UNISYS

PUBLICATIONS UPDATE

OS/3

1974 American Standard COBOL Programming Reference Manual

UP-8613 Rev. 2-F

This Library Memo announces the release and availability of Update F to the *OS/3 1974 American Standard COBOL Programming Reference Manual*, UP-8613 Rev. 2.

This manual is a standard library item (SLI). It is part of the standard library provided automatically with the purchase of the product.

This manual presents the rules for writing COBOL programs compiled by the 1974 American National Standard COBOL compiler and executed under the control of the Unisys Operating System/3 (OS/3). The COBOL language described in this manual conforms to the specifications of the American National Standard COBOL, X3.23-1974 and supports Federal Information Processing Standards (FIPS) Publication 21-1.

The update for release 12.0 provides an additional parameter (SUBCK) for the PARAM job control statement, which you can use to specify that the compiler is to generate code to check for subscript or out-of-range conditions. Several diagnostic messages have been added and changed as well.

All other changes in this document are expanded descriptions applicable to items present in the software prior to this release.

Copies of Update F are now available. You can order the update only, or the complete manual with all updates, through your local Unisys representative. To receive only the update, order UP-8613 Rev. 2-F. To receive the complete manual, order UP-8613 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists MBZ, MCZ, MMZ, M18, M18U, M19, M20, M21, M28U, M29U, M75, M75U, M76, and M76U	Mailing Lists MB00, MBW, and MB13 (49 pages plus Memo)	Library Memo for UP-8613 Rev. 2-F
		RELEASE DATE: October 1988

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

1974 American National Standard COBOL Programmer Reference

UP-8613 Rev. 2-D

This Library Memo announces the release and availability of Update D to "SPERRY® Operating System/3 (OS/3) 1974 American National Standard COBOL Programmer Reference", UP-8613 Rev. 2.

This manual presents the rules for writing COBOL programs compiled by the 1974 American National Standard COBOL compiler and executed under the control of the operating system. The COBOL language described in this manual conforms to the specifications of the American National Standard COBOL, X3.23-1974 and supports Federal Information Processing Standards Publication 21-1.

This update for release 10.0 provides:

- An additional parameter for the IMSCOD option of the COBOL PARAM statement. You can now select IMSCOD=REN to compile reentrant action programs.
- A new compiler diagnostics message concerning reentrant action programs.
- New information about compiler parameter specification and work area usage for reentrant action programs.

All other changes are technical corrections or clarifications that apply to American National Standard COBOL prior to this release.

Copies of Update D are now available. You can order the update only or the complete manual with the update through your local Sperry representative. To receive only the update, order UP-8613 Rev. 2-D. To receive the complete manual, order UP-8613 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ, MZ, 18, 18U, 19, 19U, 20, 21, 21U, 28U, 29U, 75, 75U, 76 and 76U	Mailing Lists AF01, A00, A13, B00, MBW, and B13 (Package D to UP-8613 Rev. 2, 81 pages plus Memo)	Library Memo for UP-8613 Rev. 2-D RELEASE DATE: May 1986

Horst

SPERRY

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

SPERRY UNIVAC OS/3
1974 AMERICAN NATIONAL
STANDARD COBOL

Programmer Reference

UP-8613 Rev. 2 – C

This Library Memo announces the release and availability of Updating Package C to "SPERRY® Operating System/3 (OS/3) 1974 American National Standard COBOL Programmer Reference", UP-8613 Rev. 2.

This manual presents the rules for writing COBOL programs to be compiled by the 1974 American National Standard COBOL compiler and executed under the control of the operating system. The COBOL language described in this manual conforms to the specifications of the American National Standard COBOL, X3.23–1974 and supports Federal Information Processing Standard Publication 21–1.

This update includes information on the compilation summary listing and updates the procedure call statement for the job control stream requirements (Appendix H).

All other changes are corrections or expanded descriptions applicable to features present in 1974 American National Standard COBOL to the 8.2 release.

Copies of Updating Package C are now available for requisitioning. Either the updating package only or the complete manuals with the updating package may be requisitioned by your local Sperry representative. To receive only the updating package, order UP-8613 Rev. 2–C. To receive the complete manual, order UP-8613 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ, MZ, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76 and 76U	Mailing Lists A00, A13, B00, and B13 (Package C to UP-8613 Rev. 2,) 41 pages plus Memo)	Library Memo for UP-8613 Rev.2–C RELEASE DATE: November, 1984

SPERRY

Horst.

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

1974 American National Standard COBOL

Programmer Reference

UP-8613 Rev. 2-B

This Library Memo announces the release and availability of Updating Package B to "SPERRY Operating System/3 (OS/3) 1974 American National Standard COBOL Programmer Reference", UP-8613 Rev. 2.

This update for release 8.2 provides the following:

- Additional names for the SOURCE–COMPUTER and OBJECT–COMPUTER paragraphs
- An additional name for the SPECIAL–NAMES paragraph
- Information about size of COMPUTATIONAL items in a PICTURE character string
- Additional information on the procedure division storage map listings
- Additional reserved words

All other changes are technical corrections or clarifications applicable to American National Standard COBOL prior to release 8.2.

Copies of Updating Package B are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry representative. To receive only the updating package, order UP–8613 Rev. 2–B. To receive the complete manual, order UP–8613 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS:
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A13, B00, B13, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76 and 76U (Package B to UP–8613 Rev. 2, 82 pages plus Memo)	Library Memo for UP-8613 Rev. 2–B RELEASE DATE: February, 1984

Horst

SPERRY UNIVAC

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

1974 American National Standard COBOL

Programmer Reference

UP-8613 Rev. 2-A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) 1974 American National Standard COBOL Programmer Reference", UP-8613 Rev. 2.

This update incorporates additional information about 1974 American National Standard COBOL for release 8.0:

- RESERVE clause
- OCCURS DEPENDING clause

All other changes are corrections or expanded descriptions applicable to features present in 1974 American National Standard COBOL prior to the 8.0 release.

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8613 Rev. 2—A. To receive the complete manual, order UP-8613 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00,A13,B00,B13,18,18U, 19,19U,20,20U,21,21U,28U,29U,75,75U, 76 and 76U (Package A to UP-8613 Rev. 2, 39 pages plus Memo)	Library Memo for UP-8613 Rev. 2-A RELEASE DATE: January, 1983

SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

1974 American National Standard COBOL

Programmer Reference

UP-8613 Rev. 2

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) 1974 American National Standard COBOL Programmer Reference", UP-8613 Rev. 2.

This revision describes the following 1974 ANS COBOL features for release 8.0:

- Multiworkstation support through ACCEPT and DISPLAY statements
- Error file processing
- ERRFIL parameter
- CDMIO parameter
- SYSTEM LINES options of the LINAGE clause

All other changes are corrections or expanded descriptions applicable to features present in 1974 ANS COBOL before the 8.0 release.

Destruction Notice: If you are going to OS/3 release 8.0, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.0, retain the copy you are now using and store this revision for future use.

Copies of UP-8613 Rev. 1 and UP-8613 Rev. 1—A will be available for 6 months after the release of 8.0. Should you need additional copies of this edition, you should order them within 90 days of the release of 8.0. When ordering the previous edition of a manual, be sure to identify the exact revision and update packages desired and indicate that they are needed to support an earlier release.

Additional copies may be ordered by your local Sperry Univac representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A13, B00, B13, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 28U, 29U, 75, 75U, 76, and 76U (Cover and 411 pages)	Library Memo for UP-8613 Rev. 2
		RELEASE DATE:
		September, 1982

UNISYS

OS/3
1974 American Standard COBOL

Programming Reference Manual

Relative to Release
Level 10.0

Priced Item

August 1987

Printed in U S America
UP–8613 Rev. 2

UNISYS

OS/3 1974 American Standard COBOL

Programming Reference Manual

Previous Title: OS/3 1974 American Standard
COBOL Programmer Reference

Relative to Release
Level 10.0

Priced Item

August 1987

Printed in U S America
UP–8613 Rev. 2

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

FASTRAND, ✦SPERRY, SPERRY✦UNIVAC, SPERRY, SPERRY UNIVAC, UNISCOPE, UNISERVO, UNIS, UNIVAC, and ✦ are registered trademarks of Unisys Corporation. ESCORT, PAGEWRITER, PIXIE, PC/IT, PC/HT, PC/microIT, SPERRYLINK, and USERNET are additional trademarks of Unisys Corporation. MAPPER is a registered trademark and service mark of Unisys Corporation. CUSTOMCARE is a service mark of Unisys Corporation.

PAGE STATUS SUMMARY

ISSUE: Update F – UP-8613 Rev. 2
RELEASE LEVEL: 12.0 Forward

Part/Section	Page Number	Update Level
Cover		E
Title Page/Disclaimer		E
PSS	1, 2	F
Acknowledgment	1	Orig.
Preface	1	B
	2	Orig.
Contents	1	Orig.
	2	D
	3,4	Orig.
	5	F
	6 thru 8	Orig.
	9	C
	10 thru 12	D
1	Tab Breaker	D
	1 thru 5	Orig.
2	Tab Breaker	D
	1 thru 9	Orig.
	10	C
	11 thru 17	Orig.
3	Tab Breaker	D
	1, 2	Orig.
4	Tab Breaker	D
	1, 2	Orig.
	3, 4	F
	4a	F*
	5	C
	6	B
	6a	C
	7	F
	8	C
	9	Orig.
	10	C
	11	A
	12, 13	Orig.
	14	B
	15 thru 17	Orig.
	18, 19	C
	20	A
	20a	A
	21	Orig.
	22	A
	23,24	Orig.
	25,26	F
5	Tab Breaker	D
	1 thru 7	Orig.
	8	D
5 (cont)	9,10	F
	11	D
	12 thru 15	Orig.
	16	B
	17, 18	D
	19 thru 22	Orig.
	23, 24	B
	25 thru 28	Orig.
	29	A
	30, 31	Orig.
	32	B
	32a	C
	33 thru 36	B
	37 thru 62	Orig.
6	Tab Breaker	D
	1 thru 20	Orig.
	21	C
	22	B
	22a	D
	23	D
	24	C
	25,26	Orig.
	27,28	F
	29 thru 32	Orig.
	33	A
	34	F
	35 thru 37	Orig.
	38 thru 40	A
	41 thru 60	Orig.
	61,62	F
	63 thru 114	Orig.
	115	B
	116	A
	117 thru 120	Orig.
7	Tab Breaker	D
	1 thru 3	Orig.
	4	F
	5	F*
8	Tab Breaker	D
	1,2	Orig.
	3,4	F
	5,6	Orig.
	7	D
	8	Orig.
	9	B
	10	Orig.
	11	B
	12 thru 14	Orig.
	15	B
	16, 17	Orig.
9	Tab Breaker	D
	1 thru 4	Orig.
	5	D
10	Tab Breaker	D
	1 thru 3	Orig.
	4, 5	F
11	Tab Breaker	D
	1	C
	2	Orig.
12	Tab Breaker	D
	1 thru 5	Orig.
13	Tab Breaker	D
	1 thru 4	Orig.
14	Tab Breaker	D
	1 thru 7	Orig.
	Tab Breaker	D
Appendix A	1, 2	Orig.
	3,4	F
	5	D
Appendix B	1	C
	2	B
	3	F
Appendix C	1 thru 10	Orig.
	11,12	F
	12a	F*
	13 thru 15	Orig.
	16	F
	17 thru 20	Orig.
	21, 22	F
	22a	F*
	23 thru 26	Orig.
	27, 28	D
	29 thru 31	Orig.
	32	F
	32a	F*
	33	B
	34	F
Appendix D	1, 2	Orig.
Appendix E	1 thru 4	Orig.
Appendix F	1, 2	Orig.
	3	A
	4, 5	B
	6	Orig.
Appendix G	1 thru 3	D
	4, 5	D

*New pages

All the technical changes are denoted by an arrow (⟹) in the margin. A downward pointing arrow (⇓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (⇑) is found. A horizontal arrow (⟹) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

PAGE STATUS SUMMARY

ISSUE: Update F – UP-8613 Rev. 2
RELEASE LEVEL: 12.0 Forward

Part/Section	Page Number	Update Level
Appendix H	1, 2	C
	3, 4	F
	5 thru 16	Orig.
Appendix I	1,2	B
	3	F
Appendix J	1 thru 4	Orig.
Appendix K	1	B
	2	C
	3, 4	B
	5	C
	6	D
	7 thru 9	B
	10	D
	11, 12	B
	13	D
	14	D
Glossary	1 thru 17	Orig.
	18 thru 20	D
	20a	D*
	21 thru 23	Orig.
Index	Tab Breaker	D
	1, 2	Orig.
	3	A
	4	Orig.
	5	D
	6	Orig.
	7	B
	8	F
	9	B
	10	F
	11	Orig.
User Comments		

Part/Section	Page Number	Update Level

Part/Section	Page Number	Update Level

*New pages

All the technical changes are denoted by an arrow (⟹) in the margin. A downward pointing arrow (⇓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (⇑) is found. A horizontal arrow (⟹) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Acknowledgment

The following acknowledgment is reproduced from the *American National Standard COBOL, X3.23—1974* as requested in that publication:

"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 Corporation), Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry 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."

Preface

This manual presents the rules for writing COBOL programs to be compiled by the 1974 American National Standard COBOL compiler and executed under the control of the operating system. The COBOL language described in this manual conforms to the specifications of the American National Standard COBOL, X3.23-1974 and supports Federal Information Processing Standards Publication 21-1 (Appendix D).

Sections 1 and 2 explain the overall structure of the language and introduce the language elements. Section 1 describes the structure of the language and the rules for its use; Section 2 provides the general specifications of the language.

Sections 3 through 6 provide a detailed description of the four divisions of the language: identification division, environment division, data division, and procedure division.

Sections 7 through 14 summarize the following COBOL features: table handling, file processing techniques, sort/merge, segmentation, library usage, debugging language, interprogram communications, and communication facility.

The appendixes provide the following information:

- Appendixes A, B, and C provide compiler options, listings, and diagnostics, respectively.
- Appendix D explains the Federal Information Processing Standard (FIPS PUB 21-1).
- Appendix E describes intermediate results in arithmetic operations.
- Appendix F describes the non-English language feature.
- Appendix G describes IMS COBOL action programs.
- Appendix H describes job control stream requirements for invoking the COBL74 compiler.
- Appendix I provides a listing of reserved words.
- Appendix J contains the character sets.
- Appendix K provides a tutorial description of the PICTURE clause.

Other current OS/3 publications, referenced in this manual, will be necessary or useful to the programmer working with COBOL.

- System 80

Document name and number	Description
Consolidated data management macro language user guide/programmer reference, UP-8826	Describes the data management macroinstructions
System service programs user guide, UP-8841	Describes various system utilities, such as librarian, linkage editor, etc

- Series 90 systems (90/25, 30, 30 B, and 40)

Document name and number	Description
Data management user guide, UP-8068	Describes the data management macroinstructions
System service programs user guide, UP-8062	Describes various system utilities, such as librarian, linkage editor, etc

- Both System 80 and Series 90 Systems

Document name and number	Description
System messages programmer/operator reference, UP-8076	Lists and describes the system console messages issued during comilation by COBOL
Job control user guide, UP-8065	Provides information on the format and usage of job control statements and linkage editor job control procedure call (jproc)
COBOL editor user guide/ programmer reference, UP-9106	Describes the use of the COBOL editor to create and update COBOL source code
General editor user guide/ programmer reference, UP-8828	Describes the general editor, including its use in entering COBOL source code
IMS action programming in COBOL and basic assembly language (BAL) user guide, UP-9207	Describes IMS action programming
IMS system support functions user guide, UP-8364	Describes IMS utilities and recovery
ICAM utilities user guide, UP-8552	Describes the communication message control system (CMCS)

A glossary of terms relating to the language follows the appendixes.

Contents

PAGE STATUS SUMMARY

ACKNOWLEDGMENT

PREFACE

CONTENTS

1. INTRODUCTION

1.1.	SCOPE	1–1
1.2.	STRUCTURE OF COBOL LANGUAGE	1–1
1.2.1.	Module Overview	1–2
1.2.2.	Extensions to COBOL	1–3
1.3.	SYMBOLS, RULES, AND NOTATIONS USED IN THIS MANUAL	1–4
1.3.1.	Format	1–4
1.3.2.	Rules	1–4
1.3.3.	Elements	1–4

2. GENERAL SPECIFICATIONS

2.1.	COBOL CHARACTER SET	2–1
2.2.	SEPARATORS	2–4
2.3.	CHARACTER-STRINGS	2–5
2.3.1.	COBOL Words	2–5
2.3.1.1.	User-Defined Words	2–5
2.3.1.2.	System Names	2–6
2.3.1.3.	Reserved Words	2–6
2.3.2.	Literals	2–7
2.3.3.	PICTURE Character-String	2–11
2.3.4.	Comment-Entries	2–11

2.4. CLASSES OF DATA 2–11

2.5. STANDARD ALIGNMENT RULES 2–12

2.6. UNIQUENESS OF REFERENCE 2–12
2.6.1. Qualification 2–12
2.6.2. Subscripting 2–13
2.6.3. Indexing 2–13
2.6.4. Identifier 2–14
2.6.5. Condition-Name 2–14

2.7. REFERENCE FORMAT 2–15

3. IDENTIFICATION DIVISION

3.1. GENERAL 3–1

3.2. STRUCTURE 3–1

4. ENVIRONMENT DIVISION

4.1. GENERAL 4–1

4.2. STRUCTURE 4–1

4.3. CONFIGURATION SECTION 4–2
4.3.1. SOURCE-COMPUTER Paragraph 4–3
4.3.2. OBJECT-COMPUTER Paragraph 4–3
4.3.3. SPECIAL-NAMES Paragraph 4–4

4.4. INPUT-OUTPUT SECTION 4–15
4.4.1. FILE-CONTROL Paragraph 4–15
4.4.2. I-O-CONTROL Paragraph 4–23

5. DATA DIVISION

5.1. GENERAL 5–1

5.2. STRUCTURE 5–1
5.2.1. Heading and Sections 5–1
5.2.2. Entries 5–2
5.2.2.1. Level-Indicators 5–2
5.2.2.2. Level-Numbers 5–2
5.2.2.3. Special Level-Numbers 5–3

5.3. FILE SECTION 5–4
5.3.1. File Description 5–4
5.3.1.1. BLOCK CONTAINS Clause 5–6
5.3.1.2. RECORD CONTAINS Clause 5–10
5.3.1.3. LABEL RECORDS Clause 5–11
5.3.1.4. VALUE OF Clause 5–12

5.3.1.5. DATA RECORDS Clause 5–12
5.3.1.6. LINAGE Clause 5–13
5.3.1.7. CODE-SET Clause 5–17
5.3.2. Sort/Merge File Description 5–17
5.3.3. Data Description 5–18
5.3.3.1. Level-Number 5–21
5.3.3.2. Data-Name/FILLER Clause 5–21
5.3.3.3. REDEFINES Clause 5–22
5.3.3.4. PICTURE Clause 5–23
5.3.3.5. USAGE Clause 5–33
5.3.3.6. SIGN Clause 5–36
5.3.3.7. OCCURS Clause 5–38
5.3.3.8. SYNCHRONIZED Clause 5–41
5.3.3.9. JUSTIFIED Clause 5–44
5.3.3.10. BLANK WHEN ZERO Clause 5–45
5.3.3.11. VALUE Clause 5–45
5.3.3.12. RENAMES Clause 5–48

5.4. WORKING-STORAGE SECTION 5–49
5.4.1. 77-Level Description Entry 5–50
5.4.2. Record Description Entry 5–51

5.5. LINKAGE SECTION 5–51
5.5.1. 77-Level Description Entry 5–52
5.5.2. Record Description Entry 5–53

5.6. COMMUNICATION SECTION 5–53
5.6.1. Input Communication Description 5–53
5.6.2. Output Communication Description 5–57

6. PROCEDURE DIVISION

6.1. GENERAL 6–1
6.1.1. Declaratives 6–1
6.1.2. Procedures 6–1
6.1.3. Procedure Division Structure 6–2
6.1.3.1. Procedure Division Header 6–2
6.1.3.2. Procedure Division Body 6–3

6.2. CATEGORIES OF STATEMENTS 6–4
6.2.1. Imperative Statements 6–4
6.2.2. Conditional Statements 6–5
6.2.3. Compiler-Directing Statements 6–6

6.3. ARITHMETIC EXPRESSIONS 6–6
6.3.1. Arithmetic Operators 6–6
6.3.2. Formation and Evaluation Rules 6–7

6.4. CONDITIONAL EXPRESSIONS 6–8
6.4.1. Simple Conditions 6–8
6.4.1.1. Relation Condition 6–9
6.4.1.1.1. Comparison of Numeric Operands 6–10
6.4.1.1.2. Comparison of Nonnumeric Operands 6–11

6.4.1.1.3. Comparisons Involving Index-Names or Index Data Items 6-11
6.4.1.2. Class Condition 6-12
6.4.1.3. Condition-Name Condition 6-12
6.4.1.4. Switch-Status Condition 6-13
6.4.1.5. Sign Condition 6-13
6.4.2. Complex Conditions 6-13
6.4.2.1. Negated Simple Conditions 6-14
6.4.2.2. Combined and Negated Combined Conditions 6-15
6.4.2.3. Abbreviated Combined Relation Conditions 6-16
6.4.3. Condition Evaluation Rules 6-17

6.5. COMMON PHRASES AND GENERAL RULES FOR STATEMENT FORMATS 6-17
6.5.1. The ROUNDED Phrase 6-17
6.5.2. The SIZE ERROR Phrase 6-18
6.5.3. The CORRESPONDING Phrase 6-18
6.5.4. The Arithmetic Statements 6-19
6.5.5. Overlapping Operands 6-20
6.5.6. Multiple Results in Arithmetic Statements 6-20
6.5.7. WHEN-COMPILED Special Register 6-20

6.6. COBOL VERBS 6-20
6.6.1. ACCEPT Statement 6-20
6.6.2. ADD Statement 6-25
6.6.3. ALTER Statement 6-26
6.6.4. CALL Statement 6-27
6.6.5. CANCEL Statement 6-28
6.6.6. CLOSE Statement 6-29
6.6.7. COMPUTE Statement 6-33
6.6.8. COPY Statement 6-33
6.6.9. DELETE Statement 6-36
6.6.10. DISABLE Statement 6-37
6.6.11. DISPLAY Statement 6-38
6.6.12. DIVIDE Statement 6-41
6.6.13. ENABLE Statement 6-43
6.6.14. EXHIBIT Statement 6-44
6.6.15. EXIT Statement 6-44
6.6.16. GO TO Statement 6-45
6.6.17. IF Statement 6-46
6.6.18. INSPECT Statement 6-47
6.6.19. MERGE Statement 6-53
6.6.20. MOVE Statement 6-56
6.6.21. MULTIPLY Statement 6-59
6.6.22. ON Statement 6-60
6.6.23. OPEN Statement 6-60
6.6.24. PERFORM Statement 6-64
6.6.25. READ Statement 6-72
6.6.26. RECEIVE Statement 6-76
6.6.27. RELEASE Statement 6-77
6.6.28. RETURN Statement 6-78
6.6.29. REWRITE Statement 6-79
6.6.30. SEARCH Statement 6-81
6.6.31. SEND Statement 6-85
6.6.32. SET Statement 6-88
6.6.33. SORT Statement 6-90

