBOL was a continuing one and that the language had to be maintained and improved. To this end, the COBOL Maintenance Committee was created and charged with the task of answering questions arising from users and implementors of the language and making definitive modifications, including additions, clarifications, and changes to the language.

The Maintenance Committee was comprised of a Users Group and a Manufacturers Group. These groups met together but voted on proposals separately.

In order to devote concentrated attention to publishing a revised and updated "COBOL 60", the Executive Committee created a Special Task Group. The product of this task group was the COBOL-61 manual, which was published by the Government Printing Office in mid-1961.

The next official COBOL publication was also the product of the Maintenance Committee and was called COBOL-61 Extended; released in mid-1963.

1.1.3 The COBOL Committee

In January, 1964, the COBOL Maintenance Committee was reorganized to provide a true industry group and to broaden its scope of activities. The separate user and manufacturer groups were combined into the COBOL Committee consisting of three subcommittees: the Language Subcommittee, the Evaluation Subcommittee, and the Publication Subcommittee.

The Language Subcommittee's function was much the same as was that of the former COBOL Maintenance Committee, namely, the maintenance and further development of COBOL. In addition it carried on liaison with the United States of America Standards Institute (USASI: formerly the American Standards Association -- ASA) and the International Organization for Standardization (ISO) in their work concerning the development of proposed COBOL Standards.

The Publication Subcommittee was charged with the production of official COBOL publications and liaison with USASI as to the content of the COBOL Information Bulletin (CIB). The CIB is a collection of material relating to COBOL, distributed to the COBOL community by USASI.

The Evaluation Subcommittee's task was the analysis and evaluation of compiler implementations and user surveys. This subcommittee provided information to the COBOL Committee regarding the use of COBOL.

The product of the COBOL Committee was the manual, "COBOL, Edition 1965".

1.1.4 Programming Language Committee

In July, 1968, the CODASYL Executive Committee adopted a revised constitution which accomplished certain needed organizational changes in an effort to stabilize and improve the methods of achieving CODASYL objectives. CODASYL now consists of four standing committees: the Executive Committee, the Programming Language Committee, the Planning Committee, and the Systems Committee.

With the formation of the Programming Language Committee (PLC) the former COBOL Language Subcommittee was elevated to full committee status, and its chairman became a member of the Executive Committee.

The purpose and objectives of PLC include and extend those of the former COBOL Language Subcommittee. The objectives are to make possible: compatible, uniform, source programs and object results, with continued reduction in the number of changes necessary for conversion or interchange of source programs and data. The PLC concentrates its efforts in the area of tools, techniques and ideas aimed at the programmer.

The Programming Language Committee is responsible for the presentation of the COBOL Journal of Development.

1.2 EVOLUTION OF COBOL

1.2.1 COBOL-60

COBOL-60, the first version of the language published, proved that the concept of a common business oriented language was indeed practical.

1.2.2 COBOL-61

COBOL-61, the second official version of COBOL, was not completely compatible with COBOL-60. The changes were in areas such as organization of the Procedure Division rather than the addition of any major functions. The avowed goal of CODASYL in terms of successive versions of the language was to make changes of an evolutionary rather than revolutionary nature. This version was generally implemented and was the basis for many COBOL compilers.

1.2.3 COBOL-61 Extended

This version of COBOL was generally compatible with COBOL-61. The term 'generally' must be used, not because of any basic changes in the philosophy or organization of the language, but because certain arithmetic extensions and general clarifications did make the syntax for certain statements and entries different from those in COBOL-61.

COBOL-61 Extended, then, was generally COBOL-61 with the following major additions and modifications:

a. The addition of the Sort feature,

b. The addition of the Report Writer option,

c. The modification of the arithmetics to include multiple receiving fields and to add the CORRESPONDING option to the ADD and SUBTRACT statements, and

d. The inclusion of various clarifications.

1.2.4 COBOL, Edition 1965

This version of COBOL included COBOL-61 Extended plus certain additions and modifications.

The major changes incorporated in COBOL, Edition 1965, were:

a. The inclusion of a series of options to provide for the reading, writing and processing of mass storage files,

b. The addition of the Table Handling feature which includes indexing and search options,

c. The modification of the specifications to delete the requirement for specific error diagnostic messages,

d. The deletion of the terms "Required" and "Elective", and

e. The inclusion of various clarifications.

1.2.5 COBOL, 1968

This version of COBOL, published in the Journal of Development, was based on COBOL, Edition 1965, with certain additions and deletions.

The major changes incorporated in COBOL, 1968, were:

a. The inclusion of inter-program communication and the concept of a run unit,

b. The elimination of redundant editing clauses and certain data clauses more succinctly expressed by the PICTURE clause,

c. An improved COPY specification for all divisions except the Identification Division and the elimination of the INCLUDE statement,

d. The inclusion of a hardware independent means of specifying and testing for page overflow conditions,

e. The elimination of type 4 abbreviations,

f. The elimination of the DEFINE statement,

g. The inclusion of a remainder option for the DIVIDE statement,

h. The deletion of NOTE and REMARKS in favor of a general comment capability for all divisions,

i. The inclusion of the SUSPEND statement as additional means of controlling graphic display devices,

j. The inclusion of additional abbreviations,

k. A revision of the EXAMINE statement to allow the specification of dynamic parameter values, and

l. The inclusion of various clarifications.

1.2.6 COBOL, 1969

This version of COBOL, published in the Journal of Development, is based on COBOL, 1968, with certain additions and deletions.

The major changes incorporated in COBOL, 1969, are:

a. The deletion of the EXAMINE statement and the inclusion of a more powerful statement, INSPECT, in its place,

b. The inclusion of a communication facility to permit input and output with communications devices,

c. The inclusion of the STRING and UNSTRING statements, to facilitate character string manipulation,

d. Deletion of the CONSTANT SECTION of the Data Division,

e. The inclusion of a compile-time page ejection facility,

f. The inclusion of a facility to access the system's date and time,

g. The inclusion of a SIGN clause as a means of specifying the position and mode of representation of the operational sign, and

h. The inclusion of various clarifications.

1.2.7 COBOL, 1970

This version of COBOL, published in the Journal of Development, is based on COBOL, 1969, with certain additions, deletions, and modifications.

The major changes incorporated in COBOL, 1970, are:

a. The deletion of the RANGE clause,

b. The inclusion of the INITIALIZE statement, to facilitate setting data items to values consistent with their data descriptions,

c. The inclusion of a debugging facility,

d. The inclusion of a merge facility,

e. A complete revision to the Report Writer function, and

f. The inclusion of various clarifications.

1.2.8. COBOL, 1973

This version of COBOL, published in the Journal of Development, is based on COBOL, 1970, with certain additions, deletions, and modifications.

The major changes incorporated in COBOL, 1973, are:

a. A revision and extension to the mass storage facility,

b. A clarification and extension to the COBOL library facility,

c. An enhancement of the INSPECT statement,

d. A revision to the file control entry for a sort or merge file which included the deletion of Format 3,

e. A revision to the RERUN facility,

f. The removal of the restriction on 77 level-numbers that they must precede 01 level-numbers,

g. The inclusion of a page advancing feature as part of the WRITE statement,

h. A clarification and enhancement of the COBOL language structure,

i. An enhancement of the LINAGE clause to permit specification of margins, and

j. The inclusion of various clarifications.

1.3 STANDARDIZATION OF COBOL

1.3.1 Initial Standardization Effort

American National Standards Committee on Computers and Information Processing, X3, was established in 1960 under the sponsorship of the Business Equipment Manufacturers Association. The X3 Committee in turn established the X3-4 Subcommittee to pursue standards in the area of Common Programming Languages. Subsequently, Working Group X3.4.4 with the title "Processor Specification and COBOL Standards" was established to pursue a COBOL standard. Part of the scope of X3.4.4 follows:

> "Standardization of COBOL and its characteristics, establishment of an X3.4 COBOL bulletin, publication of interpretations and clarifications, and the definition of test problems."

On December 17, 1962, invitations to an organizational meeting of X3.4.4 were sent to manufacturers and user groups who might be interested in participating in the establishment of a COBOL standard. The first meeting was held on January 15-16, 1963, in New York and the following program of work was accepted:

(1) Establish the X3.4 COBOL Information Bulletin (CIB) and provide for its broad publication.

(2) Ascertain the features of existing or proposed COBOL processors.

(3) Refer ambiguities to the COBOL Maintenance Committee for interpretation.

(4) Publish these interpretations in the CIB.

(5) Write test problems to test specific and combinatorial features of COBOL.

(6) Refer any new ambiguities which are revealed through the test problems to the COBOL Maintenance Committee.

(7) When appropriate, write and publish in the X3.4 CIB a proposed draft standard for COBOL and process it through the X3 Committee.

(8) When appropriate, publish proposed standard test problems for COBOL and process them through the X3 Committee.

(9) Review and augment these standards as necessary.

(10) Maintain close liaison with other standards bodies interested in COBOL.

The objective of the X3.4.4 Working Group was to produce a document which defined the American Standard* or standards for COBOL. The resulting standard language was to be based upon the specifications set out in the CODASYL publication.

*In August, 1966, the American Standards Association (ASA) became the USA Standards Institute (USASI). Then in the fall of 1969, the USA Standards Institute (USASI) became the American National Standards Institute (ANSI).

The criteria used to consider and evaluate various language elements for inclusion in the proposed standard included (not in order of importance):

(1) General usefulness, as determined by:
    a. Degree of implementation
    b. User acceptance
    c. User desires
    d. Experience

(2) Cost of implementation versus advantages of use.

(3) Functional capability of element, considering redundancy.

(4) Overall consistency of defined level.

(5) Upward compatibility.

(6) Processing system capability.

To accomplish its work, X3.4.4 was divided into the following four subgroups:

X3.4.4.1 - Compiler Features Study Group
X3.4.4.2 - Audit Routine Group
X3.4.4.3 - COBOL Information Bulletin
X3.4.4.4 - Standard Language Specifications

1.3.2 USA Standard COBOL

On August 30, 1966, X3.4.4 completed its work and approved the content and format for a proposed USA Standard COBOL. The proposed USA Standard COBOL was composed of a nucleus and eight functional processing modules: Table Handling, Sequential Access, Random Access, Random Processing, Sort, Report Writer, Segmentation, and Library. The Nucleus and each of the eight modules were divided into two or three levels. In all cases, the lower levels are proper subsets of the higher levels within the same module. The minimum proposed standard was defined as the low level of the nucleus plus the low level of the table handling and sequential access modules. The highest levels of the nucleus and the eight modules were defined as the full proposed USA Standard COBOL.

The USA Standards Committee on Computer and Information Processing, X3, authorized publication of the proposed USA Standard COBOL to elicit comment and criticism from the data processing community in order that the final standard reflect the largest public consensus. In April 1967, the proposed USA Standard COBOL was published, as COBOL Information Bulletin #9, by the Association for Computing Machinery, Special Interest Committee on Programming Languages (SICPLAN) in the SICPLAN Notices.

X3 also authorized that concurrent with publication of the proposed USA Standard COBOL, a letter ballot be taken of the membership of the X3 committee on the acceptability of the proposed USA Standard COBOL as a USA Standard. The ballots and comments received with the ballots indicated that the X3 members were in favor of the proposed USA Standard COBOL. X3 voted to move

the Random Processing module from the body of the proposed USA Standard COBOL to an appendix and to forward the proposed standard on to the Information Processing Systems Standards Board.

The USA Standard COBOL proposed by X3 was approved by the Information Processing Systems Standards Board on August 23, 1968, as a USA Standard. The specifications of the USA Standard COBOL were published in the USA Standards document X3.23-1968.

The Working Group on Processor Specifications and COBOL, X3.4.4, which developed the Standard, had the following personnel:

H. Bromberg, Chairman

G. F. Archer
G. N. Baird
P. A. Beard
R. F. Betscha
H. W. Fischbeck
H. R. Fletcher
R. C. Fredette
H. S. Gile
N. C. Godfrey
J. S. Grant
W. D. Green
D. C. Harris
M. Hill
K. R. Jensen
A. N. McMahan
J. S. Meach
H. S. Millman
S. N. Naftaly
P. B. Olshansky
R. S. Pettus
E. D. Phillips
L. Rodgers
R. E. Rountree, Jr
J. G. Solomon
R. L. Solt
L. J. Soma
M. Spratt
L. Sturges
M. V. Vickers
L. J. Wilson

1.3.3 International Standardization of COBOL

Throughout the entire COBOL standardization activity of the X3.4.4 Working Group, close liaison was maintained with the various international groups. As a result, American National Standard COBOL complies with the ISO (International Organization for Standardization) Recommendation on COBOL.

The ISO Recommendation R-1989, Programming Language COBOL, was drawn up by the Technical Committee ISO/TC 97, Computers and Information Processing, the Secretariat of which is held by the American National Standards Institute (ANSI). As a result of a six-year development period, the ISO Recommendation reflected the requirements of the international data processing community. The primary objective was to reflect a language rich enough to allow description of a wide variety of data processing problems and to reflect accurately the requirements of the Member Bodies of ISO. Great care was also taken to ensure as far as possible identical interpretation with respect to the national COBOL standards known to be under development.

The Draft ISO Recommendation R-1989 was circulated to all the ISO member bodies for inquiry in July, 1970. The draft was approved, subject to a few modifications of an editorial nature, by all but one of the ISO member bodies. The Draft ISO Recommendation R-1989 was then submitted to the ISO Council, which accepted it as an ISO Recommendation.

2. APPENDIX B: THE REVISION OF AMERICAN NATIONAL STANDARD COBOL

2.1 THE ROLE OF X3J4

Technical Committee X3J4 evolved from Working Group X3.4.4 and its subordinate working groups, the bodies responsible for the development of the first COBOL standard (X3.23-1968). X3J4 was charged with the responsibility for the maintenance of the COBOL standard and in the period immediately following the publication of X3.23-1968, the committee developed and put into effect procedures to handle requests for information, clarification or interpretation of that document.

X3J4 began the task of preparing a revision to the COBOL standard in 1969 with the development of criteria against which each candidate for inclusion in the proposed revision was to be matched. The criteria used were:

(1) The general usefulness of an element or function in terms of:

a. The degree of implementation

b. Acceptance by users

c. The degree to which a function was required

(2) The overall functional capability of the language, considering such things as redundancy.

(3) The state-of-the-art technology with regard to implementing the language feature.

(4) The usefulness, in terms of application requirements, of language capabilities within each level of a module.

(5) Compatibility with other standards.

(6) The cost of implementation versus advantages of use.

(7) Overall language consistency within a defined level or module.

(8) Upward compatibility of levels within a module.

Detailed work on the proposed revision began in early 1970 and, with the committee meeting every 4 to 6 weeks, a draft proposed revision was completed in June 1972. The COBOL community was apprised of the nature of the proposed changes through publication, in the first half of 1972, of COBOL Information Bulletins 14, 15, and 16.

American National Standards Committee X3 agreed, in July 1972, to accept the draft proposal for publication and subsequent letter ballot on the question of its acceptance as a proposed American National Standard. The full text of the proposed revision was made available to the COBOL community for review and comment in September 1972.

2.2 INTERACTION WITH OTHER COBOL GROUPS

2.2.1 Programming Language Committee

The entire technical content of this revision to the COBOL standard was drawn either from the existing COBOL standard (X3.23-1968) or from the CODASYL COBOL Journal of Development (JOD). The Journal of Development is a publication of the CODASYL Programming Language Committee (PLC), the body responsible for the continuing development of the COBOL language. Since the language, and hence the JOD, is constantly changing, it was necessary to select the JOD of a given date to serve as the base document for the revision process. This date, known as the cutoff date, was December 31, 1971. Changes to the language after that date were considered for inclusion in the revision only where they were in response to X3J4 proposals or where they affected items whose final disposition had been deferred by X3J4 pending specific PLC action.

Throughout the revision cycle, PLC gave priority in its agenda to X3J4 proposals and requests for language clarification. Their generous cooperation during this period made the task of X3J4 considerably lighter and contributed significantly to the quality of the revised standard.

2.2.2 International Standardization Bodies

Close and continuous liaison was maintained with the international COBOL community during the work on the revision. This culminated in February 1972 with a meeting of representatives of X3J4, European Computer Manufacturers Association Technical Committee 6 (ECMA TC6), and several ISO (International Organization for Standardization) member organizations to review the proposed changes and to resolve any differences of opinion that existed concerning the technical content of the proposed revision.

ECMA TC6 played a very active part throughout the revision process and made a number of significant contributions to the enhancement and clarification of the revision.

2.3 DIFFERENCES BETWEEN X3.23-1968 AND THE REVISED STANDARD

2.3.1 Format of the Revised Standard

As was the case with X3.23-1968, the organization of the specifications in the revised standard is based on a functional processing module concept with each module divided into two or more levels. Unlike X3.23-1968, however, where a separate chapter was devoted to each processing level, each module in the revised document is covered in a single chapter. The high level features are boxed and any restrictions in the low levels are covered by additional rules.

The revision defines a Nucleus and eleven functional processing modules: Table Handling, Sequential I-O, Relative I-O, Indexed I-O, Sort-Merge, Report Writer, Segmentation, Library, Debug, Inter-Program Communication, and Communication. Nine modules contain a null set as their lowest level and in all cases, the lower levels are proper subsets of the higher levels within the same module.

2.3.2 Overview of the Revised Modules

As in X3.23-1968, the Nucleus is divided into two levels. The major changes introduced into the Nucleus are:

(1) The REMARKS paragraph and the NOTE statement have been deleted in favor of a generalized comment facility. An * in character position 7 now identifies any line as a comment line. A further refinement of this (a slash '/' in character position 7) causes the line to be treated as a comment and causes page ejection.

(2) The EXAMINE statement has been deleted in favor of the more general and powerful INSPECT statement. The INSPECT statement provides the ability to count (Format 1), replace (Format 2) or count and replace (Format 3) occurrences of single characters or groups of characters in a data item.

(3) Level 77 items need no longer precede level 01 items in the Working-Storage Section.

(4) The punctuation rules with regard to spaces have been relaxed, e.g., spaces may now optionally precede the comma, period or semicolon, and may optionally precede or follow a left parenthesis.

(5) Two contiguous quotation marks may be used within a nonnumeric literal to represent a single occurrence of the character quotation mark.

(6) A SIGN clause has been added that permits the specification of the position that the sign is to occupy in a signed numeric item (either leading or trailing) and/or that it is to occupy a separate character position. Other changes in the Data Division permit the object of a REDEFINES clause to be subordinate to a data item described with an OCCURS clause, set the maximum size of a numeric field at 18 digits, permit the stroke '/' as an editing character and specify some tightening of the rules concerning literals in the VALUE clause (if the literal is signed, the data item must be described as signed; if the data item is numeric edited, the literal must be nonnumeric).

(7) The ACCEPT statement has been expanded to provide access to internal DATE, DAY and TIME.

(8) GIVING identifier series has been added to the arithmetic statements; identifier series has been added to the COMPUTE statement; and INTO identifier series has been added to the DIVIDE statement.

(9) The STRING statement has been added. This statement provides for the juxtapositioning within a single data item of the partial or complete contents of two or more data items. A companion statement, the UNSTRING, has also been added. This statement causes contiguous data within a single data item to be separated and placed in multiple receiving fields.

(10) Certain ambiguities in abbreviated combined conditions with regard to NOT and the use of parentheses have been eliminated. Where any portion of an abbreviated combined condition is enclosed in parentheses, all subjects and operators required for the expansion of that portion must be included within the same set of parentheses.

(11) The PROGRAM COLLATING SEQUENCE clause was added, to permit specification of the collating sequence used in nonnumeric comparisons. Native, ASCII, implementor-defined and user-defined collating sequences may be specified. This makes possible the processing of ASCII files without changing source program logic.

The Table Handling module is divided into two levels; Level 1 contains essentially all that appears in Levels 1 and 2 of X3.23-1968, and Level 3 of X3.23-1968 becomes Level 2 in the revision. Among the more important changes introduced into this module are:

(1) The left parenthesis enclosing subscripts need not be preceded by a space. Commas are not required between subscripts or indices. Literals and index-names may be mixed in a table reference.

(2) A data description entry that contains an OCCURS DEPENDING ON clause may only be followed, within that record description, by data description entries that are subordinate to it. Thus, the "fixed" portion of a record must entirely precede any "variable" portion. The effect of the OCCURS DEPENDING ON was clarified to state explicitly that internal operations involving tables described with this clause reference only the portion of the table that is "active" (i.e., the actual size as defined by the current value of the operand of the DEPENDING ON phrase is used).

(3) An index may be set up or down by a negative value.

(4) The subject of the condition in the WHEN phrase of the SEARCH ALL statement must be a data item named in the KEY phrase of the referenced table; the object of this condition may not be such a data item. X3.23-1968 specified that either the subject or the object could be a data item named in the KEY phrase.

As in X3.23-1968, the Sequential I-O module is divided into two levels. Among the significant changes introduced into this module are:

(1) The FILE-LIMITS clause, the MULTIPLE REEL/UNIT clause, and the integer implementor-name phrase of the file control entry were deleted because it was felt that these functions could be handled better outside of the COBOL program.

(2) The SEEK statement was deleted because it was felt to be redundant (it is implied by the READ, WRITE, etc.) and ineffective.

(3) OPEN REVERSED now positions a file at its end. OPEN EXTEND was added to permit the addition of records at the end of an existing sequential file.

(4) USE AFTER STANDARD ERROR was changed to read USE AFTER STANDARD ERROR/EXCEPTION; the function was expanded to permit invocation of the associated procedure on both error (e.g., boundary violation) or exception (e.g., AT END) conditions.

(5) The AT END phrase of the READ statement was made optional; it must appear, however, if no applicable USE procedure appears.

(6) The INVALID KEY phrase of the WRITE was deleted since there is no user-defined key for sequential files. Error and/or exception conditions can be monitored through appropriate USE statements.

(7) The FILE STATUS clause was added to permit the system to convey information to the program concerning the status of I/O operations. Codes for "error", AT END, etc., have been defined.