6.6.34. START Statement 6-94
6.6.35. STOP Statement 6-96
6.6.36. STRING Statement 6-97
6.6.37. SUBTRACT Statement 6-99
6.6.38. TRACE Statement 6-100
6.6.39. TRANSFORM Statement 6-101
6.6.40. UNSTRING Statement 6-103
6.6.41. USE Statement 6-106
6.6.42. WRITE Statement 6-114
6.6.43. *DEBUG Statement 6-119

7. TABLE HANDLING SUMMARY

7.1. GENERAL 7-1

7.2. LANGUAGE CONCEPTS 7-1
7.2.1. Table Definition 7-1
7.2.2. References to Table Items 7-2
7.2.2.1. Subscripting 7-3
7.2.2.2. Indexing 7-3
7.2.2.3. Range Checking 7-4

7.3. DATA DIVISION CONSIDERATIONS 7-4

7.4. PROCEDURE DIVISION CONSIDERATIONS 7-4
7.4.1. Table-Handling Statements 7-4
7.4.2. Comparisons Involving Index-Name or Index Data Items 7-5
7.4.3. Overlapping Operands in a SET Statement 7-5

8. FILE PROCESSING SUMMARY

8.1. GENERAL 8-1

8.2. LANGUAGE CONCEPTS 8-1
8.2.1. File Organization and Access Methods 8-1
8.2.1.1. Sequential Organization 8-1
8.2.1.2. Relative Organization 8-1
8.2.1.3. Indexed Organization 8-2
8.2.1.4. SAM Organization 8-2
8.2.1.5. ISAM Organization 8-3
8.2.2. Current Record Pointer 8-3
8.2.3. I/O Status 8-3
8.2.4. The AT END Condition 8-5
8.2.5. The INVALID KEY Condition 8-5
8.2.6. LINAGE-COUNTER 8-5

8.3. SEQUENTIAL FILE PROCESSING 8-5
8.3.1. Level Characteristics 8-5
8.3.2. Clauses and Statements for Sequential File Processing 8-6
8.3.2.1. Environment Division 8-6
8.3.2.2. Data Division 8-6
8.3.2.3. Procedure Division 8-7
8.3.3. Printer-Destined Files 8-8
8.3.4. Multivolume Sequential Files 8-8

8.4. RELATIVE FILE PROCESSING 8-8
8.4.1. Level Characteristics 8-8
8.4.2. Clause and Statements for Relative File Processing 8-8
8.4.2.1. Environment Division 8-8
8.4.2.2. Data Division 8-9
8.4.2.3. Procedure Division 8-10

8.5. INDEXED FILE PROCESSING 8-11
8.5.1. Level Characteristics 8-11
8.5.2. Clauses and Statements for Indexed File Processing 8-11
8.5.2.1. Environment Division 8-11
8.5.2.2. Data Division 8-12
8.5.2.3. Procedure Division 8-12

8.6. SAM FILE PROCESSING 8-13
8.6.1. Environment Division 8-13
8.6.2. Data Division 8-14
8.6.3. Procedure Division 8-14
8.6.4. Multivolume SAM Files 8-15

8.7. ISAM FILE PROCESSING 8-15
8.7.1. Environment Division 8-15
8.7.2. Data Division 8-16
8.7.3. Procedure Division 8-16

9. SORT/MERGE SUMMARY

9.1. GENERAL 9-1

9.2. LANGUAGE CONCEPTS 9-2
9.2.1. Relationship with File Processing Facility 9-2
9.2.2. Sort Special Registers 9-2

9.3. ENVIRONMENT DIVISION CONSIDERATIONS 9-2
9.3.1. File Control Entry 9-3
9.3.2. I-O-CONTROL Paragraph 9-3

9.4. DATA DIVISION CONSIDERATIONS 9-3

9.5. PROCEDURE DIVISION CONSIDERATIONS 9-4
9.5.1. RELEASE Statement 9-4
9.5.2. RETURN Statement 9-4
9.5.3. SORT Statement 9-4
9.5.4. MERGE Statement 9-5

9.6. OBJECT TIME SUBROUTINE SORT/MERGE MAIN STORAGE REQUIREMENTS 9-5

10.1. GENERAL 10–1

10.2. ORGANIZATION 10–1
10.2.1. Fixed Portion 10–1
10.2.2. Independent Segments 10–2

10.3. SEGMENTATION CLASSIFICATION 10–2

10.4. SEGMENTATION CONTROL 10–3

10.5. STRUCTURE OF PROGRAM SEGMENTS 10–3
10.5.1. Segment Numbers 10–3
10.5.2. SEGMENT-LIMIT Clause 10–3
10.5.3. Object Module Naming Convention 10–3
10.5.4. Linkage Editor Control Statement Considerations 10–4

10.6. COBOL VERBS AFFECTED BY SEGMENTATION 10–4
10.6.1. ALTER Statement 10–4
10.6.2. PERFORM Statement 10–4
10.6.3. SORT Statement 10–5
10.6.4. MERGE Statement 10–5

11. LIBRARY SUMMARY

11.1. GENERAL 11–1

11.2. COPY STATEMENT 11–1

11.3. SOURCE PROGRAM CORRECTIONS DURING COMPILATION 11–2

12. DEBUGGING LANGUAGE SUMMARY

12.1. GENERAL 12–1

12.2. LANGUAGE CONCEPTS 12–1
12.2.1. DEBUG-ITEM Register 12–2
12.2.2. Compile-Time Switch 12–2
12.2.3. Object-Time Switch 12–2

12.3. ENVIRONMENT DIVISION CONSIDERATIONS 12–2
12.3.1. WITH DEBUGGING MODE Clause 12–2

12.4. PROCEDURE DIVISION CONSIDERATIONS 12–3
12.4.1. USE FOR DEBUGGING Statement 12–3
12.4.2. Debugging Lines 12–3
12.4.3. The Extended Debugging Facility 12–3
12.4.3.1. ON Statement 12–3
12.4.3.2. EXHIBIT Statement 12–4
12.4.3.3. TRACE Statement 12–4
12.4.3.4. The Debugging Packet (*DEBUG) 12–4

13. INTERPROGRAM COMMUNICATION SUMMARY

13.1. GENERAL 13–1
13.1.1. Transfer of Control 13–1
13.1.2. Access to Data Items 13–2
13.1.3. Level Characteristics 13–2

13.2. DATA DIVISION CONSIDERATIONS 13–2
13.2.1. Noncontiguous Linkage Storage 13–2
13.2.2. Linkage Records 13–3
13.2.3. Initial Values 13–3

13.3. PROCEDURE DIVISION CONSIDERATIONS 13–3
13.3.1. Procedure Division Header 13–3
13.3.2. CALL Statement 13–3
13.3.3. CANCEL Statement 13–3
13.3.4. EXIT PROGRAM Statement 13–4

13.4. OBJECT PROGRAM EXECUTION CONSIDERATIONS 13–4

14. COMMUNICATION SUMMARY

14.1. GENERAL 14–1

14.2. MESSAGE CONTROL SYSTEM 14–1

14.3. COBOL OBJECT PROGRAM 14–2

14.4. RELATIONSHIP OF COBOL PROGRAM TO MCS AND COMMUNICATION DEVICES 14–2
14.4.1. Invoking the COBOL Object Program 14–3
14.4.1.1. Scheduled Initiation 14–3
14.4.1.2. MCS Invocation 14–4
14.4.1.3. Determining the Method of Invocation 14–4

14.5. CONCEPT OF MESSAGES AND MESSAGE SEGMENTS 14–4

14.6. CONCEPT OF QUEUES 14–5
14.6.1. Enabling and Disabling Logical Connectives 14–5
14.6.2. Enqueueing and Dequeueing Methods 14–5
14.6.3. Queue Hierarchy 14–5

14.7. MESSAGE CONTROL SYSTEM GENERATION 14–7

APPENDIXES

A. COMPILER OPTIONS

A.1. GENERAL A–1

A.2. COMPILER OPTION SPECIFICATION A–1

A.3. COMPILER OPTION SPECIFICATION CONSISTENCY CHECK A–4

B. COMPILER LISTINGS

B.1. COMPILATION SUMMARY LISTING B-1

B.2. DIAGNOSTIC LISTING B-1

B.3. SOURCE LISTING B-1

B.4. OBJECT CODE LISTING B-2

B.5. LOCATOR/MAP/CROSS-REFERENCE LISTINGS B-2

B.6. ALPHABETICALLY ORDERED CROSS-REFERENCE LISTING B-3

B.7. OBJECT CODE MAP LISTING B-3

C. COMPILER DIAGNOSTICS

C.1. GENERAL C-1

C.2. DIAGNOSTIC LISTING C-1

D. FEDERAL INFORMATION PROCESSING STANDARD FLAGGING FACILITY

D.1. FIPS PUB 21-1 COBOL LEVELS D-1

D.2. FLAGGING OPTIONS D-2

E. OBJECT PROGRAM PROCESSING CONSIDERATIONS

E.1. INTERMEDIATE RESULTS IN ARITHMETIC OPERATIONS E-1
E.1.1. Floating-Point Operands E-2
E.1.2. ADD and SUBTRACT Statements E-2
E.1.3. MULTIPLY Statement E-2
E.1.4. DIVIDE Statement E-2

E.2. EXPRESSIONS E-3

F. NON-ENGLISH LANGUAGE FEATURE

F.1. FUNCTION F-1

F.2. LANGUAGE CONCEPTS F-1
F.2.1. Control Division F-1
F.2.2. CLASS-NAME Clause F-2
F.2.3. Extended Class Condition F-2

F.3. COMPOSITE LANGUAGE FORMAT F-2

F.4. CONTROL DIVISION F–3

F.5. ENVIRONMENT DIVISION F–4
F.5.1. CLASS-NAME Clause F–4

F.6. PROCEDURE DIVISION F–6
F.6.1. Extended Class Condition F–6

F.7. NON-ENGLISH TEXT UTILITY PROGRAM F–6

G. IMS ACTION PROGRAMS

G.1. GENERAL G–1

G.2. ACTION PROGRAMS G–1

G.3. COMPILER PARAMETER SPECIFICATION AND CONFIGURATION SPECIFICATIONS G–4

G.4. REENTRANT ACTION PROGRAM WORK AREA USAGE G–4

H. JOB CONTROL STREAM REQUIREMENTS

H.1. GENERAL H–1

H.2. PROCEDURE CALL STATEMENT H–1

H.3. COMPILER STATUS INDICATORS H–15

H.4. DATA DEFINITION (DD) JOB CONTROL STATEMENT KEYWORD PARAMETERS H–16

I. RESERVED WORDS

J. CHARACTER SETS

K. PICTURE CLAUSE

K.1. GENERAL K–1

K.2. USE OF THE PICTURE CLAUSE AND ITS SYMBOLS K–1

K.3. DESCRIPTIONS AND EXAMPLES OF PICTURE CLAUSE SYMBOLS K–2

GLOSSARY

INDEX

USER COMMENT SHEET

FIGURES

3–1. Sample Identification Division Entries 3–2

4–1. Sample Environment Division Entries 4–2
4–4. Compiler Default Value of the RESERVE Clause 4–20

5–1. Sample File Section Entries 5–6
5–2. Logical Page Format for Format 1 LINAGE Clause 5–16
5–3. Sample Sort File Description Entries 5–18
5–4. Sample Working-Storage Section Entries 5–50
5–5. Sample Linkage Section Entries 5–52
5–6. Sample Communication Section Entries 5–62

6–1. Sample Procedure Division 6–4
6–2. Flowchart for the VARYING Phrase Having One Condition 6–67
6–3. Flowchart for the VARYING Phrase Having Two Conditions 6–69
6–4. Flowchart for the VARYING Phrase Having Three Conditions 6–70
6–5. Flowchart for a Format 1 Search Operation Containing Two WHEN Phrases 6–83

14–1. COBOL Communication Environment 14–3
14–2. Hierarchy of Queues 14–6

G–1. IMS Work Area Usage G–5

TABLES

1–1. COBOL Language Processing Levels 1–1

4–1. Status Key Values for Workstations 4–8
4–2. Effects of CONNECT-FREE Reporting 4–9
4–3. Effects of FUNCTION-KEYS Input 4–9

5–1. File Description Entry Clauses 5–5
5–2. Block Size Calculations for Tape, Card Reader, Card Punch, and Printer Files 5–8
5–3. Block Size Calculations for Mass Storage SAM and ISAM Files 5–9
5–4. Block Size Calculations for Mass Storage Sequential, Relative, and Indexed Files 5–10
5–5. Data Description Entry Clauses 5–20
5–6. Class and Category of Elementary and Group Data Items 5–24
5–7. Type of Editing Permissible for Each Data Category 5–29
5–8. Results Produced by Editing Sign Control Symbols 5–30
5–9. PICTURE Character Precedence Chart 5–32a
5–10. Alignment Boundaries for Various Type Elementary Items 5–41
5–11. Communication Status Key Condition 5–60
5–12. Error Key Codes 5–61

6–1. Permissible Symbol Combinations in Arithmetic Expressions 6–8
6–2. Permissible Comparisons for Relation Conditions 6–9
6–3. Logical Operators and the Resultant Values 6–14
6–4. Combinations of Conditions, Logical Operators, and Parentheses 6–15
6–5. Relationship of Categories of Files and the Options of the CLOSE Statement 6–30
6–6. Permissible MOVE Statement Data Transfers 6–58
6–7. Permissible Input/Output Statements for Each OPEN Mode 6–62
6–8. Valid Uses of the Format 1 SET Statement 6–90
6–9. Combination of FROM and TO Options in a TRANSFORM Statement 6–102

8–1. Status Key Values and Meanings 8–4

12–1. Debug Conditions and Contents of DEBUG-item 12–5

A–1. Options of the PARAM Statement A–2
A–2. Parameter Consistency Checks A–5

D–1. Federal Standard COBOL Levels D–1

G–1. IMS Configuration G–4

J–1. Correspondence between EBCDIC, ASCII-8, and Punched Card Codes J–2

1. Introduction

1.1. SCOPE

This manual describes the 1974 American National Standard COBOL compiler operating in both the Series 90 environment (90/25, 90/30, 90/30 B, and 90/40) and the System 80 environment. Keeping this in mind, two of the Sperry Univac extensions that enhance the capabilities of COBOL apply only to the Series 90 environment. They are the sequential access method (SAM) and the indexed sequential access method (ISAM). When these extensions appear throughout the manual, they are footnoted to act as a reminder. All other extensions are supported in both environments, as well as all other language features.

1.2. STRUCTURE OF COBOL LANGUAGE

COBOL is structured into a nucleus and a number of functional processing modules. The nucleus contains language elements for internal processing. The functional processing modules are: table handling, sequential I-O, relative I-O, indexed I-O, sort/merge, segmentation, library, debug, interprogram communications, and communication.

Each module contains either two or three levels. Those with three levels contain a null set at their lowest level, a low processing level (level 1), and a high processing level (level 2). In all cases, lower levels are subsets of higher levels within the same module. Table 1-1 lists all modules and levels implemented on the operating system.

Table 1—1. COBOL Language Processing Levels

Module	Level
Nucleus	2
Table handling	2
Sequential I-O	2
Relative I-O	2
Indexed I-O	2
Sort/merge	2
Segmentation	2
Library	2
Debug	2
Interprogram communication	2
Communication	2

1.2.1. Module Overview

- Nucleus

The nucleus contains the language elements for internal processing. This module is divided into two levels. The Level 1 elements perform basic internal operations, i.e., elementary options of the various clauses and verbs. Level 2 provides more extensive and sophisticated internal processing capabilities.

- Table handling

The table handling module contains the language elements necessary for:

1. the definition of tables;

2. the identification, manipulation, and use of indexes; and

3. reference to the items within tables.

This module is divided into two levels. Level 1 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. Level 2 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.

- Sequential I-O

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. Level 1 contains the basic facilities for the definition and access of sequential files and for the specification of checkpoints. Level 2 contains more complete facilities for defining and accessing these files.

- Relative I-O

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. Level 1 provides basic facilities. Level 2 provides more complete facilities, including the capability of accessing the file both randomly and sequentially in the same COBOL program.

- Indexed I-O

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. Level 1 provides basic facilities. Level 2 provides more complete facilities, including alternate keys, and the capability of accessing the file both randomly and sequentially in the same COBOL program.

- Sort/Merge

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. Level 1 contains facilities to sort a single file only; Level 2 provides extended sorting capabilities, including a merge facility.

- Segmentation

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. Level 1 provides for section segment-numbers and fixed segment limits; Level 2 adds the capability for varying the segment limit.

- Library

  The library module consists of a null set and two processing levels. It provides for the inclusion into a program of predefined COBOL text. Level 1 contains the basic COPY verb; Level 2 adds the REPLACING phrase.

- Debug

  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. Level 1 provides a basic debugging capability, including the ability to specify selective or full paragraph monitoring. Level 2 provides the full COBOL debugging capability.

- Interprogram Communication

  The interprogram 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. Level 1 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. Level 2 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 main storage for the called program. The high level also provides the capability for the release of main storage areas occupied by called programs.

- Communication

  The communication module provides the ability to access, process, and create messages or portions of messages, and to communicate through a COBOL message control system with local and remote communication devices. This module consists of a null set and two processing levels. Level 1 provides basic facilities to send or receive complete messages. Level 2 provides a more sophisticated facility including the capability to send or receive segments of a message.

1.2.2. Extensions to COBOL

Sperry Univac has provided a number of extensions to the standard COBOL language. These extensions are indicated in the manual by dashed-line boxes. The extended language elements are as follows:

Apostrophe as quotation mark
USAGE COMPUTATIONAL-n
DISPLAY floating-point data item
Floating point literal
Hexadecimal literal
CALL USING argument
IF THEN statement
TRANSFORM statement
ISAM* file processing facility
Extended debugging facility
ON statement
WHEN-COMPILED special register
Non-English language feature
APPLY clauses
SAM* file processing facility
Extended RERUN option
Standard user tape labels
Sort special registers
Assign clause in SPECIAL-NAMES
Format 4 of ACCEPT statement
Format 2 of DISPLAY statement

**Applies only to 90/25, 90/30, 90/30 B and 90/40 systems*

1.3. SYMBOLS, RULES, AND NOTATIONS USED IN THIS MANUAL

1.3.1. Format

A format is the specific arrangement of the elements of a clause or a statement. A clause or a statement consists of elements as defined in 1.3.3. Throughout this document, a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the format is separated into numbered formats. Clauses must be written in the sequence given except where specifically stated in the rules associated with a given format. (Clauses that are optional must appear in the sequence shown if they are used.) Applications, requirements, or restrictions are shown as rules. Throughout this document, specifications unique to Level 2 of a module are enclosed in boxes, and Sperry Univac extensions to the COBOL language are enclosed in dashed-line boxes (1.2.2). Default values throughout this document are indicated by shading the values.

1.3.2. Rules

Rules are used to define or clarify:

1. the syntax or arrangement of words or elements in a larger structure, such as a clause or statement; or

2. the meaning or relationship of meanings of an element or set of elements in a statement and the effect of the statement on compilation or execution.

1.3.3. Elements

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

- 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 that 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.

- 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. (See 5.2.2.2.)

- 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. 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).

- 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.

- Format Punctuation

The punctuation characters comma and semicolon are shown in some 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.

- Use of Certain Special Characters in Formats

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

2. General Specifications

2.1. COBOL CHARACTER SET

The most basic and indivisible unit of the language is the character. The set of 51 characters used to form COBOL character-strings and separators includes the letters of the alphabet, digits, and special characters. For nonnumeric literals, comment-entries, and comment lines, the character set is expanded to include the entire computer character set.

The COBOL character set consists of the following characters:

0,1,...,9	
A,B,...,Z	
	Blank or space (written on coding form as △ or a blank space)
.	Period (decimal point)
<	Less than
(	Left parenthesis
+	Plus sign
$	Currency sign
*	Asterisk
)	Right parenthesis
;	Semicolon
-	Minus sign or hyphen
,	Comma (decimal point)
>	Greater than
=	Equal sign

" or '	Quotation mark
/	Slash (stroke, virgule)

The collation sequence for these characters is given in Appendix J.

These characters may be used as follows:

- Characters Used for Words

  A COBOL word is a sequence of not more than 30 of the following characters:

  0,1,...,9

  A,B,...,Z

  -(hyphen)

  A word may neither begin nor end with a hyphen, or contain a space.

- Characters Used for Punctuation

  COBOL punctuation characters are:

(	Left parenthesis
)	Right parenthesis
	Blank or space (written on coding form as △ or a blank space)
.	Period
,	Comma
;	Semicolon
" or '	Quotation mark
=	Equal sign

- Characters Used in Relational Expressions

  The COBOL characters used to represent relational operators are:

=	Equals
>	Greater than
<	Less than

- Characters Used in Arithmetic Expressions

The characters used in arithmetic expressions are:

+	Plus sign (addition)
-	Minus sign (subtraction)
*	Asterisk (multiplication)
/	Slash (division)
**	Two asterisks (exponentiation)

- Characters Used in Editing

The characters used in editing are:

B	Blank or space
0	Zero
+	Plus sign
-	Minus sign
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
/	Stroke (virgule, slash)

2.2. SEPARATORS

A separator is a string of one or more punctuation characters. The separators and the rules for their formation are as follows:

1. Blank or space

   a. Anywhere a space is used as a separator, more than one space may be used.

   b. The space may precede all separators except:

   - As specified by reference format rules (2.7)

   - 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 space may 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.

2. Comma, semicolon, and period immediately followed by a space.

   These separators may appear in a COBOL source program only where explicitly permitted by the general formats, by format punctuation rules (1.3.3), by statement and sentence structure definitions (6.2), or by reference format rules (2.7).

3. Right and left parentheses

   Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, indexes, arithmetic expressions, or conditions.

4. Quotation mark

   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 (2.7).

5. Pseudo-text delimiters

   The delimiter consists of two contiguous equal signs. 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 and may not be continued across two lines.

Any punctuation character that 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 that comprise the contents of nonnumeric literals, comment-entries, or comment lines.

2.3. CHARACTER-STRINGS

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

2.3.1. COBOL Words

A COBOL word is a character-string of not more than 30 characters that 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.

2.3.1.1. 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 through Z, 0 through 9, and -, except that the hyphen may not appear as the first or last character.

There are 15 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
section-name
segment-number
text-name

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.

The user-defined words condition-name, mnemonic-name, paragraph-name, and section-name are defined in the following paragraphs. The definition for all other user-defined words may be found in the glossary.

- Condition-name

  A condition-name 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 SYSSWCH[-n].

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

- Mnemonic-name

  A mnemonic-name assigns a user-defined word to an implementor-name. These associations are established in the SPECIAL-NAMES paragraph of the environment division.

- Paragraph-name

  A paragraph-name 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 or characters.

- Section-name

  A section-name 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 or characters.