(8) The REWRITE statement was added to permit the explicit updating of records on a sequential file.

(9) The LINAGE clause was added to permit programmer definition of logical page size and of the size of top and bottom margins on the logical page.

(10) The PAGE phrase was added to the WRITE statement to permit presentation of a line before or after advance to the top of the next logical page.

(11) The facility of define, initialize and access user-defined labels has been deleted.

(12) The CODE-SET clause has been added to provide for conversion of sequential non-mass storage files encoded in ASCII or implementor-specified codes from/to the native character code.

The Random Access module of X3.23-1968 has been replaced by two new modules, the Relative I-O and Indexed I-O modules. Both of the modules are composed of three levels, the first of which is null. While there is much functional and even syntactic similarity between the Relative I-O module and the existing Random Access module, the Indexed I-O module has no functional equivalent in the previous standard.

Among the major features of the Relative I-O module are:

(1) An ORGANIZATION IS RELATIVE clause.

(2) A RELATIVE KEY clause.

(3) An ACCESS MODE clause which specifies random, sequential or dynamic access. Dynamic access permits the file to be accessed both randomly and sequentially.

(4) FILE STATUS and USE AFTER STANDARD ERROR/EXCEPTION as outlined in the Sequential I-O module. Here also the USE procedure may be used in place of the AT END and INVALID KEY phrases of the READ, WRITE, etc.

(5) In addition to OPEN, CLOSE, READ and WRITE, the DELETE, REWRITE, and START verbs are provided. READ NEXT provides for the intermixing of sequential with random accesses of the file (when access mode is dynamic). START provides the facility to position the file such that the next sequential READ statement will reference a specified record.

Among the major features of the Indexed I-O module are:

(1) An ORGANIZATION IS INDEXED clause.

(2) An ACCESS MODE clause with characteristics similar to that of the Relative I-O module.

(3) FILE STATUS and USE procedures, as in the Relative I-O module.

(4) The RECORD KEY clause specifies the data item that serves as the unique identifier for each record. The data item is known as the prime record key. The ALTERNATE KEY clause specifies additional (alternate) keys for the file. All insertion, updating or deletion of records is done on the basis of the prime record key. Retrieval, however, may be on the basis of either the prime or alternate record keys, thus providing more than one access path through the file.

(5) As in the Relative I-O module, the new verbs DELETE, START and REWRITE are available. READ NEXT and READ...KEY IS... are also available; the latter provides the means of specifying the key upon which retrieval is to be based (prime or alternate). The START statement also provides the means of specifying whether the prime or alternate key is to be used for positioning the file.

The Sort-Merge module contains three levels, one of which is null. The major change to the Sort module of the previous standard has been the addition of a MERGE statement to permit the combination of two or more identically ordered files. The MERGE statement parallels the SORT statement in format, except that no input procedure is provided. The COLLATING SEQUENCE phrase has been added to permit overriding of the program collating sequence when executing a SORT or MERGE statement.

The Report Writer module has two levels, one of which is null (X3.23-1968 has two non-null levels). The Report Writer module was completely rewritten in order to remove existing ambiguities and to provide a stronger and more useful facility. Care was taken in the rewrite not to imply that reports had to be presented on a printer (rather than on a type of graphic device).

The Segmentation module has three levels, the first of which is null. The major changes introduced are:

(1) There is no logical difference between fixed and fixed overlayable segments (X3.23-1968 placed certain restrictions on the range of PERFORM's involving fixed overlayable segments).

(2) A PERFORM statement in a non-independent segment may have only one of the following within its range: (1) non-independent segments or (2) sections wholly contained in a single independent segment. A similar constraint applies to a PERFORM in an independent segment, except that (2) reads "Sections wholly contained in the same independent segment." Where a SORT or MERGE statement appears in a segmented program, then any associated input/output procedures are subject to the same constraints that apply to the range of a PERFORM (e.g., where the SORT is in a non-independent segment, the associated input/output procedures must be either wholly contained in non-independent segments or wholly contained in a single independent segment).

The Library module has a null level and two non-null levels. The major changes introduced are:

(1) The COPY statement may appear anywhere in the program that a COBOL word or separator may appear (X3.23-1968 permitted the COPY statement to appear only in certain specified places).

(2) More than one library can be available.

(3) All occurrences of a given literal, identifier, word or group of words in the library text can be replaced. (X3.23-1968 did not permit replacement of groups of words.)

(4) The matching and replacement process has been significantly clarified.

The new Debug module provides a means by which the programmer can specify a debugging algorithm, including the conditions under which data items or procedures are to be monitored during program execution. This module has a null level and two non-null levels. The major features of this module are:

(1) A USE FOR DEBUGGING statement which permits full or selective procedure and data-name monitoring; control is passed to the procedure when the specified condition arises. Associated with the execution of each debugging section (i.e., the declarative procedure associated with the USE FOR DEBUGGING statement) is the special register DEBUG-ITEM. This is updated by the system each time a debugging section is executed with such information as the name (with occurrence numbers if it should be the name of a table element), that caused the execution, the line number upon which the name appears, etc. The USE FOR DEBUGGING statements and their associated declarative procedures are treated as comment lines if the WITH DEBUGGING MODE clause does not appear in the program. An object time switch is also provided, outside of the COBOL program, through which the USE FOR DEBUGGING procedures can be "turned off" without the need to recompile the program.

(2) Debugging lines. Any line with a "D" in the continuation area is a debugging line and will be compiled and executed only if the WITH DEBUGGING MODE clause appears in the program. Where this compile time switch does not appear in the program, these lines are treated as comment lines. The setting of the object time switch has no effect on the execution of debugging lines. Through the debugging line facility, the programmer has at his disposal the full power of the COBOL language for debugging purposes.

The new Inter-Program Communication module provides a facility by which a program can communicate with one or more other programs. This communication is provided by: (a) the ability to transfer control from one program to another within a run unit and (b) the ability for both programs to have access to the same data items. This module has three levels, the first of which is null. The major features of this module are:

(1) The CALL statement causes control to be transferred from one object program to another. The CALL statement can be "static" (i.e., the name of the called program is known at compile time) or dynamic (i.e., the name of the called program is not known until program execution time). The USING phrase of the CALL statement names the data to be shared with the called program; a USING phrase in the Procedure Division header of the called program specifies the names by which this shared data is to be known in the called program. The ON OVERFLOW phrase of the CALL statement will cause control to

be transferred to an associated imperative statement if there is not enough memory available at execution time to permit the loading of the called program.

(2) The CANCEL statement releases the areas occupied by called programs that are no longer required to be in memory.

(3) The EXIT PROGRAM statement marks the logical end of a called program and causes control to be returned to the calling program (i.e., the program in which the CALL statement appears).

(4) The Linkage Section appears in a program that is to operate under the control of a CALL statement. It is used in the called program to describe data that is to be made available from the calling program through the CALL USING facility described above.

The new Communication module provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System with local and remote communication devices. This new module has three levels, the first of which is null. Among the major features of the module are:

(1) The communication description entry (CD) specifies the interface area between the Message Control System (MCS) and a COBOL program. The CD specifies the input message queue structure, the symbolic names of destinations for output messages and such things as message date, message time and text length.

(2) The ENABLE and DISABLE statements notify the MCS to permit or inhibit the transfer of data between specified output queues and destinations for output or between sources and input queues for input.

(3) The RECEIVE statement makes available, from a queue maintained by the Message Control System, to the COBOL program a message, or portion thereof, and pertinent information about the message.

(4) The SEND statement causes a message or a portion of a message to be released to one or more output queues maintained by the MCS.

(5) The ACCEPT MESSAGE COUNT statement causes the number of messages in a queue to be made available.

(6) The FOR INITIAL INPUT clause of the CD entry permits the MCS to schedule a program for execution upon receipt of a message for that program.

In addition to the technical changes outlined above, a number of changes were made in the definition of an implementation of American National Standard COBOL. (See page I-4.)

2.3.3 Substantive Changes

The list beginning on page XIV-17 contains the changes of substance that have been included in the revised standard. The code reflected under the remarks column is as follows:

(1) Indicates the change will not impact existing programs. For example, a new verb or an additional capability for an old verb.

(2) Indicates the change could impact existing programs and some re-programming may be needed. For example, where the semantics or syntax of an existing verb were changed.

(3) Indicates that the change impacts an area that was implementor-defined in the original standard. As such it may or may not affect existing programs.

Additions to the reserved word list that will impact existing programs are not included in the list.

Language elements associated with the Report Writer module are not assigned codes because the report writer specifications were completely rewritten and comparison with the previous standard is therefore not meaningful.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
1.	Space may immediately precede or may immediately follow a parenthesis (except in a PICTURE character-string).	1 NUC	(1) Relaxes punctuation rules.
2.	Period, comma, or semicolon may be preceded by a space.	1 NUC 1 TBL	(1) Relaxes punctuation rules.
3.	Semicolon and comma are interchangeable.	1 NUC	(1)
4.	An asterisk (*) in the continuation area (seventh character position) causes the line to be treated as a comment by the compiler. The comment line may appear in any division.	1 NUC	(1) New feature; replaces the NOTE statement and REMARKS paragraph.
5.	A stroke (slash, '/', virgule) in the continuation area (seventh character position) of a line causes page ejection of the compilation listing. (The line is treated as a comment.)	1 NUC	(1)
6.	A phrase or clause (as well as a sentence or entry) may be continued by starting subsequent lines in area B.	1 NUC	(1)
7.	Two contiguous quotation marks may be used to represent a single quotation mark character in a nonnumeric literal.	1 NUC	(1) New feature.
8.	Last line in program may be a comment line.	1 NUC	(1)
9.	Mnemonic-name must have at least one alphabetic character.	1 NUC	(3) X3.23-1968 had no such restriction.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
10.	Number of qualifiers permitted is implementor-defined, but must be at least five.	2 NUC	(2) X3.23-1968 specified no such lower limit.
11.	Complete set of qualifiers for a name may not be same as partial list of qualifiers for another name.	2 NUC	(2)
12.	REMARKS paragraph is deleted.	1 NUC	(2) Function was replaced by the comment line.
13.	Continuation of Identification Division comment-entries must not have a hyphen in the continuation indicator area.	1 NUC	(2)
14.	PROGRAM COLLATING SEQUENCE clause specifies that the collating sequence associated with alphabet-name is used in nonnumeric comparisons.	1 NUC	(1) New feature.
15.	SPECIAL-NAMES paragraph: 'L', '/', and '=' may not be specified in the CURRENCY SIGN clause.	2 NUC	(2) This restriction did not exist in X3.23-1968.
16.	Alphabet-name clause relates a user-defined name to a specified collating sequence or character code set (ANSI, native, or implementor-specified).	1 NUC	(1) New feature.
17.	Alphabet-name clause: the literal phrase specifies a user-defined collating sequence.	2 NUC	(1) New feature.
18.	Condition-name may be given the status of an implementor-defined switch. Switches are implementor-defined and may be either software or hardware switches.	1 NUC	(1) X3.23-1968 specified hardware switches only.
19.	All items which are immediately subordinate to a group item must have the same level-number.	1 NUC	(2)
20.	Level 77 items need not precede level 01 items in the Working-Storage Section.	1 NUC	(1) New feature.
21.	Level numbers 02-49 may appear anywhere to the right of margin A. (Margin A is defined as being between character positions 7 and 8.)	1 NUC	(1)

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
22.	Object of a REDEFINES clause can be subordinate to an item described with an OCCURS clause, but must not be referred to in the REDEFINES clause with a subscript or an index.	1 NUC	(1) New feature.
23.	REDEFINES: No entry with lower level-number can appear between the redefined and redefining items.	1 NUC	(2) X3.23-1968 had no such restriction.
24.	Multiple redefinition of same storage area permitted.	1 NUC	(3)
25.	As asterisk used as a zero suppression symbol in a PICTURE clause and the BLANK WHEN ZERO clause may not appear in the same entry.	1 NUC	(2)
26.	Alphabetic PICTURE character-string may contain the character B.	1 NUC	(1) New feature.
27.	The number of digit positions that can be described by a numeric PICTURE character-string cannot exceed 18.	1 NUC	(2) X3.23-1968 had no such rule.
28.	Stroke (/) permitted as an editing character.	1 NUC	(1) New feature.
29.	PICTURE character-string is limited to 30 characters.	1 NUC	(3) X3.23-1968 defines limit as 30 symbols where one symbol could have been two characters.
30.	SIGN clause allows the specification of the sign position.	1 NUC	(1) New feature.
31.	A signed numeric literal cannot be used in a VALUE clause unless it is associated with a signed PICTURE character-string.	1 NUC	(2)
32.	If the item is numeric edited, the literal in the VALUE clause must be nonnumeric.	1 NUC	(2)
33.	In the Procedure Division a section may contain zero or more paragraphs and a paragraph may contain zero or more sentences.	1 NUC	(1) New feature.
34.	The unary + is permitted in arithmetic expressions.	2 NUC	(1) New feature.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
35.	The TO is not required in the EQUAL TO of a relation condition.	1 NUC	(1) X3.23-1968 required the word TO.
36.	In relation and sign conditions, arithmetic expressions must contain at least one reference to a variable.	1 NUC	(2)
37.	Comparison of nonnumeric operands; If one of the operands is described as numeric, it is treated as though it were moved to an alphanumeric item of the same size and the contents of this alphanumeric item were then compared to the nonnumeric operand.	1 NUC	(3)
38.	Abbreviated combined relation condition: When any portion is enclosed in parentheses, all subjects and operators required for the expansion of that portion must be included within the same set of parentheses.	2 NUC	(2) No such restriction appeared in X3.23-1968.
39.	Abbreviated combined relation condition: If NOT is immediately followed by a relational operator, it is interpreted as part of the relational operator.	2 NUC	(2) In X3.23-1968, NOT was a logical operator in such cases.
40.	Class condition: The numeric test cannot be used with a group item composed of elementary items described as signed.	1 NUC	(3)
41.	In an arithmetic operation, the composite of operands must not contain more than 18 decimal digits.	1 NUC	(2) X3.23-1968 specified limits only for ADD and SUBTRACT.
42.	ACCEPT identifier FROM DATE/DAY/TIME allows the programmer to access the date, day, and time.	2 NUC	(1) New feature.
43.	ADD statement: the GIVING identifier series.	2 NUC	(1) New feature.
44.	COMPUTE statement: the identifier series.	2 NUC	(1) New feature.
45.	DISPLAY statement: If the operand is a numeric literal, it must be an unsigned integer.	1 NUC	(2)

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
46.	DIVIDE statement: the INTO identifier series and the GIVING identifier series.	1 NUC	(2)
47.	DIVIDE statement: the remainder item can be numeric edited.	2 NUC	(1) New feature.
48.	GO TO statement: the word TO is not required.	1 NUC	(1) X3.23-1968 requires the word TO.
49.	EXAMINE statement and the special register TALLY were deleted.	1 NUC	(2) Function was replaced by the INSPECT statement.
50.	INSPECT statement provides ability to count or replace occurrences of single characters or groups of characters.	1 NUC	(1) New feature.
51.	MOVE statement: A scaled integer item (i.e., the rightmost character of the PICTURE character-string is a P) may be moved to an alphanumeric or alphanumeric edited item.	1 NUC	(1) New feature.
52.	MULTIPLY statement: the BY identifier series and the GIVING identifier series.	2 NUC	(1) New feature.
53.	PERFORM statement: Format 4 (PERFORM ...VARYING, not using index-names) identifiers need not be described as integers.	2 NUC	(1) New feature.
54.	PERFORM statement: Changing the FROM variable during execution can affect the number of times the procedures are executed in a Format 4 PERFORM if more than one AFTER phrase is specified.	2 NUC	(2)
55.	PERFORM statement: There is no logical difference to the user between fixed and fixed overlayable segments.	1 NUC	(1) X3.23-1968 did not permit fixed overlayable segments to be treated the same as a fixed segment.
56.	A PERFORM statement in a non-independent segment can have in its range only one of the following: a. Non-independent segment (fixed/fixed overlayable) b. Sections and/or paragraphs wholly contained in a single independent segment.	1 NUC 1 SEG	(3)

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
57.	A PERFORM statement in an independent segment can have in its range only one of the following: a. Non-independent segments (fixed/ fixed overlayable). b. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM.	1 NUC 1 SEG	(3)
58.	PERFORM statement: Control is passed only once for each execution of a Format 2 PERFORM statement. (i.e., an independent segment referred to by such a PERFORM is made available in its initial state only once for each execution of that PERFORM statement.)	1 NUC 1 SEG	(3)
59.	STOP statement: If the operand is numeric literal, it must be an unsigned integer.	1 NUC	(2)
60.	STRING statement provides for the juxtaposition of the partial or complete contents of two or more data items into a single data item.	2 NUC	(1) New feature.
61.	STRING: Delimiter identifiers need not be fixed length items.	2 NUC	(1)
62.	SUBTRACT statement: the GIVING identifier series.	2 NUC	(1) New feature.
63.	UNSTRING statement permits contiguous data in sending field to be separated and placed into multiple receiving fields.	2 NUC	(1) New feature.
64.	Commas are not required between subscripts or index-names.	1 TBL	(1)
65.	Literal subscripts may be mixed with index-names when referencing a table item.	1 TBL	(1) New feature.
66.	The DEPENDING phrase is now required in the Format 2 of the OCCURS clause.	2 TBL	(2) X3.23-1968 has no restriction.
67.	Integer-1 cannot be zero in the Format 2 of the OCCURS clause.	2 TBL	(2)

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
68.	A data description entry with an OCCURS DEPENDING clause may be followed within that record, only by entries subordinate to it. (i.e., only the last part of the record may have a variable number of occurrences.)	2 TBL	(2) This rule did not appear in X3.23-1968.
69.	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 defined by the value of the operand of the DEPENDING phrase will be used in the operation. (i.e., the actual size of a variable length item is used, not the maximum size.)	2 TBL	(2)
70.	If SYNCHRONIZED is specified for an item containing an OCCURS clause, any implicit FILLER generated for items in the same table are generated for each occurrence of those items.	1 TBL	(3)
71.	The results of a SEARCH ALL operation are predictable only when the data in the table is ordered as described by the ASCENDING/DESCENDING KEY clause associated with identifier-1.	2 TBL	(3)
72.	The subject of the condition in the WHEN phrase of the SEARCH ALL statement must be a data item named in the KEY phrase of the table; the object of this condition may not be a data item named in the KEY phrase.	2 TBL	(2) X3.23-1968 specified that either the subject or object could be a data item named in the KEY phrase.
73.	SEARCH...VARYING identifier-2: If identifier-2 is an index data item, it is incremented as the associated index is incremented.	2 TBL	(3) In X3.23-1968 the data item is incremented by same amount as occurrence number, i.e., by one.
74.	In Format 2 of the SET statement, literal may be negative.	1 TBL	(1) New feature.
75.	File control entry: The ASSIGN TO implementor-name-1 OR implementor-name-n clause for the GIVING file of a SORT statement was deleted.	1 SRT	(2)
76.	MERGE statement	2 SRT	(1) New feature.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
77.	RELEASE...FROM identifier is placed in Level 1 of Sort-Merge module.	1 SRT	(1) Was a Level 2 feature.
78.	RETURN...INTO identifier is placed in Level 1 of Sort-Merge module.	1 SRT	(1) Was a Level 2 feature.
79.	SORT statement: the USING file-name series.	2 SRT	(1) X3.23-1968 allowed only one file-name.
80.	SORT statement: semicolon deleted from format.	1 SRT	(2)
81.	SORT statement: COLLATING SEQUENCE phrase provides the ability to override the program collating sequence.	2 SRT	(1) New feature
82.	No more than one file-name from a multiple file reel can appear in a SORT statement.	2 SRT	(2)
83.	Where a SORT or MERGE statement appears in a segmented program, then any associated input/output procedures are subject to the same constraints that apply to the range of a PERFORM.	1 SRT 1 SEG	(2) No such restriction in X3.23-1968.
84.	Segment-numbers permitted in declaratives.	1 SEG	(1)
85.	PAGE-COUNTER and LINE-COUNTER are described as unsigned integers that must handle values from 0 through 999999.	RPW	
86.	The value in LINE-COUNTER must not be changed by the user.	RPW	
87.	LINE-COUNTER, PAGE-COUNTER and sum counters must not be used as subscripts in the Report Section.	RPW	
88.	PAGE-COUNTER is always generated.	RPW	
89.	PAGE-COUNTER does not need to be qualified in the Report Section.	RPW	
90.	LINE-COUNTER is always generated.	RPW	
91.	LINE-COUNTER does not need to be qualified in the Report Section.	RPW	

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
92.	The words LINE and LINES are optional in the PAGE clause.	RPW	
93.	The DATA RECORDS clause and the REPORT clause are mutually exclusive.	RPW	
94.	A report may not be sent to more than one file.	RPW	
95.	RESET is no longer a clause; it is a phrase under the SUM clause.	RPW	
96.	Multiple SUM clauses may be specified in an item; multiple UPON phrases may be specified.	RPW	
97.	Up to three hierarchical levels are permitted in a report group description.	RPW	
98.	A report group level 01 entry cannot be elementary.	RPW	
99.	An entry that contains a LINE NUMBER clause must not have a subordinate entry that also contains a LINE NUMBER clause.	RPW	
100.	An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.	RPW	
101.	An entry that contains a VALUE clause must also have a COLUMN NUMBER clause.	RPW	
102.	In the CODE clause, mnemonic-name has been replaced by literal. (A two character nonnumeric literal placed in the first two character positions of the logical record.)	RPW	
103.	If the CODE clause is specified for any report in a file, it must be specified for all reports in the same file.	RPW	
104.	Control data items may not be subscripted or indexed.	RPW	
105.	Each data-name in the CONTROL clause must identify a different data item.	RPW	