2.3.1.2. System Names

A system name is a COBOL word used to communicate with the environment. There are two types of system names – computer-name and implementor-name. These names are defined in the format or rules of the language element in which they appear.

2.3.1.3. Reserved Words

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

There are six types of reserved words:

- Key Words

  A key word is a word that 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 WRITE

  2. Required words that appear in statement and entry formats

  3. Words with a specific functional meaning, such as NEGATIVE and SECTION

- Optional Words

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

- 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

- Special Registers

  Special registers are compiler-generated storage areas used to store information produced when using specific COBOL features. These special registers are named with reserved words as follows: LINAGE-COUNTER (5.3.1.6), DEBUG-ITEM (12.2.1), WHEN-COMPILED (6.5.7), SORT-FILE-SIZE (9.2.2), and SORT-MODE-SIZE (9.2.2).

- Figurative Constants

  Certain reserved words are used to name and reference specific constant values as explained in 2.2.

- Special-Character Words

  The arithmetic operators and relation characters listed in 2.1 are reserved words.

2.3.2. Literals

A literal is a character-string whose value is implied by:

1. an ordered set of characters of which the literal is composed; or

2. specification of a reserved word that references a figurative constant.

Literals are nonnumeric, numeric, or hexadecimal:

- Nonnumeric Literals

  A nonnumeric literal is a character-string delimited on both ends by quotation marks and consisting of any allowable character in the EBCDIC character set. The compiler allows for nonnumeric literals of 1 through 132 characters in length. The value of a nonnumeric literal in the object program is the string of characters itself, except:

  - the delimiting quotation marks are excluded; and

  - each embedded pair of contiguous quotation marks represents a single quotation mark character.

However, the double quote character (") appearing within a nonnumeric literal bounded by single quotes is treated as part of the value of the nonnumeric literal rather than a separator.

Example:

Coding	Result
'THIS IS "EDITED" OUTPUT'	THIS IS "EDITED" OUTPUT

The single quote character (') appearing within a nonnumeric literal bounded by the double quote characters (") is also treated as part of the nonnumeric literal.

Example:

Coding	Result
"THIS IS 'EDITED' OUTPUT"	THIS IS 'EDITED' OUTPUT

To represent a single quote character within a nonnumeric literal bounded by single quotes, two contiguous single quotes must be used.

Example:

Coding	Result
'THIS IS "EDITED" OUTPUT'	THIS IS 'EDITED' OUTPUT

To represent a double quote character within a nonnumeric literal bounded by double quotes, two contiguous double quote characters must be used.

Example:

Coding	Result
"THIS IS ""EDITED"" OUTPUT"	THIS IS "EDITED" OUTPUT

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals are category alphanumeric. (See 5.3.3.4, the PICTURE clause.)

- Numeric Literals

There are two types of numeric literals – fixed-point and floating-point.

1. Fixed-Point

   A fixed-point literal is a character-string whose characters are selected from the digits 0 through 9, the plus sign, the minus sign, and the decimal point. A fixed-point literal consists of 1 through 18 digits in length. The rules for the formation of fixed-point literals are as follows:

   a. A literal must contain at least one digit.

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

c. A literal must not contain more than one decimal point. The decimal point 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.

d. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal is category numeric. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

2. Floating-Point

A floating-point literal is a numeric literal whose potential range of value is too great for fixed-point representation.

A floating-point literal must have the following format:

```
[±] mantissa E [±] exponent
```

where:

±
The two plus or minus signs are optional.

mantissa
Consists of 1 to 16 digits with a required decimal point; the decimal point may appear in any position.

exponent
Consists of the symbol E, followed by an optional sign, followed by one or two digits. (A zero exponent may be written as 0 or 00.)

The literal must contain no spaces. The exponent must appear immediately to the right of the mantissa.

The signs are the only optional characters in the format. An unsigned mantissa or exponent is assumed to be positive.

The value of the literal is the product of the mantissa and 10 raised to the power given by the exponent.

Example:

$+ 1.5E - 2 = 1.5 \times 10^{-2}$

The magnitude of the number represented by a floating-point literal must not exceed $.72 \times 10^{76}$. The smallest nonzero value that can be represented by a floating-point literal is $\pm 5.4 \times 10^{-79}$.

- Hexadecimal Literals

A hexadecimal literal is a string of hexadecimal digits bounded by single or double quotation marks and immediately preceded by an equal sign.

Examples:

```
="Ø23C"
='Ø23C'
```

The string may include any hexadecimal digits (0 through 9 and A through F). The length of a hexadecimal literal ranges from 1 through 30 hexadecimal digits. If the literal consists of an odd number of hexadecimal digits, a leading hexadecimal zero is provided by the compiler to make the literal an even number of digits.

A hexadecimal literal may be used anywhere a nonnumeric literal is permitted. In this manual, a hexadecimal literal is considered a nonnumeric literal.

A hexadecimal literal may be broken in such a way that part of it appears on a continuation line. Continuation of a hexadecimal literal follows the rules for continuation of a COBOL word.

Example:

```
BAKER.
    MOVE  ="13A
-   4C8" TO FIELD.
```

- Figurative Constant Values

Figurative constant values are generated by the compiler and referenced through the use of reserved words. 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 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 " (not 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 whenever a literal appears in a format, except that whenever the literal is restricted to numeric characters, 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 4.3.2, the OBJECT-COMPUTER paragraph, and 4.3.3, the SPECIAL-NAMES paragraph.)

Each reserved word 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.

2.3.3. PICTURE Character-String

A PICTURE character-string consists of certain combinations of characters in the COBOL character set used as symbols. See 5.3.3.4, the PICTURE clause, for the discussion of the PICTURE character-string and for the rules that govern its use.

Any punctuation character that 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.

2.3.4. 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.

2.4. CLASSES OF DATA

In COBOL, data is classified into three classes: numeric, alphabetic, and alphanumeric. The three classes are further divided into five categories: numeric, alphabetic, numeric edited, alphanumeric edited, and alphanumeric (without editing).

Every elementary item except the index data item belongs to one of the classes and to one of the categories. The class of a group item is treated as alphanumeric regardless of the class of elementary items subordinate to the group item. For further information on classes of data, refer to the PICTURE clause.

2.5. 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 rule 1a.

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 5.3.3.9, the JUSTIFIED clause.

2.6. UNIQUENESS OF REFERENCE

2.6.1. Qualification

Every user-defined name that specifies an element in a COBOL source program must be unique, either by having no other name with the identical spelling and hyphenation, or by having the name 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 the 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, or a paragraph-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:

$$\left\{\begin{matrix}\text{data-name-1}\\ \text{condition-name}\end{matrix}\right\}\left[\left\{\begin{matrix}\underline{\text{OF}}\\ \underline{\text{IN}}\end{matrix}\right\}\text{ data-name-2}\right]\ldots$$

Format 2:

$$\text{paragraph-name}\left[\left\{\begin{matrix}\underline{\text{OF}}\\ \underline{\text{IN}}\end{matrix}\right\}\text{ section-name}\right]$$

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 referenced 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.

7. Qualified data-names may have any number of qualifiers up to a limit of five. However, for compatibility with existing SPERRY UNIVAC compilers, this compiler will accept the use of up to 50 qualifiers.

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

2.6.2. Subscripting

Subscripts are used to refer to individual elements within a list or table of like elements that have not been assigned individual data-names. (Subscripting is described in detail in Section 7.)

2.6.3. Indexing

Indexing is a method of referring to elements within a table by using an index for a given level of a table. The index is assigned by specifying the INDEXED BY phrase of the OCCURS clause. (Refer to Section 7 for a detailed description of indexing.)

2.6.4. 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 indexes necessary to ensure uniqueness.

Format 1:

```
data-name-1 ⎡⎧OF⎫ data-name-2⎤... ⎡(subscript-1 ⎡,subscript-2 [,subscript-3]⎤)⎤
            ⎣⎩IN⎭            ⎦    ⎣             ⎣                           ⎦ ⎦
```

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. Such data items are called index data items.

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

2.6.5. 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 (2.6.4) except that data-name-1 is replaced by condition-name-1.

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

2.7. REFERENCE FORMAT

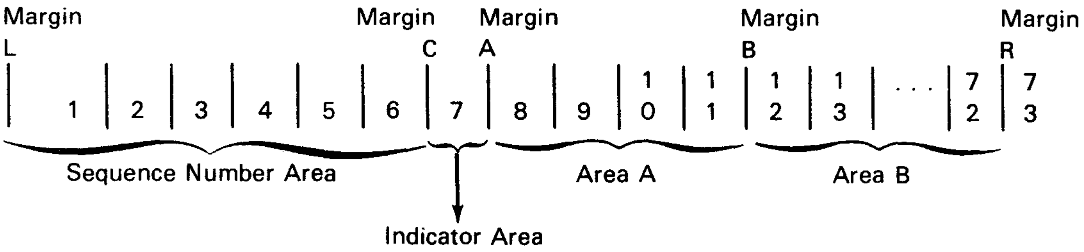
The reference format, which provides a standard method for describing COBOL source programs, is described in terms of character positions in a line on an input/output medium. A line consists of 72 character positions for any input media. The COBOL compiler accepts source programs written in reference format and produces an output listing of the source program input in reference format. Source programs written in reference format in an 80-character card image containing user identification information in character positions 73 through 80 are also accepted. The identification information has no significance except that it is printed as received on the source listing.

The rules for spacing given in this 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, the environment division, the data division, then the procedure division. Each division must be written according to the rules for the reference format.

- Format Representation

The reference format for a line is represented as follows:



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

Margin C is between the sixth and seventh character positions of a line.

Margin A is between the seventh and eighth character positions of a line.

Margin B is between the eleventh and twelfth character positions of a line.

Margin R is immediately to the right of the seventy-second 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 seventh character position of a line.

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

Area B occupies character positions 12 through 72. It begins immediately to the right of margin B and terminates immediately to the left of margin R.

- Sequence Numbers

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

- Continuation of Lines

Whenever a sentence, entry, phrase, or clause requires more than one line, it may be continued by starting subsequent lines in area B. These subsequent lines are called the continuation lines. 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.

An asterisk in the continuation indicator area of the line indicates a comment line. (See Comment Lines.)

- Blank Line

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 that has a hyphen in column 7.

- Division and Section Headers

The division header and 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.

- 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 the discussion of Comment Lines. 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 the discussion of Continuation of Lines.

- 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, 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 that begin with a level indicator and those that begin with a level-number.

1. Level Indicators

   The level indicators are FD, SD, and 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.

2. Level Numbers

   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.

All other level numbers, 02 through 49, and special level numbers 66 and 88 may begin in area A or B.

Successive data description entries may have the same format as the first or may be indented according to level-number. Indentation does not affect the magnitude of a level-number; its primary use is to improve readability.

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

- Declaratives

The keyword DECLARATIVES and the key words END DECLARATIVES that precede and follow, respectively, the declaratives portion of the procedure division must each appear on a line by itself. Each must begin in area A and be followed by a period.

- 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 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 are 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.

3. Identification Division

3.1. GENERAL

The identification 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 indicated in the format. The identification division must be included in every COBOL source program.

3.2. STRUCTURE

Paragraph headers identify the type of information contained in the paragraph. The PROGRAM-ID paragraph must be present. The other paragraphs are optional and may be specified at the user's discretion.

Format:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
[AUTHOR. [comment-entry.] ...]
[INSTALLATION. [comment-entry.] ...]
[DATE-WRITTEN. [comment-entry.] ...]
[DATE-COMPILED. [comment-entry.] ...]
[SECURITY. [comment-entry.] ...]
```

Rules:

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

2. The division header must be followed by the PROGRAM-ID paragraph.

3. The program-name may contain 1 to 30 characters. It must consist of only letters and digits and must begin with an alphabetic character.

4. The system uses only the first six characters of program-name as the identifying name of the object program. Therefore, these characters should be unique for every name in a particular program library.

5. If program-name is not supplied or not accepted because of an error, the compiler automatically supplies the program-name COB.

6. The optional paragraphs that follow the PROGRAM-ID paragraph must be in the same order as given in the format.

7. The comment-entry may be any combination of the characters from the computer character set and must start in area B as designated in the reference format. Continuation of the comment-entry by using the hyphen in the indicator area is not permitted; however, the comment-entry may be contained on one or more lines.

8. The DATE-COMPILED paragraph name 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.

Example:

An example of an identification division is given in Figure 3-1.

```
Seq
No     A   B    Text
1      8   12
------------------------------------------------------
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. PAY44.
001030 AUTHOR. JOHN SMITH.
001040 INSTALLATION. ABC COMPANY.
001050 DATE-WRITTEN. NOVEMBER 15, 1978.
001060 DATE-COMPILED. TODAY.
001070 SECURITY.  PAYROLL DEPT ONLY.
001080*THIS PROGRAM ADDS COMMISSIONS TO SALARY GIVING
001090*    TOTAL MONTHLY EARNINGS OF SALES PERSONNEL.
```

Figure 3—1. Sample Identification Division Entries

4. Environment Division

4.1. GENERAL

In the environment division of a COBOL source program, a relationship is established between the physical requirements of the computing system on which the source program is compiled and the characteristics of the computing system on which the object program is to run. In addition, this division assigns input-output devices to the files used by the object program and indicates the techniques to be used in processing the files. This division must be included in every COBOL source program.

4.2. STRUCTURE

The environment division consists of two sections, each of which has a fixed name. They are:

CONFIGURATION SECTION.

INPUT-OUTPUT SECTION.

The configuration section identifies the source computer and object computer and relates system-oriented device names to user-defined mnemonic names. The input-output section deals with the information needed to control transmission and handling of data between external media and the object program.

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.] ]
```

Rules:

1. The environment division begins with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

2. The sections and paragraphs of the environment division must be written in the order given in the format.

Example:

An example of the environment division is given in Figure 4-1.

```
 Seq.
 No.   A   B    Text
1      8   12
ØØ1Ø1Ø ENVIRONMENT DIVISION.
ØØ1Ø2Ø CONFIGURATION SECTION.
ØØ1Ø3Ø SOURCE-COMPUTER. UNIVAC-OS3.
ØØ1Ø4Ø OBJECT-COMPUTER. UNIVAC-OS3.
ØØ1Ø5Ø SPECIAL-NAMES.
ØØ1Ø6Ø     SYSCONSOLE IS TYPEIT.
ØØ1Ø7Ø INPUT-OUTPUT SECTION.
ØØ1Ø8Ø FILE-CONTROL.
ØØ1Ø9Ø     SELECT INPUT1 ASSIGN TO TAPE-INPUT-F.
ØØ11ØØ     SELECT LIST ASSIGN TO PRINTER-P115-VC.
ØØ111Ø     SELECT CDS ASSIGN TO DISK-C129-V.
ØØ112Ø I-O-CONTROL.
ØØ113Ø     RERUN ON DISK-CKPT20-1 EVERY 5000 RECORDS OF INPUT1.
```

Figure 4—1. Sample Environment Division Entries

4.3. CONFIGURATION SECTION

The configuration section specifies the operating system on which the program is to be compiled and run and relates implementor-names to user-names.

Format:

```
CONFIGURATION SECTION.
SOURCE-COMPUTER. entry.
OBJECT-COMPUTER. entry.
[SPECIAL-NAMES. entry.]
```

4.3.1. SOURCE-COMPUTER Paragraph

Function:

The SOURCE-COMPUTER paragraph identifies the operating system that will compile the source program and indicates whether the debugging sections and debugging lines are to be compiled.

Format:

```
                 ⎧UNISYS-OS3⎫
SOURCE-COMPUTER. ⎨SPERRY-OS3⎬[WITH DEBUGGING MODE].
                 ⎩UNIVAC-OS3⎭
```

Rules:

1. UNISYS-OS3, SPERRY-OS3, or UNIVAC-OS3 specifies that the source program is to be compiled under the OS/3 operating system.
2. The operating system name specified in the SOURCE-COMPUTER paragraph is for documentation only.
3. If the WITH DEBUGGING MODE clause is specified, all USE FOR DEBUGGING statements and all debugging lines are compiled (12.4).
4. If the WITH DEBUGGING MODE clause is not specified, all USE FOR DEBUGGING statements with associated debugging sections and all debugging lines are compiled as if they were comment lines.
5. The WITH DEBUGGING MODE clause has no effect on debugging packets (12.4.3.4).

4.3.2. OBJECT-COMPUTER Paragraph

Function:

The OBJECT-COMPUTER paragraph describes the operating system on which the object program is to be run.

Format:

```
                 ⎧UNISYS-OS3⎫
OBJECT-COMPUTER. ⎨SPERRY-OS3⎬
                 ⎩UNIVAC-OS3⎭
   ⎡,MEMORY SIZE integer ⎧CHARACTERS⎫⎤
   ⎢                     ⎨MODULES   ⎬⎥
   ⎣                     ⎩WORDS     ⎭⎦
   [,PROGRAM COLLATING SEQUENCE IS alphabet-name]
   [,SEGMENT-LIMIT IS segment-number] .
```

Rules:

1. UNISYS-OS3, SPERRY-OS3, or UNIVAC-OS3 specifies that the object program is to be executed under the OS/3 operating system.

2. The MEMORY SIZE clause is for documentation only. A word is 4 characters; a module is 4096 characters.

3. If the PROGRAM COLLATING SEQUENCE clause is specified, the collating sequence associated with alphabet-name (see 4.3.3, rule 8) is used to determine the truth value of any nonnumeric comparisons:

   a. Explicitly specified in relation conditions (See 6.4.1.1.)

   b. Explicitly specified in condition-name conditions (See 6.4.1.3.)

4. If the PROGRAM COLLATING SEQUENCE clause is not specified, the EBCDIC collating sequence is used.

5. 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.

6. 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 6.6.19, the MERGE statement and 6.6.33, the SORT statement.)

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

8. The segment-number in the SEGMENT-LIMIT clause must be an integer ranging in value from 1 through 49. (See 6.1.3 and Section 10.)

9. 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.

10. Those segments having segment-numbers from the segment-limit through 49 are considered as overlayable fixed segments.

11. 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.

4.3.3. SPECIAL-NAMES Paragraph

Function:

The SPECIAL-NAMES paragraph relates implementor-names to user-supplied mnemonic-names and alphabet-names to character sets or collating sequences.

Format:

```
SPECIAL-NAMES.
     [SYSIN IS mnemonic-name-1]
     [,SYSCONSOLE IS mnemonic-name-2]
     [,SYSLST  IS mnemonic-name-3]
     [,SYSLOG IS mnemonic-name-4]
     [,SYSCHAN-n IS mnemonic-name-5]
     [,SYSCOM IS mnemonic-name-6]
     [,SYSSCOPE IS mnemonic-name-7]
     ⎡,⎧SYSTERMINAL⎫IS mnemonic-name-8⎤
     ⎣ ⎩SYSOUT     ⎭                  ⎦
```

```
[,{SYSFORMAT}IS mnemonic-name-9                   ]
[ {SYSWORK  }                                     ]
[              ┌ ASSIGN TO lfdname              ┐ ]
[              ┆ [CONTROL AREA IS data-name   ] ┆ ]
[              ┆ [    [WITH FUNCTION-KEYS]    ] ┆ ]
[              ┆ [    [WITH CONNECT-FREE]     ] ┆ ]
[              └--------------------------------┘ ]
[{,SYSSWCH[-n]     }                                         ]
[{ SYSTEM-SHUTDOWN }                                         ]
[    {IS mnemonic-name,ON STATUS IS condition-name        }  ]
[    {      ,OFF STATUS IS condition-name                 }  ]
[    {IS mnemonic-name,OFF STATUS IS condition-name       }  ]
[    {      ,ON STATUS IS condition-name                  }  ] ...
[    {ON STATUS IS condition-name,OFF STATUS IS           }  ]
[    {    condition-name                                  }  ]
[    {OFF STATUS IS;condition-name,ON STATUS IS           }  ]
[    {    condition-name                                  }  ]
[ ,alphabet-name IS { STANDARD-1                                                 } ]
[                   { NATIVE                                                     } ]
[                   { STANDARD-Ø                                                 } ]
[                   { ┌─────────────────────────────────────────────────────────┐ } ]
[                   { │ 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].
```

Rules:

1. The SPECIAL-NAMES paragraph is optional.

2. A comma may be used to separate each clause, and a period must follow the last clause.

3. Mnemonic-names associated with SYSIN, SYSCONSOLE, SYSCOM, SYSTERMINAL, SYSWORK, SYSFORMAT, SYSSCOPE, SYSSWCH, and SYSTEM-SHUTDOWN may be used in the ACCEPT statement. Mnemonic-names associated with SYSLST, SYSOUT, SYSLOG, SYSCONSOLE, SYSCOM, SYSTERMINAL, SYSWORK, SYSFORMAT, SYSSCOPE, and SYSSWCH may be used in the DISPLAY statement. The mnemonic-name associated with SYSCHAN-n may be used in the WRITE statement.

   - SYSIN refers to the job stream device.

   - SYSCONSOLE refers to the system message lines of the workstation activating the task and to the system log file. If the task is not activated from a workstation or the system does not support a workstation, then SYSCONSOLE refers to the system console and the system log file. Use SYSCONSOLE when a reply from the operator is required.

   - SYSCOM refers to the 12-byte communications region within the job preamble. Note that the twelfth byte of this region is the user program switch indicator (UPSI) byte.

- SYSSWCH refers to the UPSI byte of the communications region. SYSSWCH is expanded by the compiler to an 8-byte storage area; each byte represents a switch. When condition-names are associated with SYSSWCH, the status is set:

  – On when any of the eight UPSI bytes are on

  – Off when all of the UPSI bytes are off

  When the mnemonic-name associated with SYSSCH appears:

  – In an ACCEPT statement, character value 0 or 1 (hexadecimal F0 or F1) is returned for each UPSI byte.

  – In a DISPLAY statement, the status of each corresponding UPSI byte is set on with character value 1 (hexadecimal F1) and off with character value 0 (hexadecimal F0). Any other character leaves the status unchanged.

- SYSSWCH-n refers to the individual switches within SYSSWCH. They are numbered from left to right: SYSSWCH-0 through SYSSWCH-7.

  NOTE:

  SYSSWCH-0 is reserved for the COBOL object-time debugging switch. (See 12.2.3.)

  The status of SYSSWCH-n is set on with any character other than hexadecimal F0 and set off with hexadecimal F0.

- SYSTEM-SHUTDOWN refers to an internal switch set on when the operator enters a SHUTDOWN command through the console. When SYSTEM-SHUTDOWN status is on (with hexadecimal value F1), a program that detects this status should begin termination procedures including closing all open files, displaying program information, and executing a STOP RUN statement. The status of SYSTEM-SHUTDOWN is off with hexadecimal value F0.

- SYSLST refers to the system log file.

- SYSLOG refers to the system message lines of the workstation activating the task and to the system log file. If the task is not activated from a workstation or the system does not support a workstation, then SYSLOG refers to the system console and the system log file. Use SYSLOG when no reply from the operator is expected.

- SYSCHAN-n equates a particular channel (n) on the printer loop to mnemonic-name-5. Mnemonic-name-5 may appear only in a WRITE statement. SYSCHAN 1 and 7 are normally used for form overflow and top-of-page, respectively.

- SYSSCOPE is treated as SYSCONSOLE. It is provided for compatibility with VS/9 COBOL 74 language.

- SYSTERMINAL or SYSOUT refers to system MESSAGE lines of the workstation initiating the COBOL program task. If the task is not activated from a workstation or the system does not support a workstation, then SYSTERMINAL or SYSOUT refers to the system console, SYSCONSOLE, but not the system log file.

- SYSFORMAT refers to a workstation in data mode (attached to a program) that calls screen format services. The lfdname in the required ASSIGN clause is a 1- to 8-character alphanumeric name assigned to the workstation.

- SYSWORK refers to a workstation in data mode. The lfdname in the required ASSIGN clause is a 1- to 8-character alphanumeric name assigned to the workstation.

Within the SPECIAL-NAMES paragraph, each SYSFORMAT, SYSWORK, or SYSTERMINAL clause must be specified before any alphabet-name clauses.

If the run unit is divided into subprograms, a particular lfdname in the SYSFORMAT or SYSWORK clause can be used in only one program.

The CONTROL AREA clause specifies a 40-character area that receives data describing workstation activity. That area may be defined in the WORKING-STORAGE or LINKAGE section. Its implicit description is:

```
Ø5 WS-ID                  PIC 999.
Ø5 FILLER                 PIC X.
Ø5 WS-STATUS              PIC XX.
Ø5 FUNCTION-KEY           PIC 99.
Ø5 FORMAT-NAME            PIC X(8).
Ø5 NUMBER-CONNECTED       PIC 99.
Ø5 SIZE-OF-DATA-TRANSFER  PIC 9(5).
Ø5 FILLER                 PIC X(17).
```

Specification of a CONTROL AREA clause enables the COBOL program to be aware of the details of interaction with a workstation, especially a multivolume workstation. When a workstation mnemonic-name is declared with a CONTROL AREA clause, each ACCEPT or DISPLAY statement to that workstation must include an ON EXCEPTION clause. Specification of the WITH FUNCTION-KEYS phrase causes the COBOL program to report function key input in the control area, and, unless overridden by response indicators, to cause activation of the ON EXCEPTION clause after reception of function key data.

When the WITH FUNCTION-KEYS phrase is specified, ACCEPT and DISPLAY statements that reference the workstation must appear within only one program.

Specification of the WITH CONNECT-FREE phrase causes the COBOL program to take an exception path on an ACCEPT statement after a workstation connects to a multivolume workstation or disconnects from it. In the absence of a CONTROL AREA clause, the COBOL system defaults to minimal, but operationally effective, support for multivolume workstations.

The control area specified by the CONTROL AREA clause is a repository for data supplied by the COBOL system. The content of each field is defined as follows:

- WS-ID identifies the particular device, which is part of a multivolume workstation configuration, that participated in the most recently performed ACCEPT or DISPLAY statement to the corresponding workstation file.

- WS-STATUS reports the 2-character error status for the most recently performed ACCEPT or DISPLAY statement to the corresponding workstation file. Status key details are presented in Table 4–1.

- FORMAT-NAME is the name of the screen format that is active on the workstation terminal that was the object of the most recently performed ACCEPT or DISPLAY statement. This field is for information purposes only. Thus, it is used as a read-only field. Altering the contents of the FORMAT-NAME field never changes the screen format currently active on any terminal.

- FUNCTION-KEY holds the integer denoting the keyboard function key pressed prior to the most recently performed ACCEPT statement. It is zero if no function key was pressed. The FUNCTION-KEY field is maintained only if the WITH FUNCTION-KEYS phrase is specified.

- NUMBER-CONNECTED holds the number of terminals that are currently connected to the workstation. If the workstation is not multivolume, this number will be either zero or one.

- SIZE-OF-DATA-TRANSFER holds the number of characters actually delivered to or received from the workstation terminal screen. If a screen format is in effect, this number reflects the number of characters required by that format. If a screen format is not in effect, this number, after an ACCEPT statement, represents the number of characters entered by the workstation operator but not exceeding the number requested.

Table 4—1. Status Key Values for Workstations

Status Key 1	Status Key 2
0 - Successful completion	0 - No further information
1 - At end	0 - Function key 15 received 6 - Only active terminal has disconnected. (Status keys 98 may take precedence.)
2 - Invalid format	3 - Format not found 4 - Format constructed incorrectly
3 - Permanent error	0 - No further information
9 - Workstation exception	1 - Terminal not compatible with format 2 - Statement not compatible with format 3 - Data not compatible with format 4 - Data area not large enough for format 5 - Function key, no data 7 - New device connected, no data 8 - Device disconnected (freed), no data 9 - Device not connected

The WITH CONNECT-FREE phrase specifies that an exception path is to be taken whenever a terminal connects to or disconnects from a multivclume workstation configuration. The details of CONNECT-FREE reporting are in Table 4–2.

The WITH FUNCTION-KEYS phrase specifies that whenever function key input is received in an ACCEPT statement, that function key value is to be reported in the control area specified by the CONTROL AREA clause. If the active screen format converts the function key to an indicator, the indicator portion of the accept data is returned to the COBOL program, and the ON EXCEPTION clause is not activated; otherwise, the ON EXCEPTION clause is activated. A function key and data (other than response indicators) are never returned at the same time. The details of function key processing are in Table 4–3.

WS-ID has meaning only for multivolume workstations. It is the only field in the control area that the COBOL program might reasonably alter. WS-ID identifies the particular terminal to which a DISPLAY statement directs its data. Likewise, WS-ID identifies the particular terminal from which an ACCEPT SPECIFIC statement will take its data. If a USING phrase is present on a general ACCEPT statement (i.e., not an ACCEPT SPECIFIC statement), the screen format that is named by the USING phrase is selected only for the terminal indicated by WS-ID, not for all the terminals of the multivolume workstation.

On each transaction with a workstation, the field WS-ID is updated with a number that identifies the particular terminal, within a multivolume workstation, that participated in the transaction. The WS-ID field does not need an initial value. Assignment of an initial value to the WS-ID field has no effect on the behavior of the COBOL program. It is the responsibility of the COBOL program to ensure that the WS-ID field contains a terminal number that is valid for the implicit workstation lfdname.

One way to guarantee that this will happen is never to alter the value of the WS-ID field. Another way is not to provide a control area using the CONTROL AREA clause. The only reasons for changing the WS-ID field are to display data to a particular terminal (DISPLAY statement) that is not the one that most recently supplied input or to accept data (ACCEPT statement) from a specific terminal rather than from the terminal that responded first.

Table 4—2. Effects of CONNECT-FREE Reporting

Workstation Options	Response to CONNECT-FREE Phrase
CONTROL AREA clause with CONNECT-FREE phrase	Set status byte. Set WS-ID field to report the device that was connected or freed. Update NUMBER-CONNECTED clause. Activate the EXCEPTION clause.
CONTROL AREA clause without CONNECT-FREE phrase	Update NUMBER-CONNECTED clause. If NUMBER-CONNECTED = 0, set status for end-of-file and execute the EXCEPTION clause. Otherwise, do not return control to the COBOL program until data is received.
CONTROL AREA clause not specified	If no terminal remains connected, terminate the program abnormally. Otherwise, do not return control to the COBOL program until data is received.

Table 4—3. Effects of FUNCTION-KEYS Input

Workstation Options	Response to FUNCTION-KEYS Input	
	Response Indicator Set by Function Key	Response Indicators Absent or Unaffected
CONTROL AREA clause with FUNCTION-KEYS phrase	Return indicators without screen data. Set FUNCTION-KEYS clause. Do not activate the EXCEPTION clause.	Set FUNCTION-KEY clause. Activate the EXCEPTION clause.
CONTROL AREA clause without FUNCTION-KEYS phrase	Return indicators without screen data. Do not set FUNCTION-KEYS clause. Do not activate the EXCEPTION clause.	Ignore the function key input. Do not return control to the COBOL program until data is received.
CONTROL AREA clause not specified	Return indicators without screen data.	Ignore the function key input. Do not return control to the COBOL program until data is received.

If the CONTROL AREA clause is omitted, multivolume workstation processing is restricted in the following ways:

1. Each DISPLAY statement is always directed to the terminal that completed the most recent ACCEPT statement.

2. Each ACCEPT SPECIFIC statement is always directed to the terminal that completed the most recent ACCEPT statement.

3. An EXCEPTION path cannot be specified.

If a failure occurs during the first attempt to access a particular workstation, the WS-STATUS field is set to 30 and the remainder of the control area (CONTROL AREA clause) is undefined.

For errors arising after the first access, the control area fields have these meanings:

- WS-ID - The ID of the terminal to respond to a general ACCEPT statement; or the valid ID given in the WS-ID field upon execution of the statement; or, if 0 was given, then 1; otherwise, the field is undefined.

- WS-STATUS - As defined in Table 4-1.

- FUNCTION-KEYS - Unchanged by a DISPLAY statement; or 00 for an ACCEPT statement not receiving function key input; or the actual function key received by an ACCEPT statement.

  A function key is never received at the same time data is received. However, for screen formats having response indicators, receiving a function key is a sign that indicator data is present.

  Data is never present, however, when the ON EXCEPTION clause is activated.

- FORMAT-NAME - The name of the screen format that is defined for the workstation terminal named in the WS-ID field; if no screen format is defined for that terminal, the field contains the LOW-VALUE constant.

- NUMBER-CONNECTED - Always reports the number of terminals connected to the workstation.

- SIZE-OF-DATA-TRANSFER - Usually undefined in the presence of an error.

  For status key 94, SIZE-OF-DATA-TRANSFER field is the smallest number of characters required for the transaction.

When a terminal connects to a multivolume workstation declared with the WITH CONNECT-FREE phrase, the connect event is reported to the COBOL program, in lieu of returning input data, in response to the next ACCEPT statement (but not an ACCEPT SPECIFIC statement).

When a terminal frees from a multivolume workstation declared with the WITH CONNECT-FREE phrase, the free event is reported to the COBOL program, in lieu of returning input data, in response to the next ACCEPT statement (but not an ACCEPT SPECIFIC statement).

If a COBOL program uses screen format services for a multivolume workstation without specifying the CONNECT-FREE phrase, an initial screen format must be supplied via job control language, and this screen format must be input-only.

If a COBOL program uses screen format services via job control language for a SYSWORK workstation, whether single-volume or multivolume, with or without specifying the CONNECT-FREE phrase:

- An initial screen format must be supplied via job control language. In addition, if the workstation is multivolume and the CONNECT-FREE phrase is not specified, the screen format furnished via job control language must be an input-only screen format.
- All ACCEPT and DISPLAY statements apply to the screen named by the initial screen parameter.
- The USING clause cannot be used on any ACCEPT or DISPLAY statements to the SYSWORK device.
- No data conversion may be implicit in the screen format; that is, all fields of the screen format must be specified implicitly or explicitly as USAGE IS DISPLAY.
- Only one identifier may receive data in an ACCEPT statement.
- The ACCEPT SPECIFIC statement may not be used with SYSWORK workstations.

Additional information about screen format services is described in the screen format services concepts and facilities, UP-8802 (current version).

4. In the SYSSWCH[-n] clause, at least one condition-name must be associated with a switch. The status of a switch is specified by condition-names and interrogated by testing the condition-names. (See 6.4.1.4.)

   Example:

   An individual switch can be interrogated by using condition-name in the ON/OFF STATUS option. For instance, in the following example, control is transferred to procedure-name-1 if switch 5 is ON.

```
ENVIRONMENT DIVISION.
          .
          .
          .
SPECIAL-NAMES.
     SYSSWCH-5 ON STATUS IS FIVON, OFF STATUS IS FIVOFF.
          .
          .
          .
PROCEDURE DIVISION.
          .
          .
          .
     IF FIVON GO TO procedure-name-1.
```

   In essence, SYSSWCH-5 is a conditional variable with the condition-names FIVON and FIVOFF, which are similar to level-88 entries.

   The condition-names FIVON and FIVOFF are defined and equated with ON and OFF, respectively, by the COBOL compiler and must not be defined elsewhere in the COBOL program.

5. The mnemonic-name associated with SYSCOM may be used in the UPON phrase of the DISPLAY statement to effect the passing of information to other programs within the job or in the FROM phrase of the ACCEPT statement to retrieve information from a program within the job.

There is only one communications region to store or retrieve information within a job. The entire region will be overwritten if more than one DISPLAY statement referencing SYSCOM is executed.

Examples:

```
Program A
          .
          .
          .
          SYSCOM IS OUT-PARAM
          .
          .
          .
      77 PROG-MESSAGE PIC.X(12) VALUE "317402SQ11Ø".
          .
          .
          .
          DISPLAY PROG-MESSAGE UPON OUT-PARAM
Program B
          .
          .
          .
          SYSCOM IS INPUT-PARAM
          .
          .
          .
      77 PARAM-AREA PIC X(12) VALUE ZEROS.
          .
          .
          .
          ACCEPT PARAM-AREA FROM INPUT-PARAM.
```

6. The mnemonic-name associated with SYSSWCH[-n] may be used in the FROM option of the ACCEPT statement to gain access to the contents of SYSSWCH[-n], or in the UPON option of the DISPLAY statement to set or change the contents of SYSSWCH[-n].

   Example 1:

   All 8 task switches can be interrogated by use of the ACCEPT verb. This is shown in the following example, where procedure-name-1 is performed if the SYSSWCH-2, SYSSWCH-4, SYSSWCH-6, and SYSSWCH-7 switches are ON and the others are OFF.

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
    SYSSWCH IS mnemonic-name-1.
DATA DIVISION.
WORKING-STORAGE SECTION.
Ø1 identifier PICTURE X(8).
PROCEDURE DIVISION.
    ACCEPT identifier FROM mnemonic-name-1.
    IF identifier = "ØØ1Ø1Ø11".
            PERFORM procedure-name-1.
```

Example 2:

To set or change the contents of SYSSWCH, the DISPLAY verb may be used as follows:

```
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
    SYSSWCH IS SWITCH.
    SYSSWCH-3 IS SWITCH-3.
PROCEDURE DIVISION.
    DISPLAY "11ØØØ1ØØ" UPON SWITCH ①
    DISPLAY 1 UPON SWITCH-3. ②
    DISPLAY identifier UPON SWITCH. ③
```

NOTES:

① SYSSWCH will now contain "11000100".

② SYSSWCH-3 will now contain 1; the other switches remain unchanged.

③ The 8 switches in SYSSWCH (0 through 7) are set ON or OFF, depending on the contents of the 8-character identifier.

7. The mnemonic-names associated with SYSCHAN-n may be used in the WRITE statement. SYSCHAN-n refers to a position in a printer vertical format buffer or form control loop; n ranges from 1 through 15, depending on the specific printer used. The SYSCHAN-n clause is accepted for compatibility with existing SPERRY UNIVAC COBOL compilers.

8. The alphabet-name clause provides a means for relating a name to a specified character code set or collating sequence. When alphabet-name is referenced in the PROGRAM COLLATING SEQUENCE clause, or the COLLATING SEQUENCE phrase of a SORT or MERGE statement, the alphabet-name clause specifies a collating sequence. When alphabet-name is referenced in a CODE-SET clause in a file description entry, the alphabet-name clause specifies a character code set.

    - 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 in the EBCDIC character set as specified in Appendix J.

    - If the NATIVE phrase is specified, the character code set or collating sequence identified is EBCDIC.

    - If the STANDARD-0 phrase is specified, the character code set or collating sequence identified is that defined by the International Standards Organization for the 8-bit *Coded Character Set for Information Interchange, IS 646 — 1973, International Reference Version*. The collating sequence of this code set is identical to STANDARD-1.

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

  a. The value of each literal specifies:

     (1) The ordinal number of a character within the EBCDIC character set, if the literal is numeric. The ordinal number of a character is always one greater than the binary value of a character. For example, hexadecimal 00 is the first character (ordinal position 1), and hexadecimal 01 is the second character (ordinal position 2).

     (2) The actual character within the EBCDIC 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.

  b. 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.

  c. Any characters within the EBCDIC collating sequence that 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 EBCDIC collating sequence.

  d. If the THROUGH phrase is specified, the set of contiguous characters in the EBCDIC 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 EBCDIC character set in either ascending or descending sequence.

  e. If the ALSO phrase is specified, the characters of the EBCDIC 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.

9. The literals in the literal phrase of the alphabet-name clause are specified as follows:

   - If numeric, they must be unsigned integers and must have a value within the range of 1 through 256.

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

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

11. The words THRU and THROUGH are equivalent.

12. 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.

13. 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.

14. The literal specified in the CURRENCY SIGN IS clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a nonnumeric literal of one character and must not be one of the following characters:

    - Digits 0 through 9
    - Alphabetic characters ABCDLPRSVXZ or space
    - Special characters * + - , . ; ( ) " / = '

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

15. 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.

4.4. INPUT-OUTPUT SECTION

The input-output section of the environment division is used to specify the input/output media for the files used by the program and to provide information needed for most efficient transmission of data between external media and the object program.

Format:

```
┌INPUT-OUTPUT SECTION.            ┐
│FILE-CONTROL. {entry.}...        │
└[I-O-CONTROL. entry.]            ┘
```

4.4.1. FILE-CONTROL Paragraph

Function:

The FILE-CONTROL paragraph names each file and allows specification of other file-related information. A separate format is required for each type of file organization: sequential, relative, indexed, ISAM*, ISAM*, and sort.

Refer to Section 8 for further information on sequential, relative, indexed, ISAM*, and ISAM* files, and to Section 9 for a summary of the sort feature.

Applies only to 90/25, 90/30, 90/30 B and 90/40 systems

Format 1 (Sequential Files):

```
FILE-CONTROL.
     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 (Relative Files):

```
FILE-CONTROL.
     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    }  ,RELATIVE KEY IS data-name-1 }]
     [                { {|DYNAMIC|}                                 }]
     [;FILE STATUS IS data-name-2].
```

Format 3 (Indexed Files):

```
FILE-CONTROL.
     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].
```

Format 4 (SAM* Files):

```
FILE-CONTROL.
        SELECT [OPTIONAL] file-name
        ASSIGN TO implementor-name-1 [,implementor-name-2] ...
        ⎡;RESERVE integer-1 ⎡AREA ⎤⎤
        ⎣                   ⎣AREAS⎦⎦
        ;ORGANIZATION IS SAM
        [;ACCESS MODE IS SEQUENTIAL]
        [;FILE STATUS IS data-name-1].
```

Format 5 (ISAM* Files):

```
FILE-CONTROL.
        SELECT file-name
        ASSIGN TO implementor-name-1 [,implementor-name-2] ...
        ⎡;RESERVE integer-1 ⎡AREA ⎤⎤
        ⎣                   ⎣AREAS⎦⎦
        ;ORGANIZATION IS ISAM
        ⎡;ACCESS MODE IS ⎧SEQUENTIAL⎫⎤
        ⎢                ⎨RANDOM    ⎬⎥
        ⎣                ⎩DYNAMIC   ⎭⎦
        ;RECORD KEY IS data-name-1
        [;FILE STATUS IS data-name-2].
```

Format 6 (Sort or Merge Files):

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

Rules:

1. The SELECT clause must be specified first. The order of the remaining clauses is optional.

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 or sort/merge file description entry in the data division.

3. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium.

4. All files must be assigned to an implementor-name. Any implementor-name beyond the first for a file is treated as comments.

5. Implementor-name is in the form of device-lfdname-mode.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

The format and options for each file organization are presented as follows:

- Sequential Files

$$\left\{\begin{array}{l}
\underline{\text{CARDREADER}}\text{-lfdname-}\underline{\text{F}} \\
\underline{\text{CARDPUNCH}}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{F}}\\ \underline{\text{U}}\\ \underline{\text{V}}\end{array}\right\} \\
\underline{\text{PRINTER}}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{FC}}\\ \underline{\text{UC}}\\ \underline{\text{VC}}\end{array}\right\} \\
\underline{\text{TAPE}}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{F}}\\ \underline{\text{U}}\\ \underline{\text{V}}\\ \underline{\text{FC}}\\ \underline{\text{UC}}\\ \underline{\text{VC}}\end{array}\right\} \\
\left\{\begin{array}{l}\underline{\text{DISC}}\\ \underline{\text{DISK}}\end{array}\right\}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{F}}\\ \underline{\text{V}}\\ \underline{\text{FC}}\\ \underline{\text{VC}}\end{array}\right\}
\end{array}\right\}$$

- Relative, Indexed, and ISAM* Files

$$\left\{\begin{array}{l}\underline{\text{DISC}}\\ \underline{\text{DISK}}\end{array}\right\}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{F}}\\ \underline{\text{V}}\end{array}\right\}$$

- SAM* Files

$$\left\{\begin{array}{l}\underline{\text{DISC}}\\ \underline{\text{DISK}}\end{array}\right\}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{F}}\\ \underline{\text{V}}\\ \underline{\text{FC}}\\ \underline{\text{VC}}\end{array}\right\}$$

- Sort and Merge Files

$$\left\{\begin{array}{l}\underline{\text{DISC}}\\ \underline{\text{DISK}}\\ \underline{\text{TAPE}}\end{array}\right\}\text{-lfdname-}\left\{\begin{array}{l}\underline{\text{F}}\\ \underline{\text{V}}\end{array}\right\}$$

The implementor-name in the form of device-lfdname-mode has three subfields:

- The device field specifies the type of device associated with the file. The types of devices supported are CARDPUNCH, CARDREADER, DISC (or DISK), PRINTER, and TAPE. DISC and DISK are equivalent; TAPE refers to magnetic tape.

- Lfdname specifies the lfdname to the job control definition of the file. The lfdname is a 1- to 8-character alphanumeric field. For programs using the sort/merge feature, lfdnames in the form DMxx (xx=01, 02,...,08) or SMxx (xx=01, 02,...,06) should not be used because the sort/merge feature uses these lfdnames for scratch work files.

  If the run unit contains multiple implementor-names for the same lfdname, then only one file associated with the lfdname can be opened at a time.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

- Mode is a 1- or 2-character field that specifies the format of the records in the file. It may be F, FC, V, VC, U, or UC. The character C indicates the presence of a device-independent control character for a printer-destined file. When FC, VC, or UC is specified for a printer-destined file, the compiler appends a device-independent control character preceding the logical record.

Mode F indicates fixed-length records and can be specified only when each of the logical records in a file has the same length. When F is specified, the following rules apply:

- Format 2 of the OCCURS clause must not be specified within any record description for the file.

- If the BLOCK CONTAINS clause is specified in the file description entry, it must contain a fixed number of records.

- If more than one record description entry is specified for a file description, each record in the file must have the same length.

- Record-length and block-descriptor fields are not present with fixed-length records.

- If the RECORD CONTAINS clause is specified, it must specify a fixed number of characters in the record.

Mode V indicates variable-length records. When V is specified, the following rules apply:

- For tape, SAM*, or ISAM* files, if the BLOCK CONTAINS clause is specified in the file description entry, all the logical records comprising a block must be wholly contained within the block.

- Each variable-length logical record is preceded by a control field containing the length of the logical record. This field is generated by the compiler and is not available to the user.

Mode U indicates an undefined format and may be used for any combination of record descriptions, either fixed or variable. U-mode is comparable to V-mode except that U-mode records may not be blocked and have no preceding control field. When U is specified, the BLOCK CONTAINS clause in the file description entry is not required.