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
106.	The GROUP INDICATE clause may only appear in a DETAIL report group entry that defines a printable item (contains a COLUMN and PICTURE clause.)	RPW	
107.	LINE clause integers must not exceed three significant digits in length.	RPW	
108.	The NEXT PAGE phrase of the LINE clause is no longer legal in RH, PH, and PF groups.	RPW	
109.	A relative LINE NUMBER clause can no longer be the first LINE NUMBER clause in a PAGE FOOTING group.	RPW	
110.	A NEXT GROUP clause without a LINE clause is no longer legal.	RPW	
111.	Integer-2 in the NEXT GROUP clause must not exceed three significant digits in length.	RPW	
112.	If the PAGE clause is omitted, only a relative NEXT GROUP clause may be specified.	RPW	
113.	The NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a PAGE FOOTING report group.	RPW	
114.	The NEXT GROUP clause must not be specified in a REPORT FOOTING report group.	RPW	
115.	The phrases of the PAGE clause may be written in any order.	RPW	
116.	In the PAGE clause, the maximum size of the integer is three significant digits.	RPW	
117.	It is no longer possible to sum upon an item in another report.	RPW	
118.	Source-sum correlation is not required. (Operands of a SUM clause need not be operands of a SOURCE clause in DETAIL groups.)	RPW	
119.	TYPE clause data-names may not be subscripted or indexed.	RPW	

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
120.	PAGE HEADING and PAGE FOOTING report groups may be specified only if a PAGE clause is specified in the corresponding report description entry.	RPW	
121.	In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE clauses and USE statements may not reference: a. Group data items containing control data items. b. Data items subordinate to a control data item. c. A redefinition or renaming of any part of a control data item. In PAGE HEADING and PAGE FOOTING report groups, SOURCE clauses and USE statements must not reference control data-name.	RPW	
122.	In summary reporting, only one detail group is allowed.	RPW	
123.	The description of a report must include at least one body group.	RPW	
124.	Report files must be opened with either the OPEN OUTPUT or OPEN EXTEND statement.	RPW	
125.	A file described with a REPORT clause cannot be referenced by any input-output statement except the OPEN or CLOSE statement.	RPW	
126.	The SUPPRESS statement	RPW	
127.	If no GENERATE statements have been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for that report, the TERMINATE statement does not cause the Report Writer Control System to perform any of the related processing.	RPW	
128.	A USE procedure may refer to a DETAIL group.	RPW	

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
129.	FILE STATUS clause: data-name is updated by the system at the completion of each input-output operation.	1 SEQ 1 REL 1 INX	(1) New feature.
130.	ACCESS MODE IS DYNAMIC clause: provides ability to access a file sequentially or randomly in the same program.	2 REL 2 INX	(1) New feature.
131.	ALTERNATE RECORD KEY clause: allows specification of multiple keys, any of which can be used to access an indexed file	2 INX	(1) New feature.
132.	ACTUAL KEY clause deleted.		(2)
133.	RELATIVE KEY clause added for relative organization.	1 REL	(1) New feature.
134.	RECORD KEY clause added for indexed organization.	1 INX	(1) New feature.
135.	FILE-LIMITS clause deleted.		(2)
136.	PROCESSING MODE clause deleted.		(2)
137.	FILE-CONTROL paragraph: except for the ASSIGN clause, the order of clauses following file-name is optional.	1 SEQ 1 REL 1 INX	(1)
138.	ORGANIZATION IS RELATIVE clause	1 REL	(2) New feature.
139.	ORGANIZATION IS SEQUENTIAL clause	1 SEQ	(2) New feature.
140.	ORGANIZATION IS INDEXED clause	1 INX	(2) New feature.
141.	MULTIPLE REEL/UNIT clause deleted.		(2)
142.	RESERVE...ALTERNATE AREAS deleted.		(2)
143.	RESERVE integer AREAS to allow the user to specify the exact number of areas to be used.	1 SEQ 1 REL 1 INX	(1) New feature.
144.	The file description entry for file-name must be equivalent to that used when this file was created.	1 SEQ 1 REL 1 INX	(3) No such rule in X3.23-1968.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
145.	The data-name option of the LABEL RECORDS clause was deleted.	1 SEQ 1 REL 1 INX	(2) X3.23-1968 provided for user-defined label records.
146.	Data-name in the VALUE OF clause must be an implementor-name.	1 SEQ	(2) X3.23-1968 provided for user-defined field in label records.
147.	LINAGE clause permits programmer definition of logical page size.	2 SEQ	(1) New feature.
148.	CLOSE...FOR REMOVAL statement.	2 SEQ	(1) New feature.
149.	DELETE statement.	1 REL 1 INX	(1) New feature.
150.	OPEN REVERSED positions file at its end.	2 SEQ	(2)
151.	OPEN INPUT or OPEN I-O makes a record area available to the program.	1 SEQ 1 REL 1 INX	(1) New feature.
152.	OPEN EXTEND statement: adds records to an existing file.	2 SEQ	(1) New feature.
153.	The OPEN and CLOSE statements with the NO REWIND phrase apply to all devices that claim support for this function.	2 SEQ	(1) X3.23-1968 restricted the application of this phrase.
154.	The OPEN REVERSED statement applies to all devices that claim support for this function.	2 SEQ	(1) X3.23-1968 restricted the application of this phrase.
155.	READ statement: AT END phrase required only if no applicable USE AFTER ERROR/ EXCEPTION procedure specified.	1 SEQ 1 REL 1 INX	(1) New feature.
156.	READ statement: INVALID KEY phrase required only if no applicable USE AFTER ERROR/EXCEPTION procedure specified.	1 REL 1 INX	(1) New feature.
157.	READ statement: INTO phrase placed in Level 1.	1 SEQ 1 REL 1 INX	(1) Level 2 feature in X3.23-1968.
158.	READ...NEXT statement: used to retrieve the next logical record from a file when the access mode is dynamic.	2 REL	(1) New feature.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
159.	REWRITE statement	1 SEQ 1 REL 1 INX	(1) New feature.
160.	SEEK statement was deleted.		(2)
161.	START statement: provides for logical positioning within a relative or indexed file for sequential retrieval of records.	2 REL	(1) New feature.
162.	USE statement: the label processing options were deleted.	1 SEQ 1 REL 1 INX	(2) X3.23-1968 provided for the processing of user-defined labels.
163.	USE...ERROR/EXCEPTION statement	1 SEQ 1 REL 1 INX	(1) New feature.
164.	Recursive invocation of USE procedures prohibited.	1 SEQ 1 REL 1 INX	(2)
165.	WRITE statement: INVALID KEY phrase deleted.	1 SEQ	(2)
166.	WRITE statement: INVALID KEY phrase required only if no applicable USE AFTER ERROR/EXCEPTION procedure specified.	1 REL 1 INX	(1)
167.	WRITE statement: FROM phrase placed in Level 1.	1 SEQ 1 REL 1 INX	(1) Level 1 feature in X3.23-1968.
168.	WRITE statement: BEFORE/AFTER PAGE phrase provides ability to skip to top of a page.	1 SEQ	(1)
169.	WRITE statement: END-OF-PAGE phrase	2 SEQ	(1) New feature.
170.	Debugging line: defined by a 'D' in the continuation column.	1 DEB	(1) New feature.
171.	WITH DEBUGGING MODE clause: a compile time switch; in addition an object time switch can be used to suppress coding at object time.	1 DEB	(1) New feature.
172.	USE FOR DEBUGGING statement.	1 DEB	(1) New feature.
173.	DEBUG-ITEM	1 DEB	(1) New feature.

	SUBSTANTIVE CHANGE	MODULE AFFECTED	REMARKS
174.	Linkage Section	1 IPC	(1) New feature.
175.	Procedure Division header: the USING phrase.	1 IPC	(1) New feature.
176.	CALL identifier statement.	1 IPC	(1) New feature.
177.	CALL identifier ON OVERFLOW statement.	2 IPC	(1) New feature.
178.	CANCEL statement	2 IPC	(1) New feature.
179.	EXIT PROGRAM statement	1 IPC	(1) New feature.
180.	COPY statement may appear anywhere a COBOL word may appear.	1 LIB	(1) New feature.
181.	Identifier, COBOL word, or a group of COBOL words may be replaced.	2 LIB	(1) New feature.
182.	Multiple libraries are permitted.	2 LIB	(1) New feature.
183.	Library-name is a user-defined word.	2 LIB	(1) New feature.
184.	Communication description entry (CD)	1 COM	(1) New feature.
185.	ACCEPT cd-name MESSAGE COUNT statement.	1 COM	(1) New feature.
186.	ENABLE statement	1 COM	(1) New feature.
187.	DISABLE statement	1 COM	(1) New feature.
188.	RECEIVE statement	1 COM	(1) New feature.
189.	SEND statement	1 COM	(1) New feature.

2.3.4 Elements Deleted From X3.23-1968

The following elements were deleted from X3.23-1968 in the process of revising the standard. Page numbers refer to pages in the document X3.23-1968.

REMARKS Paragraph (page 2-4). The REMARKS paragraph of the Identification Division was deleted and the function replaced by the * comment line.

EXAMINE Statement (pages 2-33 and 2-85). The EXAMINE statement and the special register TALLY were deleted in favor of the new more powerful INSPECT statement.

NOTE Statement (pages 2-40 and 2-92). The NOTE statement was deleted and the function replaced by the * comment line.

FILE-LIMITS Clause (pages 2-119 and 2-155). This clause was deleted from the file control entry because the function could be handled better outside the COBOL program.

SEEK Statement (page 2-164). This statement was redundant; it is implied by the READ, WRITE, etc.

MULTIPLE REEL/UNIT Clause (page 2-119). This clause was deleted from the file control entry because the function could be handled better outside the COBOL program.

ACTUAL KEY Clause (page 2-156). This clause was replaced by the RELATIVE KEY clause.

RESERVE integer ALTERNATE AREAS Clause (page 2-134). This clause was replaced by the RESERVE integer AREAS clause.

OR implementor-name (page 2-138). This clause was deleted from the file control entry because the function could be handled better outside the COBOL program.

integer implementor-name (pages 2-119 and 2-155). This clause was deleted from the file control entry because the function could be handled better outside the COBOL program.

PROCESSING MODE IS SEQUENTIAL Clause (pages 2-119 and 2-155). This clause was deleted from the file control entry as not being needed in a synchronous environment.

USE...LABEL Statement (pages 2-150 and 2-180). An extensive revision to label processing is currently underway to remove ambiguities and provide for the processing of ANSI standard labels. This work was not completed in time to be included in this revision. In order not to hinder the introduction of this new facility, it was decided to define only a minimum label processing capability in the revised standard.

LABEL RECORDS IS data-name Clause (pages 2-141 and 2-174). An extensive revision to label processing is currently underway to remove ambiguities and provide for the processing of ANSI standard labels. This work was not completed in time to be included in this revision. In order not to hinder the introduction of this new facility, it was decided to define only a minimum label processing capability in the revised standard.

2.3.5 JOD Elements Not Chosen For Standardization

This list represents the language elements in the Journal of Development at the cutoff date (December 31, 1971) which were not chosen for inclusion in the revised standard. Many of these elements were previously excluded from X3.23-1968. An asterisk indicates those elements not available for consideration at the time the original standard was specified. The symbol + represents an element which was in X3.23-1968 but was excluded from the revised standard.