The hyphens shown in the format must appear in this form of the implementor-name.

6. The RESERVE clause specifies the number of input/output buffer areas allocated for the file.

   The value of integer-1 may be 1 or 2. For sequential files, two areas may be reserved. For relative files, if the ACCESS MODE is sequential, two areas may be specified and if the ACCESS MODE is dynamic or random, only one area may be specified. For indexed files, only one area is permitted. For SAM* and ISAM* files, two areas may be reserved.

   If the value of integer-1 specifies two areas where the operating system permits only one area, the compiler reserves only one area regardless of the value specified in the RESERVE clause.

   If the RESERVE clause is not specified, the compiler supplies a default value compatible with the operating system specifications. The compiler-supplied default value is shown in Table 4-4.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

Table 4—4. Compiler Default Value of the RESERVE Clause

File Organization	Access Mode	Default Integer-1 Value
SEQUENTIAL	All	2
RELATIVE	SEQUENTIAL	2
RELATIVE	RANDOM	1
RELATIVE	DYNAMIC	1
INDEXED	All	1
SAM	All	2
ISAM	All	2

7. When the FILE STATUS clause is specified, a value is moved by the operating system into the data item specified by data-name after the execution of every statement that refers to the file either explicitly or implicitly. This value indicates the status of execution of the statement. (See Section 8.)

8. The organization clause specifies the logical structure of a file. The file organization is established at the time a file is created and, subsequently, cannot be changed.

NOTE:

Rules 9 through 14 pertain to sequential and SAM files only.*

9. Data-name-1 must be defined in the data division as a 2-character alphanumeric data item and must not be defined in the file section or the communication section.

10. Data-name-1 may be qualified.

11. In format 1, if the ORGANIZATION IS SEQUENTIAL clause is not specified, it is implied.

In format 4, the ORGANIZATION IS SAM* clause is required. This clause specifies that the file is to be supported by disk sequential access method data management (disk SAM). If this clause is not specified, ORGANIZATION IS SEQUENTIAL is assumed. (See 8.3.)

12. The OPTIONAL phrase may only be specified for input files that are not necessarily present each time the object program is executed.

13. 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.

14. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

NOTE:

Rules 15 through 26 pertain to relative files only.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

15. Data-name-2 must be defined in the data division as a 2-character alphanumeric data item and must not be defined in the file section or the communication section.

16. Data-name-1 and data-name-2 may be qualified.

17. If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be specified for that file.

18. Data-name-1 must not be defined in a record description entry associated with that file-name.

19. The data item referenced by data-name-1 must be defined as an unsigned integer.

20. The ORGANIZATION clause is required. If the ORGANIZATION IS RELATIVE clause is not specified, ORGANIZATION IS SEQUENTIAL is assumed.

21. 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.

22. If the access mode is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

23. When the access mode is dynamic, records in the file may be accessed either sequentially or randomly or both. (See rules 22 and 24.)

24. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

25. All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the logical ordinal position of the record in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, ...

26. The data item specified by data-name-1 is used to communicate a relative record number between the COBOL object program and the operating system.

NOTE:

Rules 27 through 40 pertain to indexed files only.

27. Data-name-1, data-name-2, and data-name-3 may be qualified.

28. 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.

29. The maximum size of the data item referenced by data-name-1 or data-name-2 may not exceed 80 bytes. Neither data-name-1 nor data-name-2 can describe an item whose size is variable. (See 5.3.3.7.)

30. 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.

31. Data-name-3 must be defined in the data division as a 2-character alphanumeric data item and must not be defined in the file section, or the communication section.

32. The ORGANIZATION IS INDEXED clause is required. If this clause is not specified, ORGANIZATION IS SEQUENTIAL is assumed.

33. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

34. 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.

35. If the access mode is random, the value of the record key data item indicates the record to be accessed.

36. When the access mode is dynamic, records in the file may be accessed either sequentially or randomly or both. (See rules 34 and 35.)

37. The RECORD KEY clause specifies 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.

38. An ALTERNATE RECORD KEY clause specifies an alternate record key for the file. This alternate record key provides an alternate access path to records in an indexed file. A maximum of four alternate record keys may be specified for an indexed file.

39. 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 those 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.

40. 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.

NOTE:

Rules 41 through 51 pertain to ISAM files only.*

41. When ORGANIZATION IS ISAM is specified, the file is to be processed by the ISAM data management.

42. When the access mode is sequential, records in the file are accessed in the sequence dictated by the file organization. For ISAM files, this sequence is the order of ascending record key values within a given key of reference.

43. If the access mode is random, the value of the record key data item indicates the record to be accessed.

44. When the access mode is dynamic, records in the file may be accessed sequentially or randomly or both. (See rules 41 and 42.)

45. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

46. The RECORD KEY clause specifies the record key for the file. The values of the record key must be unique among records of the file. This record key provides the access path to records in an ISAM file.

47. Data-name-1 and data-name-2 may be qualified.

48. The data item referenced by data-name-1 must be defined as an alphanumeric data item within a record description entry associated with that file-name.

49. The size of the data item referenced by data-name-1 must be greater than 2 and less than or equal to 249 bytes. Data-name-1 must not describe an item whose size is variable. (See 5.3.3.7.)

50. Data-name-2 must be defined in the data division as a 2-character alphanumeric data item and must not be defined in the file section, or the communication section.

51. The data description of data-name-1 and its relative location within a record must be the same as that used when the file was created.

NOTE:

Rule 52 pertains to sort or merge files only.

52. Only the ASSIGN clause is permitted to follow file-name in the FILE-CONTROL paragraph for a sort or merge file.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

4.4.2. I-O-CONTROL Paragraph

Function:

The I-O-CONTROL paragraph specifies the points at which rerun is to be established and the main storage area to be shared by different files. In addition, this paragraph specifies the location of files on a multiple file reel for sequential file organization as well as special input/output techniques for file processing using the APPLY clauses.

Format:

```
[I-O-CONTROL.

   [;RERUN ON {{DISC}-lfdname-{1}} EVERY integer-1 RECORDS OF file-name-1]...
   [          {{DISK}         {2}}                                       ]
   [          {{TAPE}            }                                       ]
   [          {[lfdname]         }                                       ]

   [;SAME  [RECORD    ]  AREA FOR file-name-2{,file-name-3}...]...
   [       [SORT      ]                                       ]
   [       [SORT-MERGE]                                       ]

   [;MULTIPLE FILE TAPE CONTAINS file-name-4 [POSITION integer-2]
   [,file-name-5 [POSITION integer-3]]...] ...

   [;APPLY BLOCK-COUNT ON {file-name-6 [file-name-7] ...}]...
   [                      {TAPES                        }]
   [;APPLY CYLINDER-INDEX AREA OF integer-4 INDICES ON file-name-8
       [,file-name-9] ...] ...
   [;APPLY CYLINDER-OVERFLOW AREA OF integer-5 PERCENT ON file-name-1Ø
       [,file-name-11] ...] ...
   [;APPLY VERIFY ON file-name-12 [,file-name-13] ...] ...
   [;APPLY INDEX-AREA OF integer-6 CHARACTERS ON file-name-14
       [,file-name-15]...]... .]
```

Rules:

1. The I-O-CONTROL paragraph is optional.

2. The RERUN clause specifies when and where the rerun information is recorded. The rerun information is recorded on the device specified whenever integer-1 records of file-name-1 are processed. File-name-1 may be any type of file with any organization or access except a sort or merge file.

3. The value of integer-1 in the RERUN clause must be within the range of 1 to 8,388,607.

4. There are two forms of the implementor-name in the RERUN clause.

   a. The form

   ```
   {DISC}-lfdname- {1}
   {DISK}          {2}
   {TAPE}
   ```

   specifies a dedicated rerun receiver.

   - DISC, DISK, and TAPE are the types of devices supported for a user file dedicated for receiving checkpoint records.

- The lfdname field is a 1- to 8-character alphanumeric field. This field specifies the lfdname of the dedicated receiver file.

- The field

  $\left\{\begin{matrix}1\\ \underline{2}\end{matrix}\right\}$

  is a 1-character field, where the value 1 indicates that all checkpoint records are to be written consecutively on one dedicated receiver file, and the value 2 indicates that checkpoint records are to be written alternately on two dedicated receiver files, each file containing only the latest alternate checkpoint record. When two dedicated receiver files are specified, the INIT parameter must be designated in the LFD job control statements for both receiver files.

b. The form `lfdname` specifies the name of an output data file on which both data records and checkpoint records are to be written. The name specified must be the lfdname of a standard sequential EBCDIC tape file described by an FD entry with standard system labels.

   NOTE:

   This form of the implementor-name is not supported in the mixed mode or consolidated data management mode.

5. The filename of a dedicated rerun receiver file is generated by the compiler by using the lfdname specified in the lfdname field of the RERUN clause.

   If one dedicated receiver file is specified, the lfdname is used as the filename of the rerun receiver.

   If two dedicated receiver files are specified, a suffix, A or B, is appended as the last character of a given lfdname of seven or fewer characters, or the suffix replaces the last character of an 8-character lfdname to form a unique file-name for each receiver file. The file-names for the two receiver files are lfdnameA and lfdnameB.

   Odd-numbered checkpoints are written on the file:

   ```
   lfdnameA
   ```

   Even-numbered checkpoints are written on the file:

   ```
   lfdnameB
   ```

6. More than one RERUN clause may be specified, but no two of them may specify the same file-name-1.

7. The SAME AREA clause specifies that two or more files that do not represent sort or merge files are to use the same main storage area during processing. The area being shared includes all storage area assigned to the files specified; therefore, only one file may be open at any given time. (See rule 9c.)

8. The SAME RECORD AREA clause specifies that two or more files are to use the same main storage area for processing of the current logical record. All the files may be open at the same time. A logical record in the SAME RECORD AREA is considered as: (1) a logical record of each opened output file whose file-name appears in this SAME RECORD AREA clause and (2) a logical record 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.

9. More than one SAME clause may be included in a program; however, the following rules also apply:

   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 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.

10. The files referenced in the SAME AREA or SAME RECORD AREA clause need not all have the same organization or access.

11. The SAME SORT AREA or SAME SORT-MERGE AREA clause is for documentation only.

12. The MULTIPLE FILE clause applies to sequential files only and is required when two or more files share 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 are 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.

13. All files sharing the same physical reel of tape must specify the LABEL RECORDS STANDARD or data-name clause in the associated FD entries; the LABEL RECORDS OMITTED clause is not permitted.

14. The REVERSE phrase of the OPEN statement must not be used for files sharing the same physical reel of tape.

15. The APPLY BLOCK-COUNT clause is used only for tape files. For each file-name specified, this clause inserts a 3-byte block number at the beginning of each block on tape.

    If the TAPES option is specified, all tape files present are affected. This clause must be specified for all input tape files that contain a block count.

16. The APPLY CYLINDER-INDEX AREA clause is used only for ISAM files (ORGANIZATION IS ISAM). The clause indicates that sufficient main storage area should be allocated to contain integer-4 top index entries.

    The method for calculating the value of integer-4 is described in detail in the consolidated data management macroinstructions programming guide, UP-9979 (current version).

    If the file already exists, use the following formula to determine the value of integer-4:

    ```
    n = b/(s+3)
    ```

where:

n

Is integer-4 of the APPLY clause.

b

Signifies bytes that are required for main storage and that can be obtained from a display of VTOC. The number of bytes is shown under the heading: Bytes Required for Main Storage.

s

Signifies size of the record key.

NOTE:

If the remainder of the divide operation in the formula is not equal to zero, add 1 to the quotient (i.e., add 1 to n).

17. The APPLY CYLINDER-OVERFLOW AREA clause is used only for ISAM files. It indicates that integer-5 percent of each cylinder in the prime data area will be reserved for cylinder overflow.

    If this clause is omitted, 20 percent of the cylinders specified as prime data area are automatically allocated. If no cylinder overflow is desired, 0 percent should be specified. If no overflow area exists, new records cannot be added to the file.

18. The APPLY VERIFY clause is used for any mass storage files. It requests verification of disk records after they have been written (read after write). If this clause is omitted, no verification is performed.

19. The APPLY INDEX-AREA clause is used only for indexed files (ORGANIZATION IS INDEXED). The clause specifies the size of the index-area used by MIRAM data management during the loading and retrieving of indexed MIRAM files. The size of the index-area for file retrieval, therefore, must be the same as the size when the file was created.

    Integer-6 must be a multiple of 256.

    If the clause is not specified, an index-area of 256 characters is provided for each indexed file defined in the file section.

5. Data Division

5.1. GENERAL

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

1. That which is contained in files and enters or leaves main storage from a specified area or areas

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

3. Constants that are defined by the user

The data division must be included in every COBOL source program.

5.2. STRUCTURE

5.2.1. Heading and Sections

The data division begins with the reserved words DATA DIVISION followed by a period and a space and is structured into file, working-storage, linkage, and communication sections. Each section is optional but, when used, must be in the following order:

```
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] ...   ┘          ┘
```

The file section defines the structure of data files. Each file is defined by a file description (FD) or sort merge-file description (SD) entry and is followed by one or more record descriptions. A record description describes all named items of data in the record.

The working-storage section describes records and noncontiguous data items that 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 called program and describes data items that are defined by the calling program and referred to by the called program. Its structure is the same as the working-storage section.

The communication section describes the data items that name the interface areas between the message control system and the object program. (See 5.6 and Section 14.)

5.2.2. Entries

Each data division entry begins with a level-indicator or a level-number, followed by one or more spaces, the name of the data item, and sequence of clauses describing the data item. The last clause is always terminated by a period followed by a space.

5.2.2.1. Level-Indicators

There are three types of level-indicators: FD, SD, and CD. FD indicates the start of a file description entry, SD indicates the start of a sort-file description entry, and CD indicates the start of a communication description entry.

5.2.2.2. Level-Numbers

Level-numbers are used to specify subdivisions of a logical record. 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, they 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.

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 49.

A 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 that 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. The following example indicates how level-numbers may be used to indicate this structure in the description of the record.

```
Ø1 RECORD-A
    Ø5 GROUP-ITEM-1
        Ø7 GROUP-ITEM-2
            Ø8 GROUP-ITEM-3
                1Ø ELEMENTARY-ITEM-1
                1Ø ELEMENTARY-ITEM-2
            Ø8 ELEMENTARY-ITEM-3
        Ø7 GROUP-ITEM-4
            Ø8 ELEMENTARY-ITEM-4
            Ø8 ELEMENTARY-ITEM-5
    Ø5 ELEMENTARY-ITEM-6
```

In the preceding example, both GROUP-ITEM-3 and ELEMENTARY-ITEM-3 are part of GROUP-ITEM-2; and GROUP-ITEM-2 and GROUP-ITEM-4 are part of GROUP-ITEM-1. Therefore, the level numbers assigned to both GROUP-ITEM-3 and ELEMENTARY-ITEM-3 must be identical and must be greater than that assigned to GROUP-ITEM-2. Similarly, GROUP-ITEM-2 and GROUP-ITEM-4 must be assigned identical level numbers greater than that assigned to GROUP-ITEM-1.

The principal rules for assigning level-numbers are listed as follows:

- The level-number 01 is reserved exclusively for identifying a logical record.

- Level-numbers range from 01 through 49.

- An item at any level may be an elementary item when no items are subordinate to it.

- An item is contained in the preceding group, if the following conditions are met:

    - The item has been assigned a numerically higher level-number than that of the preceding group.

    - The item directly follows the group of which it is a part.

5.2.2.3. Special Level-Numbers

Three types of entries exist for which there is no true concept of level and for which the special level-numbers 66, 77, and 88 are assigned. The three types of entries are as follows:

1. Level-number 66 introduces entries that specify elementary items or groups by means of RENAMES clauses for the purpose of regrouping data items.

2. Level-number 77 introduces entries that specify noncontiguous data items but which are not subdivisions of other items and are not themselves subdivided.

3. Level-number 88 introduces entries that specify condition-names to be associated with particular values of a conditional variable.

5.3. FILE SECTION

The file section begins with the reserved words FILE SECTION followed by a period and a space. The file section contains file description (FD) entries and sort-merge-file description (SD) entries, each one followed by its associated record description entries.

In a COBOL program, an FD or SD entry represents the highest level of organization in the file section. The FD entry provides information about the physical structure and identification of a file and gives the names of data records associated with the file. The SD entry indicates the size and names of the data records associated with the file to be sorted or merged. There are no label procedures that the user can control, and the rules for blocking and storage are peculiar to the SORT and MERGE statements.

A record description consists of a set of data description entries that 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 it is followed by subordinate entries.

Format:

```
┌FILE SECTION.                                  ┐
│   ┌file-description-entry           ┐         │
│   └     {record-description-entry} ...┘...     │
│                                               │
│   ┌sort-merge-file-description-entry┐         │
│   └     {record-description-entry} ...┘...     │
└                                               ┘
```

5.3.1. File Description

Function:

A file description is written for each file processed in the program. The information contained therein pertains to the physical aspects, identification, and record names of the file. A file description consists of a level indicator (FD), a file-name, and a series of independent clauses that describe the physical and logical characteristics of the file. The FD entry itself is terminated by a period.

The functions and usage of the file description entry clauses are summarized in Table 5-1. Sample program entries are given in Figure 5-1.

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               }
                        {{data-name-0} ...     }
   [;VALUE OF {FILE-ID IS {data-name-1}[,PASSWORD IS {data-name-2}]}]
   [          {           {literal-1  }[             {literal-2  }]}]
   [          {PASSWORD IS {data-name-2}[,FILE-ID IS {data-name-1}]}]
   [          {            {literal-2  }[            {literal-1  }]}]
```

```
[;DATA {RECORD IS  } data-name-3 [,data-name-4] ...]
       {RECORDS ARE}
 ________________________________________________
| [;LINAGE IS {data-name-5} LINES                |
|             {integer-5  }                      |
|     [,WITH FOOTING AT {data-name-6}]           |
|                       {integer-6  }            |
|     [,LINES AT TOP {data-name-7}]              |
|                    {integer-7  }               |
|     [,LINES AT BOTTOM {data-name-8}]]          |
|                       {integer-8  }            |
|  - - - - - - - - - - - - - - - - - - -         |
| ¦;LINAGE IS SYSTEM LINES              ¦        |
|  - - - - - - - - - - - - - - - - - - -         |
|________________________________________________|
  [;CODE-SET IS alphabet-name].
```

Rules:

1. The level indicator FD identifies the beginning of a file description and must precede the file-name.
2. The clauses that follow the name of the file are optional except for LABEL RECORDS, and their order of appearance is optional.
3. One or more record description entries must follow the file description entry.

Table 5—1. File Description Entry Clauses

Clause	Usage	File Organization	Function
BLOCK CONTAINS	Optional*	All	Specifies block size or buffer size of a file
RECORD CONTAINS	Optional	All	Specifies logical record size
LABEL RECORDS	Required	All	Specifies whether labels are standard, omitted, or user's
VALUE OF	Optional	All	Indicates values of standard label items
DATA RECORDS	Optional	All	Specifies names of records in file
LINAGE	Optional	Sequential or SAM	Defines the size of a logical page
CODE-SET	Optional	Sequential	Specifies character code set used to represent data in sequential tape files

* Required in some instances. See rule 1 in 5.3.1.1.