1. The figurative constants: UPPER-BOUND, UPPER-BOUNDS, LOWER-BOUND, and LOWER-BOUNDS.

2. In the SOURCE-COMPUTER paragraph, SUPERVISOR CONTROL, MEMORY SIZE, ADDRESS option, and implementor-name(s).

3. In the OBJECT-COMPUTER paragraph, SUPERVISOR CONTROL, MEMORY SIZE (ADDRESS option), implementor-name(s), and ASSIGN OBJECT-PROGRAM.

4. In the file control entry, ORGANIZATION IS RELATIVE clause for files referenced as the object of:

   a. USING/GIVING phrase of a SORT or MERGE statement

   b. file description entry containing the REPORT clause

5. In the file control entry, ORGANIZATION IS INDEXED clause for files referenced as the object of:

   a. USING/GIVING phrase of a SORT or MERGE statement

   b. file description entry containing the REPORT clause

6. In the file control entry, the PROCESSING MODE clause.

7. In the I-O-CONTROL paragraph, the APPLY clause.

8. In the I-O-CONTROL paragraph, an indexed or relative file may be specified in the ON clause of RERUN.

9. In the I-O-CONTROL paragraph, an indexed or relative file may be specified in the END OF REEL/UNIT clause of RERUN.

+10. In the file description entry, the LABEL RECORDS IS data-name clause.

11. In the file description entry, the RECORDING MODE clause.

12. The saved area description entry.

13. In the PICTURE clause, the DEPENDING ON phrase and the character L.

14. In the USAGE clause, COMPUTATIONAL-n, DISPLAY-n, and INDEX-n.

15. The requirement of supporting more than five levels of qualification for a data-name.

16. The complete set of qualifiers for a data-name may be the same as the partial list of qualifiers for another data-name.

17. The relational operators: UNEQUAL TO, EQUALS, EXCEEDS.

18. In the COMPUTE statement, FROM and EQUALS.

19. In the DISPLAY statement, numeric literal may be signed and/or noninteger.

20. The HOLD statement.

*21. The INITIALIZE statement.

22. In Format 3 of the INSPECT statement, the BEFORE/AFTER REPLACING phrase.

23. In the MOVE CORRESPONDING statement, the identifier series.

24. The PROCESS statement.

25. Dynamic redefinition of the collating sequence by means of the SET statement.

26. In the STOP statement, numeric literal may be signed and/or noninteger.

*27. The SUSPEND statement.

+28. In the USE statement, the LABEL option.

*29. In the USE statement, the RANDOM PROCESSING option.

*30. In the USE statement, recursive invocation of USE procedures.

3. APPENDIX C: CONCEPTS

3.1 FEATURES OF THE LANGUAGE

COBOL offers many features which allow the user to obtain a necessary function without programming the function in detail. In this appendix each of these features and the concept of its use will be discussed.

3.2 RECORD ORDERING

The ability to arrange records into a particular order is a common requirement of the data processing user. The Sort and Merge features of COBOL provide facilities to assist in meeting this requirement.

While both are concerned with record ordering, the functions and capabilities of the SORT and MERGE statements are different in a number of respects. The Sort will produce an ordered file from one or more files that may be completely unordered in the sort sequence whereas the Merge can only produce an ordered file from two or more files each of which is already ordered in the specified sequence.

In many applications it is necessary to apply some special processing to the contents of the sort or merge file(s) before or after sorting or merging. This special processing may consist of addition, deletion, creation, altering, editing, or other modification of the individual records in the file. The COBOL Sort-Merge feature allows the user to express these procedures in the COBOL language. A COBOL program may contain any number of sorts and merges, and each of them may have its own independent special procedures. The Sort-Merge feature automatically causes execution of these procedures in such a way that extra passes over the sort or merge files are not required.

3.3 REPORT WRITER

The Report Writer is a feature which places its emphasis on the organization, format, and contents of an output report. Although a report can be produced using the standard COBOL language, the Report Writer language features provide a more concise facility for report structuring and report production. Much of the Procedure Division programming which would normally be supplied by the programmer is instead provided automatically by the Report Writer Control System (RWCS). Thus the programmer is relieved of writing procedures for moving data, constructing print lines, counting lines on a page, numbering pages, producing heading and footing lines, recognizing the end of logical data subdivisions, updating sum counters, etc. All these operations are accomplished by the RWCS as a consequence of source language statements that appear primarily in the Report Section of the Data Division of the source program.

Data movement to a report is directed by the Report Section clauses SOURCE, SUM, and VALUE. Fields of data are positioned on a print line by means of the COLUMN NUMBER clause. The PAGE clause specifies the length of the page, the size of the heading and footing areas, and the size of the area in which the detail lines will appear. Data items may be specified to form a control hierarchy. During the execution of a GENERATE statement, the Report Writer Control System uses the control hierarchy to check automatically for control breaks. When a control break occurs, summary information (e.g. subtotals) can be presented.

3.4 TABLE HANDLING

Tables of data are common components of business data processing problems. Although items of data that make up a table could be described as contiguous data items, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

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

3.4.1 Table Definition

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

Example 1.

```
01  TABLE-1.
    02  TABLE-ELEMENT  OCCURS 20 TIMES.
        03  NAME .........
        03  SSAN .........
```

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that table element. To define a three-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the description of 2 group items which contain the element. In COBOL, tables of up to 3 dimensions are permitted. Example 2 shows a table which has one dimension for CONTINENT-NAME, two dimensions for COUNTRY-NAME, and three dimensions for CITY-NAME and CITY-POPULATION. The table includes 100,510 data items -- 10 for CONTINENT-NAME, 500 for COUNTRY-NAME, 50,000 for CITY-NAME, and 50,000 for CITY-POPULATION. Within the table there are ten occurrences of CONTINENT-NAME. Within each CONTINENT-NAME there are 50 occurrences of COUNTRY-NAME and within each COUNTRY-NAME there are one hundred occurrences of CITY-NAME and CITY-POPULATION.

Example 2.

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

3.4.2 References to Table Items

Whenever the user refers to a table element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table, the occurrence number of the desired element provides complete information. For access to tables of more than one dimension, an occurrence number must be supplied for each dimension of the table accessed. In Example 2 then, a reference to the 4th CONTINENT-NAME would be complete, whereas a reference to the 4th COUNTRY-NAME would not. To refer to COUNTRY-NAME, which is an element of a two-dimensional table, the user must refer to, for example, the 4th COUNTRY-NAME within the 6th CONTINENT-TABLE.

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

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

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

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

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

3.4.3 Table Searching

Data that has been arranged in the form of a table is very often searched. In COBOL the SEARCH statement provides facilities, through its two options, for producing serial and non-serial (for example, binary) searches. In using the SEARCH statement, the programmer may vary an associated index-name or data-name. This statement also provides facilities for execution of imperative statements when certain conditions are true.

3.5 FILE ORGANIZATION AND ACCESS METHODS

Magnetic tape, punched paper tape, and punched card files are normally organized in a sequential manner and the Procedure Division of COBOL reflects this use. Mass storage media can be used to store sequentially organized files, and this technique has been provided; but, more significantly, mass storage devices have been designed to provide nonsequential organization and access capabilities.

3.5.1 Sequential Organization

A file whose organization is sequential can only be accessed in the sequential mode. Records in such a file can be accessed in the sequence established as a result of writing the records to the file. A sequential mass storage file may be used for input and output at the same time. One file maintenance method made possible by this facility is to read a record, process it, and, if it is updated, return it, modified, to its previous position.

3.5.2 Relative Organization

A file whose organization is relative can be accessed either sequentially, dynamically, or randomly. Sequential access provides the same results as if the file were organized sequentially. Random access allows the sequence in which the records are accessed to be controlled by the programmer. Each record in a relative file is identified by an integer value greater than zero which specifies the record's logical ordinal position in the file. The desired record is accessed by placing its relative record number in a Relative Key data item. Such a file may be thought of as 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 is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through the ninth record areas.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

3.5.3 Indexed Organization

A file whose organization is indexed can be accessed either sequentially, dynamically, or randomly. Sequential access provides access to the records in 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 order 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. Each record in the file is identified by the value of one or more keys within that record, and the desired record is accessed by placing the value of its record key in a record key data item before accessing the record.

In the dynamic access mode, the programmer may change at will from sequential access to random access by using appropriate forms of input-output statements.

3.6 RERUN

The RERUN feature of COBOL provides a facility for check restart. That is, executing a RERUN takes a snapshot of the program status and stores the information. It is then possible to restart the program from the point of the most recent RERUN. The use of the RERUN clause protects the user from having to start a program over from the beginning in the event of a hardware failure while the job is running.

There are two basic parts to the RERUN clause. The user must designate a medium to receive the data and a criterion from which the frequency of checkpoints may be determined. The receiving medium may be specified by designating a file name or a separate hardware device. The determination of frequency of the dump may be made on the basis of a number of records of a particular file having been processed, of the end of a reel of a particular file having been reached, of the setting of a hardware switch or of a specified number of units of an internal clock having been counted.

3.7 PROGRAM MODULARITY

Complex data processing problems are frequently solved by the use of separately compiled but logically coordinated programs, which, at execution time, form logical and physical subdivisions of a single run unit. This approach lends itself to dividing a large problem into smaller, more manageable segments which can be programmed and debugged independently. At execute time, control is transferred from program to program by the use of CALL and EXIT PROGRAM statements.

Under certain circumstances, e.g., a shortage of computer storage, it is desirable to subdivide a single program into physical segments, so that, at execute time, it is not necessary to load the entire program into computer storage at one time. This approach would permit the overlaying of some segments, with a corresponding saving in total computer storage required to execute the program. This facility is called segmentation.

There are no special statements in COBOL for communication between segments of such a program. There are, however, some special clauses used by the COBOL programmer to specify how the object program is to be segmented.

3.7.1 Inter-Program Communication

In COBOL terminology, a program is either a source program or an object program depending on context; a source program is a syntactically correct set of COBOL statements; an object program is the set of instructions, constants, and other machine-oriented data resulting from the operation of a compiler on a source program; and a run unit is the total machine language necessary to solve a data processing problem. It includes one or more object programs as defined above, and it may include machine language from sources other than a COBOL compiler.

When the statement of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other. This communication may take two forms: transfer of control and reference to common data.

3.7.1.1 Transfer of Control

The CALL statement provides the means whereby control can be passed from one program to another within a run unit. A program that is activated by a CALL statement may itself contain CALL statements. However, results are unpredictable where circularity of control is initiated; i.e., where program A calls program B, then program B calls program A or another program that calls program A.

When control is passed to a called program, execution proceeds in the normal way from procedure statement to procedure statement beginning with the first nondeclarative statement. If control reaches a STOP RUN statement, this signals the logical end of the run unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the point immediately following the CALL statement in the calling program. Stated briefly, the EXIT PROGRAM statement terminates only the program in which it occurs, and the STOP RUN statement terminates the entire run unit.

If the called program is not COBOL then the termination of the run unit or the return to the calling program must be programmed in accordance with the language of the called program.