```
Seq.
No.
         A   B
1        8   12        Text
--------------------------------------------------------------------
Ø3ØØ1Ø   DATA DIVISION.
Ø3ØØ2Ø   FILE SECTION.
Ø3ØØ3Ø   FD SAMPLE-FILE.
Ø3ØØ4Ø       BLOCK CONTAINS 5 RECORDS,
Ø3ØØ5Ø       RECORD CONTAINS 1ØØ CHARACTERS,
Ø3ØØ6Ø       LABEL RECORD IS STANDARD,
Ø3ØØ7Ø       VALUE OF FILE-ID IS "SAMPLE FILE",
Ø3ØØ8Ø       DATA.RECORD IS SAMPLE-RECORD-1, SAMPLE-RECORD-2.
Ø3ØØ9Ø   Ø1  SAMPLE-RECORD-1             PICTURE X(1ØØ).
Ø3Ø1ØØ   Ø1  SAMPLE-RECORD-2.
Ø3Ø11Ø       Ø2 NAME.
Ø3Ø12Ø           /3 GIVEN                PICTURE X(15).
Ø3Ø13Ø           Ø3 MIDDLE               PICTURE X(15).
Ø3Ø14Ø           Ø3 FAMILY               PICTURE X(2Ø).
Ø3Ø15Ø       Ø2 SEX                      PICTURE X.
Ø3Ø16Ø           88 MALE             VALUE IS "M", "B", "1".
Ø3Ø17Ø           88 FEMALE           VALUE IS "F", "G", "2".
Ø3Ø18Ø       Ø2 MARITAL-STATUS           PICTURE X.
Ø3Ø19Ø           88 SINGLE                VALUE "S".
Ø3Ø2ØØ           88 MARRIED               VALUE "M".
Ø31Ø1Ø           88 DIVORCED              VALUE "D".
Ø31Ø2Ø           88 WIDOWED               VALUE "W".
Ø31Ø3Ø           88 OTHER                 VALUE "O".
Ø31Ø4Ø       Ø2 ADDRESS.
Ø31Ø5Ø           Ø3 SPECIAL              PICTURE X(5).
Ø31Ø6Ø           Ø3 STREET               PICTURE X(13).
Ø31Ø7Ø           Ø3 CITY                 PICTURE X(13).
Ø31Ø8Ø           Ø3 STATE                PICTURE X(2).
Ø31Ø9Ø           Ø3 COUNTRY              PICTURE X(1Ø).
Ø311ØØ           Ø3 ZIP                  PICTURE X(5).
```

Figure 5—1. Sample File Section Entries

5.3.1.1. BLOCK CONTAINS Clause

Function:

The BLOCK CONTAINS clause specifies the size of the physical record or block. For processing more efficiently files in which the concept of grouping logical records into blocks is not applicable, this clause may be used to specify the size of buffers.

Format:

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

Rules:

1. This clause may be specified on any file but is only required when:

    - the file is assigned to TAPE and the block (or physical record) contains more than one logical record; or

    - the file ORGANIZATION IS SAM* or ISAM* and the block contains more than one logical record.

2. Integer-1, if used, is for documentation only.

3. When the word RECORDS is used, integer-2 defines the block size in terms of the number of records (using maximum 01 record size) in each block. When variable-length records are blocked, more than integer-2 records may be grouped in a block due to some records being smaller than the maximum 01 record size.

4. When the word CHARACTERS is specified, integer-2 specifies the number of characters (bytes) per block including all system control fields.

5. Files specified with device type CARDREADER or CARDPUNCH may be directed to the diskette device as a card substitute device. In this case, the BLOCK CONTAINS clause, if specified, indicates the size of the buffer areas to be used for multisector access. Multisector access improves processing efficiency because multiple records may be read or written with one physical input/output command even though they are not grouped into blocks. The maximum buffer size for multisector access is 1024 bytes.

6. The MIRAM data management processes files specified with ORGANIZATION IS SEQUENTIAL and assigned to disk, or with ORGANIZATION IS RELATIVE, or INDEXED; in MIRAM the concept of grouping logical records does not apply. For these files, the BLOCK CONTAINS clause specifies the size of the buffer areas. Larger buffers allow multiple records to be read or written with one physical access.

7. If the BLOCK CONTAINS clause is not specified, BLOCK CONTAINS 1 RECORD is assumed.

Tables 5-2, 5-3, and 5-4 show the relationship between block size and record size.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

Table 5-2. Block Size Calculations for Tape, Card Reader, Card Punch, and Printer Files

Field	Device Type and Mode												
	Tape						Card Reader	Card Punch			Printer		
	F	V	U	FC	VC	UC	F	F	V	U	FC	VC	UC
PC (Printer Control)	0	0	0	1	1	1	0	0	0	0	1	1	1
BH (Block Header)	0	4*	0	0	4	0	0	0	0	0	0	4	0
RH (Record Header)	0	4	0	0	4	0	0	0	4	0	0	4	0
Multiple Records per Block Permitted (determines blocking factor)	Y	Y	N	Y	Y	N	Y**	Y**	Y**	Y**	N	N	N

BLOCK SIZE (bytes) = (((01 RECORD SIZE) + RH + PC) * BLOCKING FACTOR) + BH

Example:

Assume a tape file with variable mode, 8 records per block, and a maximum 01 record size of 230 bytes.

BLOCK SIZE = (((230) + 4 + 0) * 8) + 4 = 1876

*0 if CODE-SET is STANDARD-0 or STANDARD-1

**See 5.3.1.1, rule 5.

LEGEND:

Y = Yes

N = No

Table 5—3. Block Size Calculations for Mass Storage SAM and ISAM Files

Field	File Organization and Mode					
	SAM				ISAM*	
	F	V	FC	VC	F	V
PC (Printer Control)	0	0	1	1	0	0
BH (Block Header)	0	4	0	4	2	2
RH (Record Header)	0	4	0	4	0	2
LF (Link Field)	0	0	0	0	5	5

BLOCK SIZE (bytes) = (((01 record size) + PC + RH + LF) * BLOCKING FACTOR)+BH

Example:

> Assume a SAM file with VC mode, ORGANIZATION SAM, BLOCK 10 RECORDS, and a maximum 01 record size of 200 bytes.
>
> BLOCK SIZE = (((200) + 1 + 4 + 0) * 10) + 4 = 2054 bytes

*For ISAM files, the minimum block size is 256 bytes.

LEGEND:

Y = Yes

N = No

Table 5—4. Buffer Size Calculations for Mass Storage Sequential, Relative, and Indexed Files

Field	File Organization and Mode							
	Sequential (Mass Storage)				Relative		Indexed	
	F	V	FC	VC	F	V	F	V
PC (Printer Control)	0	0	1	1	0	0	0	0
RH (Record Header)	0	4	0	4	0	4	0	4
RCB (Record Control Byte)	1**	0	1**	0*	1	0*	1	0*

BUFFER SIZE (as a number of 256-byte sectors) = (((((01 RECORD SIZE) + RH + PC + RCB) *BLOCKING FACTOR) + 255) /256) + 1 where the / operator is an integer divide.

Example:

Assume a relative file with F mode, BLOCK 5 RECORDS, and the record size of 300 bytes.

BUFFER SIZE = (((((300) + 0 + 0 + 1) *5) + 255) /256) + 1 = 7 sectors of 256 bytes each.

NOTE:

Relative and indexed files always use the MIRAM record control byte feature. If a language processor other than COBOL creates an IRAM or MIRAM file without the record control byte feature, and no DELETE statement is issued for these files, they can be processed by a COBOL program.

*For files with V or VC mode, the record control byte is contained in the record header.

**For compatibility, COBOL assumes the RCB to be present and allows for it in the block size. However, sequential files created by COBOL do not have a RCB.

LEGEND:

Y = Yes

N = No

5.3.1.2. RECORD CONTAINS Clause

Function:

The RECORD CONTAINS clause specifies the size of data records.

Format:

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

Rules:

1. Since the size of each data record is defined within its respective record description entry, the RECORD CONTAINS clause is optional.

2. Integer-2 may not be used by itself unless the size of each data record in the file is the same. In this case, integer-2 represents the exact number of characters in the data record.

   Example:

   RECORD CONTAINS 80 CHARACTERS

3. 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.

   Example:

   RECORD CONTAINS 115 TO 165 CHARACTERS.

   No record in the file is shorter than 115 characters nor longer than 165 characters. However, if "115 TO" were deleted, each record would be exactly 165 characters long.

4. The size is specified in terms of the number of character positions (bytes) required to store the logical record, regardless of the types of characters used to represent the items within the logical record.

5. The length of a data record in the file section may not exceed 524,287 bytes.

6. The record size specified in a COBOL program, whether specified in a RECORD CONTAINS clause or in the record description clause (the size of the 01 record), refers only to the logical data part of the record and not to any OS/3 control fields appended to the record. The BLOCK CONTAINS clause shows when control fields are present and how big they are. For more detailed information, refer to tables 5–2, 5–3, and 5–4 of this section.

NOTE:

When the term "record size" is used in places other than a COBOL program, such as in a data management manual or a VTOC print, the record size includes the control fields.

5.3.1.3. LABEL RECORDS Clause

Function:

The LABEL RECORDS clause specifies whether labels are present. If labels are present, this clause also identifies the label.

Format:

```
LABEL {RECORDS ARE} { OMITTED                            }
      {RECORD  IS } { STANDARD                           }
                    { [data-name-1 [,data-name-2] ...]   }
```

Rules:

1. This clause is required in every file description entry.

2. OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned. OMITTED must be specified for files assigned to CARDREADER, CARDPUNCH, and PRINTER.

3. STANDARD specifies that standard system labels exist for the file or the device to which the file is assigned and the labels conform to the standard system label specifications. STANDARD must be specified for files assigned to mass storage devices.

4. Data-names may be specified for sequential tape files only. Data-names specify that both standard system labels and standard user labels exist for the file or the device to which the file is assigned. Standard user labels must conform to system specifications. Refer to the OS/3 data management user guide.

   If LABEL RECORDS STANDARD is specified for the tape file, standard user labels may also be present. However, standard user labels should not be checked on input files or written on output files.

5. Data-names are names of standard user label records and must have record descriptions subordinate to the associated file description.

6. References to data-names specified in this clause, or to items subordinate to these data-names, must appear within USE LABEL procedures. (See 6.6.41, the USE statement.)

5.3.1.4. VALUE OF Clause

Function:

The VALUE OF clause specifies the value of an item in the standard system file label record associated with a file. This clause is for documentation only.

Format:

```
VALUE OF ⎧FILE-ID  IS ⎧data-name-1⎫ ⎡,PASSWORD IS ⎧data-name-2⎫⎤⎫
         ⎪            ⎩literal-1  ⎭ ⎣             ⎩literal-2  ⎭⎦⎪
         ⎨                                                      ⎬
         ⎪PASSWORD IS ⎧data-name-2⎫ ⎡,FILE-ID  IS ⎧data-name-1⎫⎤⎪
         ⎩            ⎩literal-2  ⎭ ⎣             ⎩literal-1  ⎭⎦⎭
```

5.3.1.5. DATA RECORDS Clause

Function:

The DATA RECORDS clause only documents the names of data records in a given file.

Format:

```
DATA ⎧RECORDS ARE⎫ data-name-1 [,data-name-2] ...
     ⎩RECORD IS  ⎭
```

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.

2. 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 and formats and can be listed in any order.

3. 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.

5.3.1.6. LINAGE Clause

Function:

The LINAGE clause specifies the size of a logical page in terms of number of lines. It also specifies 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. (See Figure 5-2.)

Format 1:

```
LINAGE IS {data-name-1} LINES
          {integer-1  }
   [,WITH FOOTING AT {data-name-2}]
   [                 {integer-2  }]
   [,LINES AT TOP {data-name-3}]
   [              {integer-3  }]
   [,LINES AT BOTTOM {data-name-4}]
   [                 {integer-4  }]
```

Format 2:

```
    LINAGE IS SYSTEM LINES
```

Rules:

NOTE:

Rules 1 through 13 apply to format 1 only.

1. The LINAGE clause may be used only for sequential files assigned to devices other than CARDREADER and CARDPUNCH.

2. When the LINAGE clause is specified, the character C must be specified in the mode field for an implementor-name. (See 4.4.1, the ASSIGN clause.)

3. Data-name-1, data-name-2, data-name-3, and data-name-4 must reference elementary unsigned numeric integer data items.

4. The LINAGE clause expresses logical page size as the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrase is not specified, the value for this function is 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. Although the FOOTING value is assumed to be equal to the LINAGE value, when the FOOTING phrase is not specified, there is no FOOTING area.

5. The value of integer-1 or the data item referenced by data-name-1 specifies the number of lines that can be written or spaced on the logical page. The value must be greater than zero and must not exceed 999. That part of the logical page in which these lines can be written or spaced is called the page body.

6. 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.

7. 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.

8. 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.

9. The value of integer-1, integer-3, and integer-4, if specified, is 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, is 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.

10. The values of the data items referenced by data-name-1, data-name-3, and data-name-4, if specified, are used as follows:

    - When an OPEN statement with the OUTPUT phrase is executed for the file, they specify the number of lines that are to comprise each of the indicated sections for the first logical page.

    - When a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, they are used to specify the number of lines that are to comprise each of the indicated sections for the next logical page.

11. 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 is 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 is used to define the footing area for the next logical page.

12. A LINAGE-COUNTER register 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 having a file description entry containing 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 during the execution of a WRITE statement to an associated file as follows:

(1) When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to 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 1. (See 6.6.42, the WRITE statement.)

(4) The value of LINAGE-COUNTER is automatically reset to 1 when the device is repositioned to the first line that can be written on for each of the succeeding logical pages. (See 6.6.42, the WRITE statement.)

d. The value of LINAGE-COUNTER is automatically set to 1 at the time an OPEN statement is executed for the associated file.

13. Each logical page is contiguous to the next with no additional spacing provided.

NOTE:

Rules 14 through 19 apply to format 2 only.

14. Format 2 of the LINAGE clause may be used only with files assigned to PRINTER. It may not be used with printer-destined files assigned to other devices.

15. Format 2 of the LINAGE clause specifies a logical page in which the first line of the page is defined by the home-paper position in the printer file's vertical format buffer. This format of the LINAGE clause also allows detection of an end-of-page condition based upon the overflow line position in the printer file's vertical format buffer. See the job control programmer reference, UP-8217 (current version), and data management user guide, UP-8068 (current version), for information regarding vertical format buffer specification and page overflow reporting.

16. The size of the logical page is undefined (no upper limit). The end of a logical page does not occur until a WRITE statement with an ADVANCING PAGE phrase is executed.

17. The footing area comprises the area of the logical page beginning with the line on which the operating system reports that the overflow line position in the vertical format buffer has been crossed and ending at the end of the logical page.

18. Top and bottom margins are not part of the logical page processed by the COBOL program. However, top and bottom margins may be created on the physical page by executing a WRITE statement with an ADVANCING PAGE phrase. This positions the form to the next home-paper position.

19. A LINAGE-COUNTER register 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 having a file description entry containing a LINAGE clause.

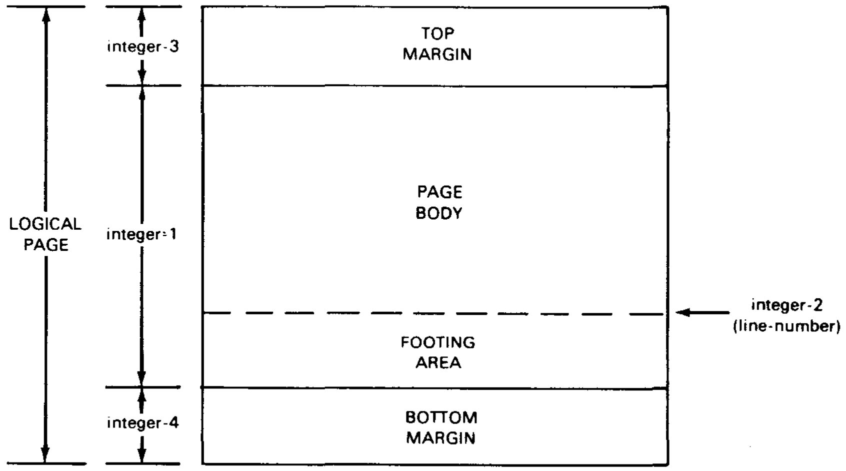
b. LINAGE-COUNTER may be referenced, but may not be modified, by procedure division statements. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.

c. LINAGE-COUNTER is automatically modified during the execution of a WRITE statement to an associated file as follows:

- When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to 1.
- When the ADVANCING identifier-2 or integer phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer or the value of the data item referenced by identifier-2.
- When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value 1. (See 6.6.42, the WRITE statement.)

  When you specify the ADVANCING mnemonic-name phrase of the WRITE statement, the LINAGE-COUNTER is unchanged, but does not accurately reflect the device position in the current page. The LINAGE-COUNTER remains inaccurate until an ADVANCING PAGE phrase is used.

d. The value of LINAGE-COUNTER is automatically set to 1 at the time an OPEN statement is executed for the associated file.



NOTES:

1. Size of logical page in number of lines = the sum of integer-3, integer-1, and integer-4.

2. Size of FOOTING area in number of lines = integer-1 less integer-2 + 1

3. integer-1 must be > zero.
integer-2 must be ≤ integer-1.
integer-3 may be ≥ zero.
integer-4 may be ≥ zero.

Figure 5—2. Logical Page Format for Format 1 LINAGE Clause

5.3.1.7. CODE-SET Clause

Function:

The CODE-SET clause specifies the character code set used to represent data on the external media.

Format:

```
CODE-SET IS alphabet-name
```

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 sequential tape files or files assigned to a CARDREADER or CARDPUNCH.

4. 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 EBCDIC character codes. This code conversion occurs during the execution of an input or output operation. (See the SPECIAL-NAMES paragraph.)

5. If the CODE-SET clause is not specified, the native character code set is assumed for data on the external media.

6. For sequential tape files, if the alphabet-name is associated with STANDARD-1 in the SPECIAL-NAMES paragraph, the tape must conform to the standards set forth in *American National Standard Magnetic Tape Labels for Information Interchange, X3.27–1969,* at the level supported by SAM data management. Fixed, variable, and undefined record formats are permitted. However, the compiler assumes a buffer offset of zero. For an explanation of buffer offset and ASCII tape file formats, see the Basic data management user guide, UP-8068 (current version).

7. If the alphabet-name is associated with STANDARD-0, it is treated the same as STANDARD-1.

5.3.2. Sort/Merge File Description

The sort/merge file description furnishes information concerning the physical structure, identification, and record names of the file to be sorted or merged. Sample sort file description entries are given in Figure 5-3.

Format:

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

Rules:

1. The level indicator SD identifies the beginning of the sort/merge file description and must precede the file-name.

2. The clauses that follow the name of the file are optional and their order of appearance is optional.

3. One or more record description entries must follow the sort/merge file description entry; however, no input/output statements can be executed for this file.

4. The RECORD CONTAINS and DATA RECORDS clauses are described in 5.3.1.2 and 5.3.1.5, respectively.

```
Seq.
No.
          A   B        Text
1         8   12
-----------------------------------------------------------------------------
Ø4ØØ1Ø    SD  SORT-FILE
Ø4ØØ2Ø        RECORD CONTAINS 50 TO 100 CHARACTERS
Ø4ØØ3Ø        DATA RECORD IS SORT-RECORD.
Ø4ØØ4Ø    Ø1  SORT-RECORD.
Ø4ØØ5Ø            Ø2 ACCOUNT-NUMBER           PICTURE 9(8).
Ø4ØØ6Ø            Ø2 NUMBER-OF-CUSTOMERS, USAGE IS COMPUTATIONAL,
Ø4ØØ7Ø                                        PICTURE S9(4).
Ø4ØØ8Ø            Ø2 CUSTOMER-DESCRIPTION, OCCURS 4 TO 9 TIMES DEPENDING
Ø4ØØ9Ø                 ON NUMBER-OF-CUSTOMERS, PICTURE X(1Ø).
```

Figure 5—3. Sample Sort File Description Entries

5.3.3. Data Description

Function:

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

Format 1:

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

     [;REDEFINES data-name-2]

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

     [;[USAGE IS]  / COMPUTATIONAL      \   ]
     [            |  COMP                |  ]
     [            | [COMPUTATIONAL-1]    |  ]
     [            | |COMP-1         |    |  ]
     [            | |COMPUTATIONAL-2|    |  ]
     [            | |COMP-2         |    |  ]
     [           <  |COMPUTATIONAL-3|     > ]
     [            | |COMP-3         |    |  ]
     [            | |COMPUTATIONAL-4|    |  ]
     [            | [COMP-4         ]    |  ]
     [            |  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
                  {VALUES ARE}
   [{THROUGH} literal-2]
   [{THRU   }          ]
   [,literal-3 [{THROUGH} literal-4]]... .
   [           [{THRU   }          ]]
```

Rules:

1. There are three formats for data description entries:

   - Format 1 is used for record description entries in the file, working-storage, and linkage sections and for data item description entries in the working-storage and linkage sections.

   - Format 2 is used to assign alternative names to existing data items or groups of items. (See 5.3.3.12.)

   - Format 3 is used to assign a name to the values that an associated conditional variable may possess during object program execution. (See 5.3.3.11, format 2.)

2. In Level 1, the level-number in format 1 may be any number from 01 through 10 or 77. In Level 2, the level-number in format 1 may be any number from 01 through 49 or 77.

3. 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.

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

5. The words THRU and THROUGH are equivalent.

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

7. 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 that contains a level-number except the following:

   - Another condition-name
   - A level 66 item
   - A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE other than USAGE IS DISPLAY
   - An index data item (see the USAGE IS INDEX clause)

Table 5-5 summarizes the functions of the clauses used in data description entries. The formats and functions of these clauses are described in detail in the paragraphs that follow.

Table 5—5. Data Description Entry Clauses

Clause	Function
data-name or FILLER	Specifies the name of the data being described
REDEFINES	Allows the programmer to give an alternate description of an area of computer storage
PICTURE	Indicates the size, class (alphabetic, numeric, or alphanumeric), and the editing requirements for an elementary data item
USAGE	Specifies the manner in which the data is stored in main storage
SIGN	Specifies the position and mode of representation of the operational sign for numeric data
OCCURS	Indicates the number of elements contained in a table
SYNCHRONIZED	Specifies the alignment of an elementary item on a natural boundary of the computer memory
JUSTIFIED	Specifies that nonnumeric data is to be right-justified in a nonnumeric field
BLANK WHEN ZERO	Specifies that an item is to be set to blanks whenever its value is zero
VALUE	Defines the initial value of a working storage item or a value or range of values associated with a condition-name
RENAMES	Permits alternate, possibly overlapping, groupings of elementary items

5.3.3.1. Level-Number

Function:

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

Format:

```
level-number
```

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 through 10 in Level 1; 01–49, 66, or 88 in Level 2.

3. 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. The level-number 01 identifies the first entry in each record description.

5. Special level-numbers are assigned to certain entries where there is no real concept of level:

   - Level-number 77 is assigned to identify noncontiguous data items and can be used only as described by format 1.

   - Level-number 66 is assigned to identify RENAMES entries and can be used only as described in format 2.

   - Level-number 88 is assigned to entries that define condition-names associated with a conditional variable and can be used only as described in format 3.

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

5.3.3.2. Data-Name/FILLER Clause

Function:

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

Format:

$$\left\{\begin{array}{l}\text{data-name}\\ \underline{\text{FILLER}}\end{array}\right\}$$

Rules:

1. In the file, working-storage, 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.

2. 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.

5.3.3.3. REDEFINES Clause

Function:

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

Format:

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

NOTE:

Level-number, data-name-1, and the semicolon are shown in the format to improve clarity. Level-number and data-name-1 are not part of the REDEFINES clause.

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, or the communication section.

4. 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.

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

6. 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. The REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

7. 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 that contains a REDEFINES clause. In Level 2, data-name-2 may be subordinate to an entry that 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.

8. 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.

   Example:

```
Ø2 A.
   Ø4 A1 PICTURE X(3).
   Ø4 A2 PICTURE 99V99.
Ø2 B REDEFINES A.
   Ø4 B1 PICTURE 9.
   Ø4 B2 PICTURE A(4).
   Ø4 B3 PICTURE XX.
Ø2 C REDEFINES A PICTURE 9(4)V9(3).
```

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

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

5.3.3.4. PICTURE Clause

Function:

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

See Appendix K for a tutorial description of the PICTURE clause and for additional examples.

Format:

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

Rules:

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

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

3. PIC is an abbreviation for PICTURE.

4. The PICTURE clause must be specified for every elementary item except an index data item and an internal floating-point item for which use of this clause is prohibited.

5. 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.

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

   The five categories of data items are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the class and the category are synonymous. The alphanumeric class includes the categories of alphanumeric (without editing), alphanumeric edited, and numeric edited.

   Every elementary item except for an index data item belongs to one of the classes and 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 relationship of the class and category for elementary and group data items is shown in Table 5-6.

Table 5—6. Class and Category of Elementary and Group Data Items

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric edited Alphanumeric edited Alphanumeric
Group	Alphanumeric	Alphabetic Numeric Numeric edited Alphanumeric edited Alphanumeric

7. The maximum size of an elementary item is defined as follows:

Alphabetic	4092 bytes
Numeric	Size in bytes is determined by the USAGE and SIGN clauses (see 5.3.3.5 and 5.3.3.6).
Numeric edited	120 bytes
Alphanumeric edited	120 bytes
Alphanumeric	4092 bytes

8. To define an item as alphabetic:

   - Its PICTURE character-string can only contain the symbols A and B.

   - Its contents when represented in standard data format must be any combination of the 26 letters in the alphabet and the space.

9. To define an item as numeric:

   - Fixed-Point Items

     There are three types of fixed-point items: external decimal, binary, and internal decimal. (See 5.3.3.5.)

The PICTURE character-string of a fixed-point item 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.

If unsigned, the contents of a data item represented in standard data format must be a combination of the numerals 0 through 9; if signed, the item may also contain a +, –, or other representation of an operational sign. (See 5.3.3.6.)

- Floating-Point Items

The floating-point items define data having a potential range of value too great for fixed-point presentation. The magnitude of the number represented by a floating-point item must be greater than 5.4×10^{-79} but must not exceed 0.72×10^{76}.

There are two types of floating-point items: internal floating-point and external floating-point.

No PICTURE clause may be associated with an internal floating-point item. The USAGE clause for an internal floating-point item is COMPUTATIONAL-1 or COMPUTATIONAL-2. (See 5.3.3.5.)

An external floating-point item has the USAGE of DISPLAY and a PICTURE character-string in the following format:

```
{±} mantissa E {±} exponent
```

where:

{±}

A plus indicates that the data is positive if preceded by a plus or negative if preceded by a minus.

A minus indicates that the data is positive if preceded by a space character or negative if preceded by a minus.