3.7.1.2 Inter-Program Data Storage

Program interaction requires that both programs have access to the same data items. In the calling program the common data items are described along with all other data items in the File Section, Working-Storage Section, Communication Section, or Linkage Section. At object time memory is allocated for the entire Data Division. In the called program, common data items are described in the Linkage Section. At object time memory space is not allocated for this section. Communication between the called program and the common data items stored in the calling program is effected through USING clauses contained in both programs. The USING clause in the calling program is contained in the CALL statement and the operands are a list of common data-identifiers described in its Data Division. The USING clause in the called program follows the Procedure Division header and the operands are a list of common data identifiers described in its Linkage Section. The identifiers specified by the USING clause of the CALL statement indicate those data items available to a calling program that may be referred to in the called program. The sequence of appearance of the identifiers in the USING clause of the CALL statement and the USING clause in the Procedure Division header is significant. Corresponding identifiers refer to a single set of data which is available to the calling program. The correspondence is positional, and not by name. While the called program is being executed, every reference to an operand whose identifier appears in the called program's USING clause is treated as if it were a reference to the corresponding operand in the USING clause of the active CALL statement.

Once control leaves a called program its state is maintained until a CANCEL is executed naming that program. Therefore, initialization of the program in case of repetitive calls is not necessary.

Execution of the CANCEL statement allows the user to indicate that the memory areas occupied by the called program(s) may be released. In addition, the CANCEL guarantees that the program cancelled will be in its initial state when called by a subsequent CALL statement.

3.7.2 Segmentation

The segmentation facility permits the user to subdivide physically the Procedure Division of a COBOL object program. All source paragraphs which contain the same segment-number in their section headers will be considered at object time to be one segment. Since segment-numbers can range from 00 through 99, it is possible to subdivide any object program into a maximum of 100 segments.

Program segments may be of three types: fixed permanent, fixed overlayable, and independent as determined by the programer's assignment of segment numbers.

Fixed segments are always in computer storage during the execution of the entire program, i.e., they cannot be overlayed except when the system is executing another program, in which case fixed segments may be 'rolled out' temporarily.

Fixed overlayable segments may be overlayed during program execution, but any such overlaying is transparent to the user, i.e., they are logically identical to fixed segments, but physically different from them.

Independent segments may be overlayed, but such overlaying will result in the initialization of that segment. Therefore, independent segments are logically different from fixed permanent/fixed overlayable segments, and physically different from fixed segments.

3.8 COMMUNICATION FACILITY

The communication facility provides the ability to access, process, and create messages or portions thereof. It provides the ability to communicate through a Message Control System with local and remote communication devices.

3.8.1 The Message Control System

The implementation of the communication facility requires that a Message Control System (MCS) be present in the COBOL object program's operating environment.

The MCS is the logical interface to the operating system under which the COBOL object program operates. The primary functions of the MCS are the following:

(1) To act as an interface between the COBOL object program and the network of communication devices, in much the same manner as an operating system acts as an interface between the COBOL object program and such devices as card readers, magnetic tape and mass storage devices, and printers.

(2) To perform line discipline, including such tasks as dial-up, polling, and synchronization.

(3) To perform device-dependent tasks, such as character translation and insertion of control characters, so that the COBOL user can create device-independent programs.

The first function, that of interfacing the COBOL object program with the communication devices, is the most obvious to the COBOL user. In fact, the COBOL user may be totally unaware that the other two functions exist. Messages from communication devices are placed in input queues by the MCS while awaiting disposition by the COBOL object program. Output messages from the COBOL object program are placed in output queues by the MCS while awaiting transmission to communication devices. The structures, formats, and symbolic names of the queues are defined by the user to the MCS at some time prior to the execution of the COBOL object program. Symbolic names for message sources and destinations are also defined at that time. The COBOL user must specify in his COBOL program symbolic names which are known to the MCS.

During execution of a COBOL object program, the MCS performs all necessary actions to update the various queues as required.

3.8.2 The COBOL Object Program

The COBOL object program interfaces with the MCS when it is necessary to send data, receive data, or to interrogate the status of the various queues which are created and maintained by the MCS. In addition, the COBOL object program may direct the MCS to establish or break the logical connection

between the communication device and a specified portion of the MCS queue structure. The method of handling the physical connection is a function of the MCS.

3.8.3 Relationship of the COBOL Program to the Message Control System and Communication Devices

The interfaces which exist in a COBOL communication environment are established by the use of a CD and associated clauses in the Communication Section of the Data Division. There are two such interfaces:

(1) The interface between the COBOL object program and the MCS, and;

(2) The interface between the MCS and the communication devices.

The COBOL source program uses three statements to control the interface with the MCS:

(1) The RECEIVE statement, which causes data in a queue to be passed to the COBOL object program,

(2) The SEND statement, which causes data associated with the COBOL object program to be passed to one or more queues, and;

(3) The ACCEPT statement with the COUNT phrase, which causes the MCS to indicate to the COBOL object program the number of complete messages in the specified queue structure.

The COBOL source program uses two statements to control the interface between the MCS and the communication devices:

(1) The ENABLE statement, which establishes logical connection between the MCS and one or more given communication devices, and;

(2) The DISABLE statement, which breaks a logical connection between the MCS and one or more given communication devices.

These relationships are shown in Figure 1, COBOL Communication Environment, which is located on page XIV-44.

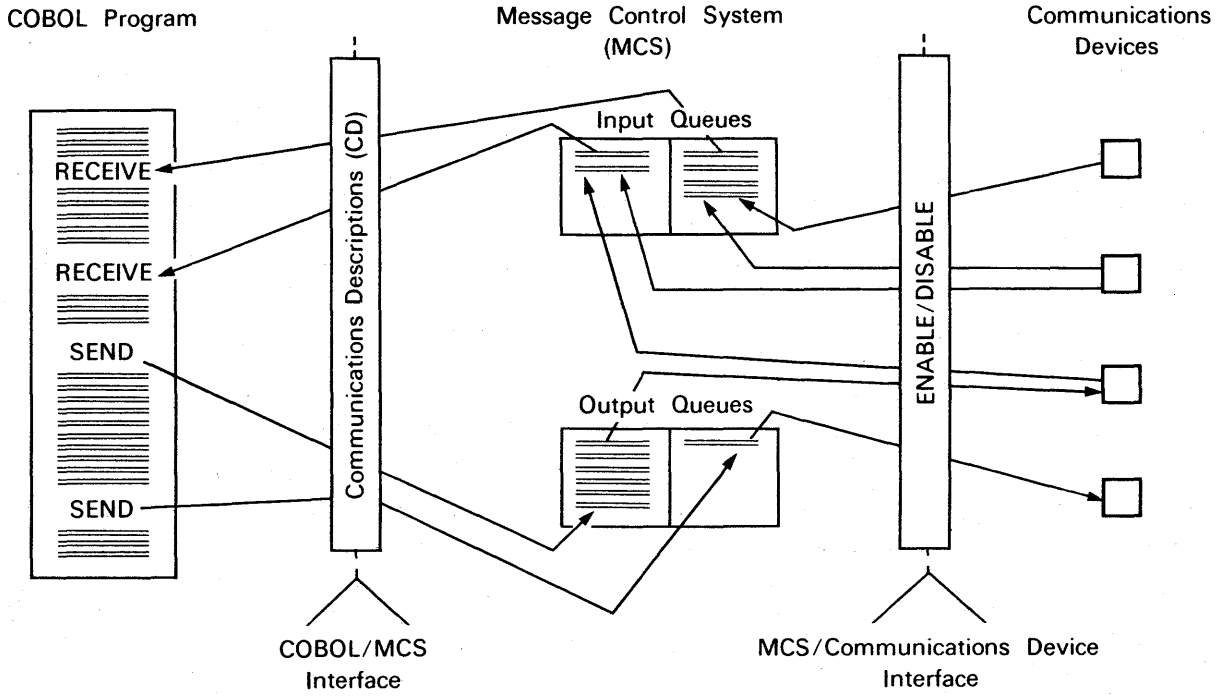


Figure 1: COBOL Communication Environment

3.8.3.1 Invoking the COBOL Object Program

There are two methods of invoking a COBOL communication object program:

(1) Schedule initiation

(2) MCS invocation

Regardless of the method of invocation, the only operating difference between the two methods is that MCS invocation causes the areas referenced by the symbolic queue and subqueue names in the specified CD to be filled.

3.8.3.1.1 Scheduled Initiation of the COBOL Object Program

A COBOL object program using the communication facility may be scheduled for execution through the normal means available in the program's operating environment, such as job control language. In that case, the COBOL program can use three methods to determine what messages, if any, are available in the input queues:

(1) The ACCEPT statement with the COUNT phrase,

(2) The RECEIVE statement with a NO DATA phrase, and

(3) The RECEIVE statement without a NO DATA phrase (in which case a program wait is implied if no data is available).

3.8.3.1.2 Invocation of the COBOL Object Program by the MCS

It is sometimes desirable to schedule a COBOL object communication program only when there is work available for it to do. Such scheduling occurs if the MCS determines what COBOL object program is required to process the available message and subsequently causes that program to be scheduled for execution. Prior to the execution of the COBOL object program, the MCS places symbolic queue and sub-queue names in the data items of the CD that specifies the FOR INITIAL INPUT clause.

A subsequent RECEIVE statement directed to that CD will result in the available message being passed to the COBOL object program.

3.8.3.1.3 Determining the Method of Scheduling

A COBOL source program can be written so that its object program can operate with either of the two modes of scheduling. In order to determine which method was used to load the COBOL object program, the following is one technique that may be used:

(1) One CD must contain a FOR INITIAL INPUT clause.

(2) The Procedure Division may contain statements to test the initial value of the symbolic queue name in that CD. If it is space-filled, job control statements were used to schedule the COBOL object program. If not space filled, the MCS has invoked the COBOL object program and replaced the spaces with the symbolic name of the queue containing the message to be processed.

3.8.4 The Concept of Messages and Message Segments

A message consists of some arbitrary amount of information, usually character data, whose beginning and end are defined or implied. As such, messages comprise the fundamental but not necessarily the most elementary unit of data to be processed in a COBOL communication environment.

Messages may be logically subdivided into smaller units of data called message segments which are delimited within a message by means of end of segment indicators (ESI). A message consisting of one or more segments is delimited from the next message by means of an end of message indicator (EMI). In a similar manner, a group of several messages may be logically separated from succeeding messages by means of an end of group indicator (EGI). When a message or message segment is received by the COBOL program, a communication description interface area is updated by the MCS to indicate which, if any, delimiter was associated with the text transferred during the execution of that RECEIVE statement. On output the delimiter, if any, to be associated with the text released to the MCS during execution of a SEND statement is specified or referenced in the SEND statement. Thus the presence of these logical indicators is recognized and specified both by the MCS and by the COBOL object program; however, no indicators are included in the message text processed by COBOL programs.

A precedence relationship exists between the indicators EGI, EMI and ESI. EGI is the most inclusive indicator and ESI is the least inclusive indicator. The existence of an indicator associated with message text implies the

association of all less inclusive indicators with that text. For example, the existence of the EGI implies the existence of EMI and ESI.

3.8.5 The Concept of Queues

Queues consist of one or more messages from or to one or more communication devices, and as such, form the data buffers between the COBOL object program and the MCS. Input queues are logically separate from output queues.

The MCS logically places in queues or removes from queues only complete messages. Portions of messages are not logically placed in queues until the entire message is available to the MCS. That is, the MCS will not pass a message segment to a COBOL object program unless all segments of that message are in the input queue, even though the COBOL source program uses the SEGMENT phrase of the RECEIVE statement. For output messages, the MCS will not transmit any segment of a message until all its segments are in the output queue. The number of messages that exist in a given queue reflects only the number of complete messages that exist in the queue.

The process by which messages are placed into a queue is called enqueueing. The process by which messages are removed from a queue is called dequeueing.

3.8.5.1 Independent Enqueueing and Dequeueing

It is possible that a message may be received by the MCS from a communication device prior to the execution of the COBOL object program. In this case the MCS enqueues the message in the proper input queue until the COBOL object program requests dequeueing with the RECEIVE statement. It is also possible that a COBOL object program will cause the enqueueing of messages in an output queue which are not transmitted to a communication device until after the COBOL object program has terminated. Two common reasons for this occurrence are:

(1) When data transfer between the specified output queue and its destination is inhibited.

(2) When the COBOL object program creates output messages at a speed faster than the destination can receive them.

3.8.5.2 Enabling and Disabling Logical Connectives

Usually, the MCS will logically connect and disconnect sources and destinations based on time of day, message activity, or other factors unrelated to the COBOL program. However, the COBOL program has the ability to perform these functions through use of the ENABLE and DISABLE statements.

A key is required in both statements in order to prevent indiscriminate use of the facility by a COBOL user who is not aware of the total network environment, and who may therefore disrupt system functions by the untimely issuance of ENABLE and DISABLE statements. However, this action never interrupts a transmission.

3.8.5.3 Enqueueing and Dequeueing Methods

In systems that allow the user to specify certain MCS functions, it may be necessary that the user specify to the MCS, prior to execution of programs which reference these facilities, the selection algorithm and other designated MCS functions to be used by the MCS in placing messages in the various queues. A typical selection algorithm, for example, would specify that all messages from a given source be placed in a given input queue, or that all messages to be sent to a given destination be placed in a given output queue.

Dequeueing is often done on a first in, first out basis. Thus, messages dequeued from either an input or output queue are those messages which have been in the queue for the longest period of time. However, the MCS can, upon prior specification by the user, dequeue on some other basis, i.e., priority queueing can be employed.

3.8.5.4 Queue Hierarchy

In order to control more explicitly the messages being enqueued and dequeued, it is possible to define in the MCS a hierarchy of input queues, i.e., queues comprising queues. In COBOL, four levels of queues are available to the user. In order of decreasing significance, the queue levels are named queue, sub-queue-1, sub-queue-2 and sub-queue-3. The full queue structure is depicted in Figure 2, Hierarchy of Queues, where queues and sub-queues have been named with the letters A through O. Messages have been named with a letter according to their source (X, Y, or Z) and with a sequential number.

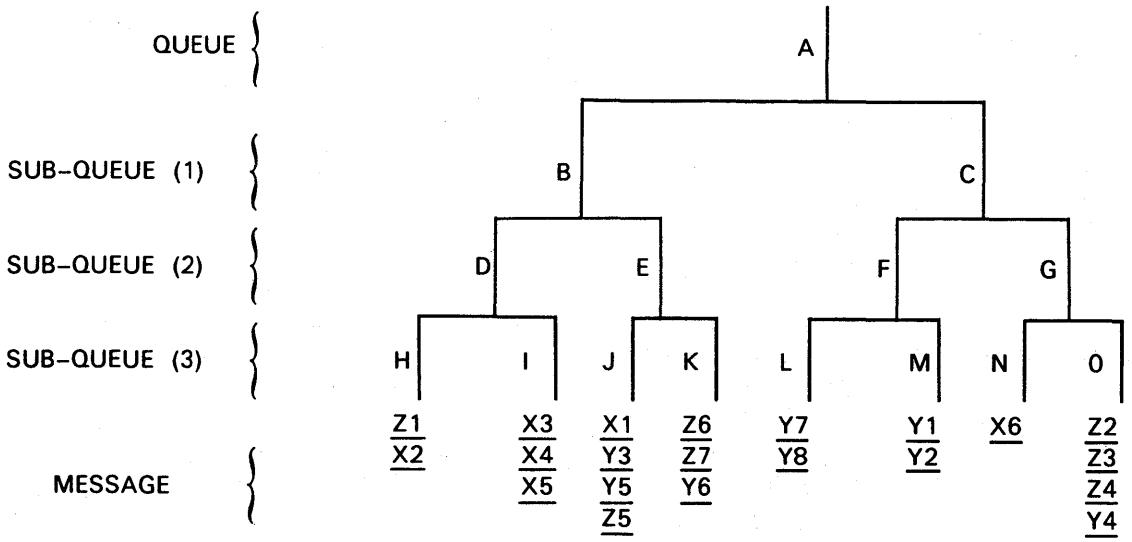


Figure 2: Hierarchy of Queues

Let us assume that the MCS is operating under the following queueing algorithm:

(1) Messages are placed in queues according to the contents of some specified data field in each message.

(2) With the RECEIVE statement, if the user does not specify a given sub-queue level, the MCS will choose the sub-queue from that level in alphabetical order, e.g., if sub-queue-1 is not specified by the user, the MCS will dequeue from sub-queue-1 B.

The following examples illustrate the effect of the algorithms shown in Figure 2 on page XIV-47:

(1) The program executes a RECEIVE statement, specifying via the CD:

Queue A

MCS returns: Message Z1

(2) The program executes a RECEIVE statement, specifying via the CD:

Queue A

Sub-queue-1 C

MCS returns: Message Y7

(3) The program executes a RECEIVE statement, specifying via the CD:

Queue A

Sub-queue-1 B

Sub-queue-2 E

MCS returns: Message X1

(4) The program executes a RECEIVE statement, specifying via the CD:

Queue A

Sub-queue-1 C

Sub-queue-2 G

Sub-queue-3 N

MCS returns: Message X6

If the COBOL programmer wishes to access the next message in a queue, regardless of which sub-queue that message may be in, he specifies the queue name only. The MCS, when supplying the message, will return to the COBOL object program, any applicable sub-queue names via the data items in the associated CD. If, however, he desires the next message in a given sub-queue, he must specify both the queue name and any applicable sub-queue names.

For output, the COBOL user specifies only the destination(s) of the message, and the MCS places the message in the proper output queue structure.

There is no one-to-one relationship between a communication device and a source/destination. A source or destination may consist of one or more physical devices. The device or devices which comprise a source/destination are defined to the MCS.

3.9 DEBUGGING

To assist in error detection, COBOL provides the facility to monitor, during program execution:

(1) transfers of control to user selected procedures and

(2) values of user selected data items.

The user statements required to accomplish this monitoring are included in the source program and can be compiled or not according to the presence or absence of one clause in the source program. Once compiled into the program, these statements may be executed or ignored at run time according to the setting of a run-time switch.

3.10 LIBRARY

The library feature provides the facility to copy source text from a "library" of source text material that is available at compile time. A short phrase can cause inclusion of large amounts of source library material into the source program, thus saving repetitious coding. Once established, a source library may be referenced many times by many programs.

INDEX

American National Standards on Computers and Information Processing

X3.1-1976 Synchronous Signaling Rates for Data Transmission

X3.2-1970 (R1976) Print Specifications for Magnetic Ink Character Recognition

X3.3-1970 (R1976) Bank Check Specifications for Magnetic Ink Character Recognition

X3.4-1968 Code for Information Interchange

X3.5-1970 Flowchart Symbols and Their Usage in Information Processing

X3.6-1965 (R1973) Perforated Tape Code for Information Interchange

X3.9-1966 FORTRAN

X3.10-1966 Basic FORTRAN

X3.11-1969 Specification for General Purpose Paper Cards for Information Processing

X3.12-1970 Vocabulary for Information Processing

X3.14-1973 Recorded Magnetic Tape for Information Interchange (200 CPI, NRZI)

X3.15-1976 Bit Sequencing of the American National Standard Code for Information Interchange in Serial-by-Bit Data Transmission

X3.16-1976 Character Structure and Character Parity Sense for Serial-by-Bit Data Communication in the American National Standard Code for Information Interchange

X3.17-1974 Character Set and Print Quality for Optical Character Recognition (OCR-A)

X3.18-1974 One-Inch Perforated Paper Tape for Information Interchange

X3.19-1974 Eleven-Sixteenths-Inch Perforated Paper Tape for Information Interchange

X3.20-1967 (R1974) Take-Up Reels for One-Inch Perforated Tape for Information Interchange

X3.21-1967 Rectangular Holes in Twelve-Row Punched Cards

X3.22-1973 Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI)

X3.23-1974 Programming Language COBOL

X3.24-1968 Signal Quality at Interface between Data Processing Terminal Equipment and Synchronous Data Communication Equipment for Serial Data Transmission

X3.25-1976 Character Structure and Character Parity Sense for Parallel-by-Bit Communication in the American National Standard Code for Information Interchange

X3.26-1970 Hollerith Punched Card Code

X3.27-1969 Magnetic Tape Labels for Information Interchange

X3.28-1976 Procedures for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links

X3.29-1971 Specifications for Properties of Unpunched Oiled Paper Perforator Tape

X3.30-1971 Representation for Calendar Date and Ordinal Date for Information Interchange

X3.31-1973 Structure for the Identification of the Counties of the United States for Information Interchange

X3.32-1973 Graphic Representation of the Control Characters of American National Standard Code for Information Interchange

X3.34-1972 Interchange Rolls of Perforated Tape for Information Interchange

X3.36-1975 Synchronous High-Speed Data Signaling Rates between Data Terminal Equipment and Data Communication Equipment

X3.37-1974 Programming Language APT

X3.38-1972 Identification of States of the United States (Including the District of Columbia) for Information Interchange

X3.39-1973 Recorded Magnetic Tape for Information Interchange (1600 CPI, PE)

X3.40-1976 Unrecorded Magnetic Tape for Information Interchange (9-Track 200 and 800 CPI, NRZI, and 1600 CPI, PE)

X3.41-1974 Code Extension Techniques for Use with the 7-Bit Coded Character Set of American National Standard Code for Information Interchange

X3.42-1975 Representation of Numeric Values in Character Strings for Information Interchange

X3.44-1974 Determination of the Performance of Data Communication Systems

X3.45-1974 Character Set for Handprinting

X3.46-1974 Unrecorded Magnetic Six-Disk Pack (General, Physical, and Magnetic Characteristics)

X3.49-1975 Character Set for Optical Character Recognition (OCR-B)

X3.50-1976 Representations for U.S. Customary, SI, and Other Units to Be Used in Systems with Limited Character Sets

X3.51-1975 Representations of Universal Time, Local Time Differentials, and United States Time Zone References for Information Interchange

X3.52-1976 Unrecorded Single-Disk Cartridge (Front Loading, 2200 BPI), General, Physical, and Magnetic Requirements

X3.53-1976 Programming Language PL/I

X3.54-1976 Recorded Magnetic Tape for Information Interchange (6250 CPI, Group Coded Recording)

For a free and complete list of all American National Standards, write:

American National Standards Institute, Inc
1430 Broadway
New York, N.Y. 10018