The plus sign, the space character, or the minus sign occupies one byte of main storage.

mantissa

Is represented by the symbols: 9 . or V. Each 9 represents a digit position and occupies a byte of main storage. From one to sixteen 9's may be present in the mantissa string.

The period represents an actual decimal point and occupies a byte of storage. The V represents an assumed decimal point, which does not occupy any main storage.

One actual or assumed decimal point must be present in the mantissa as a leading, embedded, or trailing symbol.

E

Indicates the exponent. It occupies a byte of main storage.

exponent

Specifies a power of 10 that is used as a multiplier. It is represented by two consecutive 9's. Each 9 occupies a byte of main storage.

No VALUE clause may be associated with an external floating-point item.

10. To define an item as alphanumeric:

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

    - Its contents, when represented in standard data format, are allowable characters in the computer character set.

11. To define an item as alphanumeric edited:

    - Its PICTURE character-string is restricted to certain combinations of the following symbols:

      A X 9 B 0 /

      As a minimum, it must contain:

      - at least one B, 0, or / and one X; or

      - at least one 0 or / and one A.

    - The contents, when represented in standard data format, are allowable characters in the computer character set.

12. To define an item as numeric edited:

    - Its PICTURE character-string is restricted to certain combinations of the following symbols:

      B / P V Z 0 9 , . * + – CR DB or currency symbol

      - The allowable combinations are determined from the order of precedence of symbols and the editing rules.

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

      - The character-string must contain at least one of the following symbols:

        0 B / Z * + , . – CR DB or currency symbol

    - 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.

13. An integer that is enclosed in parentheses following the symbols:

    A , X 9 P Z * B / 0 + – or currency symbol

    indicates the number of consecutive occurrences of the symbol. The following symbols may appear only once in a given PICTURE:

    S V . CR DB E

14. The functions of the symbols used in a PICTURE character-string other than floating-point to describe an elementary item are explained as follows:

Symbol Description

A Represents a character position that contains only a letter of the alphabet or a space

B Represents a character position into which the space character is to be inserted

P Indicates an assumed decimal scaling position and specifies the location of an assumed decimal point when the point is not within the number that appears in the data item. The P is not counted in the size of the data item, but is counted in determining the maximum number of digit positions (18) in numeric edited items or numeric items. The P can appear only to the left or right as a continuous string of P's within a PICTURE description. Since the P implies an assumed decimal point (to the left of the P's if P's are leftmost PICTURE characters, and to the right if the 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 Indicates the presence of an operational sign but not its representation nor, necessarily, its position. It must be written as the leftmost character in the PICTURE and 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 that specifies the optional SEPARATE CHARACTER phrase. (See the SIGN clause.)

V Indicates 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 Represents a character position that contains any allowable character from the computer character set

Z Represents a leading numeric character position. When that position contains a 0, the 0 is replaced by a space character. Each Z is counted in the size of the item.

9 Represents a character position that contains a numeral and is counted in the size of the item

0 (zero) Represents a character position into which the numeral 0 is to be inserted. The 0 is counted in the size of the item.

/ (stroke) Represents a character position into which the stroke character is to be inserted. The / is counted in the size of the item.

Symbol Description

, (comma) Represents a character position into which a comma is to 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.

. (period) Is an editing symbol that represents the decimal point for alignment purposes and, in addition, represents a character position into which a decimal point is to be inserted. A period is counted in the size of the item. 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 period must not be the last character in the PICTURE character-string.

+, -,
CR, DB Are used as editing sign control symbols. They represent the character position into which the editing sign control symbol is placed. These 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.

* Represents a leading numeric character position into which an asterisk is placed when that position contains a zero. Each asterisk is counted in the size of the item.

cs 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.

15. There are two general methods of performing editing in the PICTURE clause; insertion or suppression and replacement.

    The four types of insertion editing are:

    - Simple insertion
    - Special insertion
    - Fixed insertion
    - Floating insertion

    The two types of suppression and replacement editing are:

    - Zero suppression and replacement with spaces
    - Zero suppression and replacement with asterisks

16. The type of editing that may be performed upon an item is dependent upon the category to which the item belongs. (See Table 5-7.)

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

18. Insertion editing is described as follows:

    - The simple insertion editing characters are:

      , B 0 /

      The insertion characters are counted in the size of the item and represent the position in the item into which the character is to be inserted.

    - The special insertion character is the . (period). When used as an actual decimal point, the insertion character is counted in the size of the item. In addition, the period is used to represent the decimal point for alignment purposes. 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 insertion character appears in the edited item in the same position as shown in the character-string.

Table 5—7. Type of Editing Permissible for Each Data Category

Data 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 rule 17

    - The fixed insertion editing characters are the currency symbol (cs) and the editing sign control symbols:

      + - CR DB

      Only one currency symbol and one editing sign control symbol 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 - 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. The insertion character occupies the same character position in the edited item as it occupies in the PICTURE character-string. Editing sign control symbols produce the results given in Table 5-8, depending upon the value of the data item.

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

Table 5—8. Results Produced by Editing Sign Control Symbols

Editing Symbol in PICTURE Character-String	Result	
	Data Item Positive or Zero	Data Item Negative
+	+	-
-	Space	-
CR	2 spaces	CR
DB	2 spaces	DB

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two floating insertion characters. This string 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. Nonzero numeric data replaces all the characters at or to the right of this limit.

There are 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 (examples 1 and 2). The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character (example 3).

If the insertion characters are only to the left of the decimal point, only a single floating insertion character is placed into the character position immediately preceding either the decimal point or the first nonzero 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 (examples 4, 5, and 6).

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 nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

Examples:

	PICTURE	Data	Edited Result
1.	$$9.99	12‸34	$12.34
2.	$$,$$$.99	1234‸ØØ	$1,234.ØØ
3.	$$$.$$	12‸34	$12.34
4.	$$,$$$.$$	ØØØØ‸ØØ	(all spaces)
5.	++,+++.++	ØØØ1‸ØØ	+1.ØØ
6.	--,---.--	—ØØØØ‸Ø1	—.Ø1

19. In zero suppression editing, the suppression of leading 0's 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 is a space. If the asterisk is used, the replacement character is * (asterisk).

Zero suppression and replacement is indicated in a PICTURE character-string in the following manner. A string of one or more of the allowable symbols (* or Z) is used to represent leading numeric character positions that are to be replaced when the associated character position in the data contains a 0. Any of the simple insertion characters (, B 0 /) embedded in the string of symbols or to the immediate right of this string are part of the string.

The two ways of representing zero suppression in a PICTURE character-string are:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols.
- All numeric character positions in the character string are represented by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading 0 in the data that corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero 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.

Examples:

PICTURE	Data Item	Edited Result
ZZ99.99	ØØØØ‸ØØ	ØØ.ØØ
ZZZZ.99	ØØØØ‸ØØ	.ØØ
ZZZZ.ZZ	ØØØØ‸ØØ	(all spaces)
****.99	ØØØØ‸ØØ	****.ØØ
****.**	ØØØØ‸ØØ	****.**
*,***,***.99BBCR	—2135‸Ø5	****2,135.Ø5 CR

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

21. Table 5-9 shows the order of precedence when using characters as symbols in a character-string.

    At least one of the symbols

    A X Z 9 *

    or at least two of the symbols

    + - cs (currency symbol)

    must be present in a PICTURE character-string.

    Nonfloating insertion symbols + and -, floating insertion symbols Z * + - and cs, and other symbol P appear twice in the PICTURE character precedence chart, Table 5-9. 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.

    The PICTURE character precedence chart (Table 5–9) summarizes the preceding rules and provides a quick check on the legal order of PICTURE symbols. For example, the chart shows that picture string $+99 is illegal because the intersection of the column (nonfloating insertion symbol cs) and row (nonfloating insertion symbol {±}Ⓛ) contains no X. This summarizes rule 18.

Table 5—9. PICTURE Character Precedence Chart (Part 1 of 2)

Second Symbol		First Symbol																				
		Nonfloating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ −} Ⓛ	{+ −} Ⓡ	{CR DB}	cs	{Z *} Ⓛ	{Z *} Ⓡ	{+ −} Ⓛ	{+ −} Ⓡ	cs Ⓛ	cs Ⓡ	9	{A X}	S	V	P Ⓛ	P Ⓡ
Nonfloating Insertion Symbols	B	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	0	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	/	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	,	X	X	X	X	X	X			X	X	X	X	X	X	X	X			X		X
	.	X	X	X	X		X			X	X		X		X		X					
	{+ −} Ⓛ								–													
	{+ −} Ⓡ	X	X	X	X	X				X	X	X			X	X	X			X	X	X
	{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

LEGEND:

X Indicates that symbol at top of column may precede symbol at left of row.

{ } Indicates that symbols are mutually exclusive.

cs Indicates a currency symbol.

Ⓛ Indicates the occurrence of the symbol to the left of the decimal point.

Ⓡ Indicates the occurrence of the symbol to the right of the decimal point.

NOTE:

External floating-point PICTURE characters are not included in this chart.

5.3.3.5. USAGE Clause

Function:

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

Format 1:

```
[USAGE IS]  COMPUTATIONAL
            COMP
            COMPUTATIONAL-1
            COMP-1
            COMPUTATIONAL-2
            COMP-2
            COMPUTATIONAL-3
            COMP-3
            COMPUTATIONAL-4
            COMP-4
            DISPLAY
```

Format 2:

```
[USAGE IS] INDEX
```

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.

NOTE:

Rules 2 through 13 apply to format 1 only.

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. 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. The USAGE IS DISPLAY specifies that the item is stored in character form, one character per byte; it is used for alphabetic, alphanumeric, alphanumeric edited, numeric edited, external decimal, and external floating-point items.

5. COMP, COMP-1, COMP-2, COMP-3, and COMP-4 are abbreviations for COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, and COMPUTATIONAL-4, respectively.

6. COMPUTATIONAL and COMPUTATIONAL-4 are synonymous.

7. A COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, or COMPUTATIONAL-4 item is capable of representing a value to be used in computations and must be numeric. If the USAGE clause of a group item is specified with any of these options, only the elementary items within the group have the specified USAGE; the group item itself cannot be used in computations.

8. The PICTURE character-string of a COMPUTATIONAL, COMPUTATIONAL-3, or COMPUTATIONAL-4 item can contain only 9's, the operational sign character S, the implied decimal point character V, or one or more P'S. (See the PICTURE clause.)

9. No VALUE clause may be specified for items with descriptions that include the USAGE IS INDEX clause.

10. No PICTURE clause may be specified for a COMPUTATIONAL-1 or COMPUTATIONAL-2 item.

11. COMPUTATIONAL or COMPUTATIONAL-4 specifies that the value of a data item is to be stored in binary format.

Example:

```
Description              Value        Internal Representation

                                      S
PICTURE S9999            +6879        0001   1010   1101   1111
COMPUTATIONAL                                       \_________/
                                                       1 byte

                                      S
PICTURE S9999            –6879        1110   0101   0010   0001
                                                    \_________/
                                                       1 byte
```

The number of digits (9 characters) specified in the PICTURE character string determines the size, in bytes, of a COMPUTATIONAL or COMPUTATIONAL-4 item.

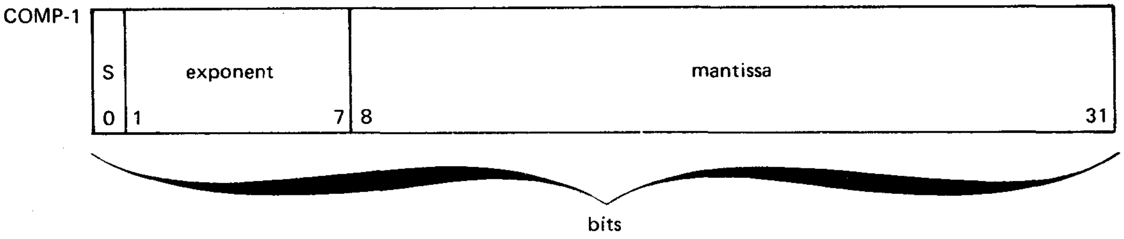
Number of Digits	Size, in Bytes
1 to 4	2
5 to 9	4
10 to 18	8

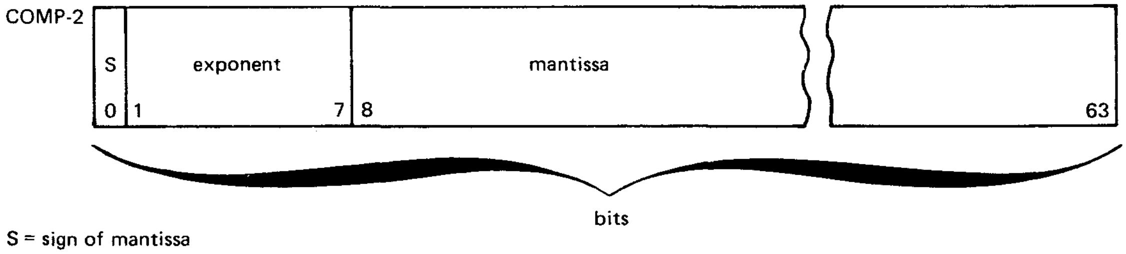
NOTE:

S indicates a sign bit.

12. COMPUTATIONAL-1 specifies that the value of a data item is to be stored in single precision floating-point format. COMPUTATIONAL-2 specifies that the value of a data item is to be stored in double precision floating-point format.

Examples:





S = sign of mantissa

13. COMPUTATIONAL-3 specifies that the value of a data item is to be stored in internal decimal format (packed decimal format).

Example:

Description	Value	Internal Representation
9999	6879	Ø6879F
S9999	+6879	Ø6879C
S9999	-6879	Ø6879D

The number of digits (9 characters) in the PICTURE character string determines the size, in bytes, of a COMPUTATIONAL-3 data item.

Number of Digits	Size, in Bytes
1	1
2 to 3	2
4 to 5	3
6 to 7	4
8 to 9	5
10 to 11	6
12 to 13	7
14 to 15	8
16 to 17	9
18	10

NOTE:

Rules 14 through 18 apply to format 2 only.

14. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value that must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. 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.

15. An index data item defines a data item of eight characters in length and contains the binary representation of the character displacement of a table element occurrence within a table. An index data item does not require synchronization and is not aligned to any machine boundary other than a character or byte boundary.

16. An index data item is a save area where the value of an index-name can be placed. Do not use it as a subscript or as an index to refer to an individual element within a table. Refer directly to an index data item only in a SEARCH or SET statement, a relational condition, the USING phrase of a procedure division header, or the USING phrase of a CALL statement.

17. An index data item can be part of a group that is referred to in a MOVE or input/output statement, in which case no conversion will take place.

18. 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.

5.3.3.6. SIGN Clause

Function:

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

Format:

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

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 except for floating-point 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.

5. The optional SIGN clause 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.

6. A numeric data description entry whose PICTURE contains the character S, but to which no optional SIGN clause applies, has an operational sign. The sign is considered to be TRAILING, without the SEPARATE CHARACTER option.

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

   - The operational sign is presumed to be associated with the leading or, respectively, trailing digit position of the elementary numeric data item.

   - 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).

   - The valid signs for numeric data items occur in the zone portion of LEADING or TRAILING character position. The hexadecimal value C represents a positive sign, and the value D represents a negative sign. The hexadecimal value F is considered as a positive sign if the PICTURE character-string contains an S, and considered unsigned if the PICTURE character-string does not contain an S.

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

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

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

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

9. 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 computation or comparisons, conversion is automatic.

5.3.3.7. OCCURS Clause

Function:

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indexes.

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] ...]
```

Rules:

1. The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. The data-name, which is the subject of the data description 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 2.6.2, subscripting; 2.6.3, indexing; 2.6.4, identifier.)

2. The KEY IS phrase indicates 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 6.4.1.1.1 and 6.4.1.1.2.) The data-names are listed in their descending order of significance.

3. 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.

4. The number of occurrences of the subject entry is defined as follows:

   - In format 1, the value of integer-2 represents the exact number of occurrences.

   - 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.

5. 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 rule 11 of the VALUE clause.)

6. The OCCURS clause cannot be specified in a data description entry that:

   - has a 01, 66, 77, or an 88 level-number; or
   - 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.

7. The length of a table element (i.e., the size of the item containing an OCCURS clause) may not exceed 32,767 bytes. The maximum number of occurrences of a table element (i.e., the value of integer-2) may not exceed 65,535.

   Example 1:

   ```
   Ø2 A PIC X(2) OCCURS 65ØØØ TIMES.
   ```

   This entry is valid because the length of table element A is two bytes and the number of occurrences of A does not exceed 65,535.

   Example 2:

   ```
   Ø2 X OCCURS 2 TIMES.
   Ø4 Y PIC X(1ØØØ) OCCURS 4Ø TIMES.
   ```

   These entries are incorrect because the length of table element X is 40,000 bytes, which exceeds the maximum length permitted for a table element.

8. In format 1, the value of integer-2 must be greater than 0 and less than 65,536.

   In format 2, the value of integer-1 may range from 0 through 65,534, and the value of integer-2 must be greater than the value of integer-1 and less than 65,536.

9. The data description of data-name-1 must describe a positive integer.

10. Data-name-1, data-name-2, data-name-3, . . . may be qualified.

11. 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.

12. Data-name-3, etc, must be the name of an entry subordinate to the group item that is the subject of this entry.

13. Index-name-1, index-name-2, . . . must be unique words within the program.

14. If data-name-2 is not the subject of this entry, then:

    - all 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;

    - items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause; and

    - 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.

15. A data description entry that contains format 2 of the OCCURS clause may only be followed, within that record description, by data description entries subordinate to it.

16. When a group item with a subordinate 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 is used in the operation.

17. In format 2, if the data item defined by data-name-1 appears in the same record as the table it controls, it must appear before that table.

Example 1:

```
01 EMPLOYEE-TABLE.
   02 DEPARTMENT, OCCURS 5 TIMES.
      03 EMPLOYEE, OCCURS 50 TIMES, PICTURE X(20).
```

This example defines a table containing 50 entries for employees, grouped into five departments. The picture for each entry is X(20). This gives a total of 5 x 50 = 250 entries.

Example 2:

```
01 DATA-RECORD.
   02 FIXED-PORTION.
      03 MAIN-INFO                  PICTURE X(25).
      03 NR-OF-TRAILERS PICTURE S99 COMPUTATIONAL.
   02 VARIABLE-PORTION OCCURS 1 TO 10 TIMES
    DEPENDING ON NR-OF-TRAILERS.
    03 TRAILER                      PICTURE X(15).
    03 TRAILER-2                    PICTURE X(5).
```

In this example, format 2 of the OCCURS clause is used to describe variable-length records. The fixed portion of 27 bytes always appears in each record. The presence of trailer items in a record is dependent on the contents of the data item NR-OF-TRAILERS. When NR-OF-TRAILERS contains a value of 0, the record length is 27 bytes; when the value is 1, record length is 47 bytes; when the value is 2, record length is 67; etc.

Example 3:

```
01 TABLE-A.
   02  ITEM-A PICTURE 99, OCCURS 5 TIMES
              INDEXED BY INDX-1.
   01  TABLE-B.
       02  ITEM-B PICTURE 99, OCCURS 5 TIMES.
```

In a program containing these descriptions, INDX-1 cannot be used to refer to an element in TABLE-B.

5.3.3.8. SYNCHRONIZED Clause

Function:

The SYNCHRONIZED clause specifies that an elementary item is to be aligned on the proper boundary of the computer main storage for efficiency in using the elementary item.

Format:

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

Rules:

1. The proper alignment boundary for the various types of elementary item formats as specified in the USAGE clause is given in Table 5-10.

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

3. SYNC is an abbreviation for SYNCHRONIZED.

Table 5—10. Alignment Boundaries for Various Type Elementary Items

Item Format	Item Length	Alignment Boundary
{ COMP [COMP-4] }	S9 through S9(4) S9(5) through S9(18)	Half word Full word
[COMP-1]		Full word
[COMP-2]		Double word
{ INDEX [COMP-3] DISPLAY }		Bytes

4. The LEFT and RIGHT options are treated as comments.

5. Regardless of whether the SYNCHRONIZED clause is used, all 01 level entries are aligned by the compiler on double-word boundaries.

6. Slack bytes (unused character positions) are inserted immediately preceding the elementary item to be synchronized. Although the length of the elementary item is not affected by the SYNCHRONIZED clause, the inserted slack bytes are included in:

   - the size of any group items to which the elementary item belongs; and

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

Example:

```
Ø1 REC.
   Ø2 A.
      Ø3 M.
         Ø4 S PIC X.
         Ø4 T PIC S9 COMP
      Ø3 N USAGE COMP-2.
   Ø2 B USAGE COMP-1.
```

If the SYNCHRONIZED clause is not specified, the elementary items appear in the computer main storage as follows:

S	T	N	B
0	1 2	3 10	11 14

If the SYNCHRONIZED clause is specified for item T, one slack byte is inserted preceding item T to align T on a half-word boundary as follows:

S	SLACK	T	N	B
0	1	2 3	4 11	12 15

The inserted slack byte does not affect the size of the synchronized item T, but is included in the length of the group item M.

If the SYNCHRONIZED clause is specified for elementary items T, N, and B, slack bytes are inserted as follows:

S	SLACK	T	SLACK	N	B
0	1	2 3	4 7	8 15	16 19

T is a COMP item and is aligned on a half-word boundary by the insertion of one slack byte. N is a COMP-2 item that requires alignment on a double-word boundary that is provided by the insertion of four slack bytes. B is on a double-word boundary and requires no slack bytes. The size of group item, then, includes the five inserted slack bytes.

The algorithm used by the compiler to determine the insertion of slack bytes is explained as follows:

- As each item to be synchronized is encountered, the total number of bytes occupied by all the elementary items up to but not including this one is added to the total number of slack bytes already inserted.

- This total divided by x, where:

x	Item	Length
2	COMP	1 to 4 digits
4	COMP	5 to 18 digits
4	COMP-1	
8	COMP-2	

- If there is no remainder for the division, no slack bytes are necessary. If there is a remainder, the number of slack bytes required is equal to x minus the remainder.

For the example, the algorithm would be used as follows:

- For the first synchronized item, T, the total number of bytes in the record so far is 1; x for this COMP item is 2; the remainder of the division is 1. Thus, x (2) minus 1 equals 1; therefore, 1 is the number of slack bytes required.

- For N, a COMP-2 item, the storage already occupied is 1 (for S) + 1 (the first slack byte) + 2 (for T), a total of 4. The value of x to be used is 8, and the remainder of the division is 4; therefore, x (8) minus 4 equals 4, so four slack bytes were inserted in positions 4 through 7 to align N.

- When B is encountered, the total storage already occupied is 16; when this is divided by 4, the value of x for B, there is no remainder. No slack bytes were required.

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

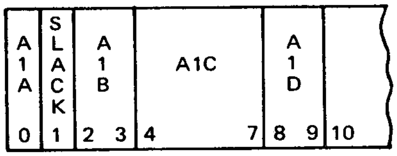
   - each occurrence of the data item is SYNCHRONIZED; and

   - any slack bytes generated for other data items within that same table are generated for each occurrence of those data items.

   Example:

```
Ø1 A.
   Ø2 A1 OCCURS 3 TIMES.
      Ø3 A1A PIC X.
      Ø3 A1B PIC S9 USAGE COMP SYNC.
      Ø3 A1C USAGE COMP-1 SYNC.
      Ø3 A1D PIC S9 USAGE COMP SYNC.
```

   One occurrence would be synchronized as:



If the second occurrence began immediately with byte 10, slack bytes in the second occurrence would have to be as follows because A1C must be aligned on a full-word boundary.

	A1A	SLACK	A1B	SLACK	A1C	A1D	
	10	11	12 13	14 15	16 19	20 21	22

The group cannot have different lengths with each occurrence; therefore, slack bytes are inserted at the end of each occurrence so that each occurrence has the same length and the proper alignment of elementary items. The actual storage use for the example is:

A1A	SLACK	A1B	A1C		A1D	SLACK	A1A	SLACK	A1B	A1C		A1D	SLACK	A1A	SLACK	A1B	A1C		A1D	SLACK
0	1	2 3	4	7	8 9	10 11	12	13	14 15	16	19	20 21	22 23	24	25	26 27	28	31	32 33	34 35

The slack bytes are inserted in positions 10 and 11, in positions 22 and 23, and in positions 34 and 35. The algorithm used is as follows:

- The total number of bytes occupied by the group, including slack bytes, is divided by the largest value of x necessary in the group.

- If there is no remainder, no slack bytes are inserted between groups. Otherwise, the number of slack bytes necessary is equal to x minus the remainder.

For the example given, the process is:

- The total number of bytes occupied in one occurrence of the group is 10 bytes. This is divided by 4, the x value for A1C, a COMP-1 item.

- The remainder of the division is 2; x (4) minus 2 equals 2, so the number of slack bytes necessary for each occurrence is 2.

5.3.3.9. JUSTIFIED Clause

Function:

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

Format:

```
{JUSTIFIED} RIGHT
{JUST     }
```

Rules:

1. The JUSTIFIED clause is used to override standard positioning of data within a receiving alphabetic or alphanumeric data item. Standard positioning for this type of data is left-justified with space fill on the right; when this clause is specified, the data is right-justified and the unused positions are space-filled.

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

3. JUST is an abbreviation for JUSTIFIED.

4. The JUSTIFIED clause has no effect on the initialization of the VALUE clause.

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

   When the sending data item is larger than the receiving data item described with the JUSTIFIED clause, the leftmost characters are truncated. When the receiving data item 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.

   When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. (See 2.5.)

5.3.3.10. BLANK WHEN ZERO Clause

Function:

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

Format:

```
BLANK WHEN ZERO
```

Rules:

1. This clause can be specified only at the elementary item level, and can be used only with a numeric or numeric-edited item. When used with a numeric item, the category of the item is considered numeric-edited.

2. The effect is not necessarily the same as zero suppression editing via the PICTURE clause because the item is affected only when its numeric value is 0.

3. When the BLANK WHEN ZERO clause is used, the item will contain only spaces if the value of the item is zero.

4. The BLANK WHEN ZERO clause and the zero suppression symbol * may not appear in the same entry.

5. The BLANK WHEN ZERO clause has no effect on the initialization of the VALUE clause.

5.3.3.11. VALUE Clause

Function:

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

Format 1:

```
VALUE IS literal
```

Format 2:

```
{VALUE IS   }  literal-1 [{THROUGH} literal-2]
{VALUES ARE }            [{THRU   }          ]

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

Rules:

1. Format 1 is used to define the initial value of a working-storage item. Format 2 is used to specify a value or range of values associated with a condition-name.

2. The words THRU and THROUGH are equivalent.

3. The VALUE clause cannot be stated for any items having variable size. (See 5.3.3.7.)

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

5. All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause and must not have a value that will require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

6. 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:

   - 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.

   - If the literal is floating-point, the data item must be an internal floating-point item.

   - 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 2.5, standard alignment rules.) Editing characters in the PICTURE clause are included in determining the size of the data item (5.3.3.4) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item must be specified in an edited form.

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

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

8. 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.

9. Format 2 can be used only in connection with condition-names. (See 2.6.5, condition-name.) Wherever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

10. Rules governing the use of the VALUE clause differ with the respective sections of the data division:

    - 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.

    - 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 description, the initial value is undefined.

    - 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.

11. The VALUE must not be stated in a data description entry that contains an OCCURS clause or is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries. (See 5.3.3.7.)

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

13. 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.

14. The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, USAGE INDEX, or any form of COMPUTATIONAL.

15. The VALUE clause must not be specified for external floating-point items.

Example 1:

```
Ø2 STATE-RATE PICTURE 9.
   88 TEXAS VALUE 1.
   88 CALIFORNIA VALUE 2.
   88 NEW YORK VALUE 5.
   88 PENNSYLVANIA VALUE 3.
```

STATE-RATE is a conditional-variable; TEXAS, CALIFORNIA, NEW YORK, and PENNSYLVANIA are condition-names. If the statement IF PENNSYLVANIA GO TO NEXT-TEST were to appear in the procedure division, the value of the conditional variable STATE-RATE would be compared to the value 3; this statement would be equivalent to the statement IF STATE-RATE IS EQUAL TO 3 GO TO NEXT-TEST.

Example 2:

```
Ø2 AGE PICTURE 99.
   88 TWENTIES VALUE 2Ø THRU 29.
   88 THIRTIES VALUE 3Ø THRU 39.
```

If the statement IF TWENTIES . . . were to appear in the procedure division, the value of the conditional variable AGE would be tested for not less than 20 and not greater than 29.

5.3.3.12. RENAMES Clause

Function:

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

Format:

$$66\ \text{data-name-1;}\underline{\text{RENAMES}}\ \text{data-name-2}\left[\begin{Bmatrix}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{Bmatrix}\ \text{data-name-3}\right].$$

NOTE:

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

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, 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 5.3.3.7.)

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 data-name-2 through data-name-3, if specified, can be an item whose size is variable as defined in the OCCURS clause.

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

9. 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).

10. 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.

Example:

```
Ø1 INPUT-RECORD.
   Ø2 STATE-TAX-NJ.
      Ø3 PER-CENT-ST       PIC 99.
      Ø3 PERCENT-CNTY      PIC 99.
      Ø3 PERCENT-LOC       PIC 99.
   Ø2 STATE-TAX-PA.
      Ø3 PER-CENT-ST       PIC 99.
      Ø3 PER-CENT-CNTY   PIC 99.
      Ø3 PER-CENT-LOC      PIC 99.
   Ø2 STATE-TAX-DEL.
      Ø3 PER-CENT-ST       PIC 99.
      Ø3 PER-CENT-CNTY     PIC 99.
      Ø3 PER-CENT-LOC      PIC 99.
   66 TAX-NJ RENAMES STATE-TAX-NJ.
   66 TAX-B1-STATES RENAMES STATE-TAX-NJ THRU STATE-TAX-PA.
   66 TAX-DEL-VAL RENAMES STATE-TAX-NJ THRU STATE-TAX-DEL.
```

A reference to TAX-NJ accesses the group item STATE-TAX-NJ, a reference to TAX-B1-STATES accesses the group items STATE-TAX-NJ and STATE-TAX-PA; and a reference to TAX-DEL-VAL accesses the items STATE-TAX-NJ, STATE-TAX-PA, and STATE-TAX-DEL.

5.4. WORKING-STORAGE SECTION

The working-storage section describes records and noncontiguous data items that 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. Sample working-storage section entries are provided in Figure 5–4.

Format:

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

Rules:

1. The working-storage section is composed of the section header followed by data description entries for noncontiguous data items or record description entries.

2. Each work-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.

```
010010  WORKING-STORAGE SECTION.
010020  77   I, COMPUTATIONAL, PICTURE S9(8).
010030  77   J, COMPUTATIONAL, PICTURE S9(8).
010040  77   ADDED-TIME, COMPUTATIONAL-3, PICTURE S9(5)V9(4).
010050  01   DATA-CONVERSION-AREA.
010060       02 BINARY-WORK-AREA.
010070           03 TWO-BYTES.
010080              04 FILLER           PICTURE X  VALUE LOW-VALUE.
010090              04 ONE-BYTE-BINARY PICTURE X.
010100           03 TWO-BYTE-BINARY REDEFINES TWO-BYTES,
010110              USAGE IS COMPUTATIONAL, PICTURE S9(4).
010120       02 CPU-TIME-WORK-AREA.
010130           03 CPU-TIME-IN,    COMPUTATIONAL-3, PICTURE S9(11).
010140           03 CPU-TIME OUT REDEFINES CPU-TIME-IN,
010150             USAGE IS COMPUTATIONAL-3, PICTURE S9(7)V9(4).
```

Figure 5—4. Sample Working-Storage Section Entries

5.4.1. 77-Level Description Entry

Function:

Items and constants in working-storage that 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 that begins with the special level-number, 77.

Format:

```
77 data-name;
    (data description clauses).
```

Rules:

1. The following are required in each data description entry:

   - level-number 77
   - data-name
   - PICTURE clause or USAGE IS INDEX or USAGE COMP-1 or COMP-2 clause

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

3. 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. Each independent entry must have a unique data-name.

5.4.2. Record Description Entry

Function:

Data elements and constants in working storage that bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions.

Format:

```
01 record-name.
     (subordinate data items and clauses)
```

Rules:

1. Each record-name must be unique because it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be qualified.

2. All clauses that are used in record descriptions in the file section (5.3.3) can be used in record descriptions in the working-storage section.

3. The length of a level-01 record may not exceed 524,287 bytes.

5.5. LINKAGE SECTION

The linkage section describes data available through a calling program but is to be referred to in both the calling and called program.

Format:

```
LINKAGE SECTION.
    ┌77-level-description-entry┐  ...
    └record-description-entry  ┘
```

Rules:

1. The linkage section is meaningful if and only if the object program is to function under the control of a CALL statement containing a USING phrase in the calling program.

2. The linkage section consists of a section header followed by data description entries for noncontiguous data items and/or record description entries. (See Figure 5-5.)

3. 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 indexes.

4. 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.

5. Each linkage section record-name and noncontiguous item name must be unique within the called program since it cannot be qualified.

6. 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, or index-names associated with such data-names or subordinate data items may be referenced in the procedure division.

7. The VALUE clause must not be specified in the linkage section except in condition-name entries (level 88).

```
Ø15Ø1Ø  LINKAGE SECTION.
Ø15Ø2Ø  77  TYPE-OF-INPUT                PICTURE X.
Ø15Ø3Ø      88  FIRST-INPUT               VALUE "F".
Ø15Ø4Ø      88  CONTINUATION              VALUE."C".
Ø15Ø5Ø      88  LAST-INPUT                VALUE."L".
Ø15Ø6Ø  77  ERROR-INDICATOR              PICTURE  X.
Ø15Ø7Ø  Ø1  PAST-RECORD.
Ø15Ø8Ø      Ø2  SALES-HISTORY
Ø15Ø9Ø          Ø3  MONTH, OCCURS 12 TIMES, PICTURE S9(7)V99,
Ø151ØØ                USAGE IS COMPUTATIONAL-3.
Ø1511Ø      Ø2  PRODUCT                  PICTURE X(3).
Ø1512Ø      Ø2  THREE-MONTH-AVERAGE      PICTURE S9(7)V99,
Ø1513Ø             USAGE IS COMPUTATIONAL-3.
Ø1514Ø      Ø2  TWELVE-MONTH-AVERAGE     PICTURE S9(7)V99,
Ø1515Ø            USAGE IS COMPUTATIONAL-3.
```

Figure 5—5. Sample Linkage Section Entries

5.5.1. 77-Level Description Entry

Function:

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 that begins with the special level-number 77.

Format:

```
77 data-name;
    (data description clauses).
```

Rules:

1. The following are required in each data description entry:

   - level-number 77
   - data-name
   - PICTURE clause or USAGE IS INDEX clause

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

5.5.2. Record Description Entry

Function:

Data elements in the linkage section that bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions.

Format:

```
Ø1 record-name.
    (subordinate data items and clauses)
```

Rules:

1. All clauses that are used in record descriptions in the file section (5.3.3) can be used in record descriptions in the linkage section.

2. Record description entries in the linkage section provide names and descriptions, but storage within the program is not reserved because the data exists elsewhere.

3. The length of a level-01 record may not exceed 524,287 bytes.

5.6. COMMUNICATION SECTION

Function:

The communication description specifies the interface area between the message control system (MCS) and a COBOL program.

5.6.1. Input Communication Description

Format:

```
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 date-name-5]                      │
    FOR [INITIAL] INPUT   │    [;MESSAGE TIME IS date-name-6]                      │
                          │    [;SYMBOLIC SOURCE IS date-name-7]                   │
                          │    [;TEXT LENGTH IS data-name-8]                       │
                          │    [;END KEY IS data-name-9]                           │
                          │    [;STATUS KEY IS data-name-1Ø]                       │
                          │    [;MESSAGE COUNT IS data-name-11]                    │
                          └[data-name-1, data-name-2, ..., data-name-11]           ┘
```

Rules:

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 6.1.3.)

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:

   - 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.

   - 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.

   - 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.

   - 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.

   - The MESSAGE DATE clause defines data-name-5 as the name of a data item whose implicit description is that of an integer of six digits without an operational sign occupying character positions 49–54 in the record.

   - The MESSAGE TIME clause defines data-name-6 as the name of a data item whose implicit description is that of an integer of eight digits without an operational sign occupying character positions 55–62 in the record.

   - 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.

   - 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 four digits without an operational sign occupying character positions 75–78 in the record.

   - The END KEY clause defines data-name-9 as the name of an elementary alphanumeric data item of one character occupying position 79 in the record.

   - The STATUS KEY clause defines data-name-10 as the name of an elementary alphanumeric data item of two characters occupying positions 80–81 in the record.

   - 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 six digits without an operational sign occupying character positions 82–87 in the record.

   The listed clauses (see bulleted items in rule 5) may be replaced by a series of data-names (data-name-1, data-name-2, ..., data-name-11) that correspond to the order of data-names defined by these clauses.

   NOTE:

   Specification of a series of data-names on a single source line results in an incorrect cross-reference listing. The preferred method of writing a series of data-names is to specify each data-name on a separate source line.

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:

The comments are for clarification and are 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, but only the first redefinition may contain VALUE clauses. However, the MCS always references the record according to the data descriptions defined in 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.

8. 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.

9. The data items referenced by data-name-1, data-name-2, data-name-3, and data-name-4 (SYMBOLIC QUEUE, SYMBOLIC SUB-QUEUE-1, SYMBOLIC SUB-QUEUE-2, and SYMBOLIC SUB-QUEUE-3) contain symblic names designating queues and subqueues. All symbolic names must follow the rules for the formation of system names and must have been defined previously to the MCS.

10. 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.

11. 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.

    When a RECEIVE statement is executed, the input CD area must contain, in the contents of data-name-1, the name of a symbolic queue. The data items specified by data-name-2, data-name-3, and data-name-4 may contain symbolic subqueue names or spaces. 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 within the queue or subqueue specified in the input CD.

    After the execution of a RECEIVE statement, the contents of the data items referenced by data-name-1 through data-name-4 contain the symblic names of all the levels of the queue structure.

12. Whenever a program is scheduled by the MCS to process a message, the symbolic names of the queue structure that demanded this activity are 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.

13. The contents of data-name-5 (MESSAGE DATE) 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.

14. The contents of data-name-6 (MESSAGE TIME) 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.

15. During the execution of a RECEIVE statement, the MCS provides, in the data item referenced by data-name-7 (SYMBOLIC SOURCE), the symbolic name of the communications terminal that is the source of the message being transferred. However, if the symbolic name of the communications terminal is not known to the MCS, the contents of the data item referenced by data-name-7 will contain spaces.

16. The MCS indicates, via the contents of the data item referenced by data-name-8 (TEXT LENGTH), the number of character positions filled as a result of the execution of the RECEIVE statement.

17. The contents of the data item referenced by data-name-9 (END KEY) are set only by the MCS as part of the execution of a RECEIVE statement according to the following rules:

    - When the RECEIVE MESSAGE phrase is specified:

        1. If an end of group has been detected, the contents are set to 3.

        2. If an end of message has been detected, the contents are set to 2.

        3. If less than a message is transferred, the contents are set to 0.

    - When the RECEIVE SEGMENT phrase is specified:

        1. If an end of group has been detected, the contents are set to 3.

        2. If an end of message has been detected, the contents are set to 2.

        3. If an end of segment has been detected, the contents are set to 1.

        4. If less than a message segment is transferred, the contents are set to 0.

- 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.

18. The contents of the data item referenced by data-name-10 (STATUS KEY) 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 Table 5-11.

19. The contents of the data item referenced by data-name-11 (MESSAGE COUNT) indicate the number of messages that exist in a queue, sub-queue-1, ..., sub-queue-3. 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.

5.6.2. Output Communication Description

Format:

```
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].
```

Rules:

1. A CD must appear only in the communication section.

2. If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.

3. 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).

   - 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.

   - 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 four digits without an operational sign occupying character positions 5–8 in the record.

   - The STATUS KEY clause defines data-name-3 to be an elementary alphanumeric data item of two characters occupying positions 9 and 10 in the record.

- Character positions 11-23 and every set of 13 characters thereafter will form table items of the following description:

  - The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of one character.

  - The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 12 characters.

Use of these clauses results in a record whose implicit description is equivalent to the following:

```
Implicit Description                                   Comment

Ø1  data-name-Ø.
    Ø2  data-name-1  PICTURE 9(Ø4).                     DESTINATION COUNT
    Ø2  data-name-2  PICTURE 9(Ø4).                     TEXT LENGTH
    Ø2  data-name-3  PICTURE XX.                        STATUS KEY
    Ø2  data-name  OCCURS integer-2 TIMES.              DESTINATION TABLE
        Ø3  data-name-4  PICTURE X.                     ERROR KEY
        Ø3  data-name-5  PICTURE X(12).                 SYMBOLIC DESTINATION
```

NOTE:

The comments are for clarification and are not part of the description.

4. 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 rule 3.

5. Data-name-1, data-name-2, ..., data-name-5 must be unique within a CD.

6. If the DESTINATION TABLE OCCURS clause is not specified, one ERROR KEY and one SYMBOLIC DESTINATION area are assumed. In this case, neither subscripting nor indexing is permitted when referencing these data items.

7. If the DESTINATION TABLE OCCURS clause is specified, data-name-4 (ERROR KEY) and data-name-5 (SYMBOLIC DESTINATION) may be referenced only by subscripting or indexing.

8. In level 1, the value of the data item referenced by data-name-1 (DESTINATION COUNT) and integer-2 must be 1.

   In level 2, the value of the data item referenced by data-name-1 and integer-2 may not exceed 9999.

9. Output CD information is not sent to the terminal, but constitutes the communication between the program and the MCS as information about the message being handled.

10. During the execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the contents of the data item referenced by data-name-1 (DESTINATION COUNT) 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.

11. It is the responsibility of the user to ensure that the value of the data item referenced by data-name-1 (DESTINATION COUNT) is valid at the time of execution of the SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement.

12. As part of the execution of a SEND statement, the MCS interprets the contents of the data item referenced by data-name-2 (TEXT LENGTH) 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 the SEND statement.)

13. Each occurrence of the data item referenced by data-name-5 contains the name of a symbolic destination previously known to the MCS. These symbolic destination names must follow the rules for the formation of system-names.

14. The contents of the data item referenced by data-name-3 (STATUS KEY) 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 Table 5-11.

15. 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 (ERROR KEY) are updated.

    The actual association between the contents of the data item referenced by data-name-3 and the status condition itself is defined in Table 5-12.

Table 5—11. Communication Status Key Condition (Part 1 of 2)

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	Description
X	X	X	X	X	X	X	X	X	00	No error detected. Action completed.
	X								10	One or more destinations disabled. Action completed. (See Table 5-12.)
			X	X	X	X	X	X	15	One or more queues or destinations already disabled/enabled. (See Table 5-12.)
	X				X			X	20	One or more destinations unknown. Action completed for known destinations. (See Table 5-12.)
X		X	X			X			20	One or more queues or subqueues unknown. No action taken.
				X			X		20	Symbolic source unknown. No action taken.
	X								2A	One or more destinations in destination table were not in the table when the first portion of the message was sent. (See Table 5-12.)
	X				X			X	30	Destination count invalid. No action taken.
			X	X	X	X	X	X	40	Password invalid. No action taken.
	X								50	Text length exceeds size of identifier-1. No action taken.
	X								60	Partial segment with zero text length or no identifier-1 specified. No action taken.
	X								65	Output queue capacity exceeded. No action taken.
	X		X		X	X		X	80	A combination of at least two status key conditions 10, 15, and 20 occurred.
									91	ICAM NATTACH error. This error occurs during network initialization. The procedure division code of the program doesn't execute. Status code 91 appears only in an CE44 error message.
		X							92	ICAM QDEPTH error. No action taken.
			X	X		X	X		93	ICAM TRMREP error. No action taken.
					X			X	94	ICAM QHOLD/QRELSE error. No action taken.
X									95	ICAM GETCP error. No action taken.
	X								96	ICAM PUTCP error. No action taken.
X	X	X	X	X	X	X	X	X	99	Unrecoverable ICAM error. No action taken.

Table 5—11. Communication Status Key Condition (Part 2 of 2)

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	Description
X		X	X	X		X	X		9A	Process file undefined. No action taken.
	X								9C	Insufficient DTFs in CMCS to handle all output CDs. No action taken.

LEGEND:

X = Possible code for statement

NOTE:

Status codes 93, 94, 99, and 9A may also be reported as part of a CE44 error message.

Table 5—12. Error Key Codes

SEND	ENABLE OUTPUT	DISABLE OUTPUT	ERROR KEY Code	Description
X	X	X	0	No error
X	X	X	1	Symbolic destination unknown
X			2	Symbolic destination disabled
	X	X	5	Symbolic destination already enabled/disabled
X			A	Entries in destination table changed before message completion. SEND performed on original destinations.

Figure 5-6 provides a sample communication section including:

- An input communication description with certain optional clauses (lines 050100 through 050800), followed by an optional level 01 record description (lines 050900 through 051600).
- An input communication description without optional clauses (line 051700), followed by a required level 01 record description (lines 051800 through 052600).
- An output communication description without optional clauses (line 052700), followed by a required level 01 record description (lines 052800 through 053400).

```
Seq.   A   B     Text
No.
1      8   12
Ø5ØØØØ COMMUNICATION SECTION.
Ø5Ø1ØØ CD  COM-A-IN FOR INPUT
Ø5Ø2ØØ     SYMBOLIC QUEUE IS QUEUE-A;
Ø5Ø3ØØ     MESSAGE DATE IS MSG-DATE-A;
Ø5Ø4ØØ     MESSAGE TIME IS MSG-TIME-A;
Ø5Ø5ØØ     SYMBOLIC SOURCE IS SYM-SRC-A;
Ø5Ø6ØØ     TEXT LENGTH IS TXT-LGTH-A;
Ø5Ø7ØØ     STATUS KEY IS STAT-KEY-A;
Ø5Ø8ØØ     MESSAGE COUNT IS QUEUE-CNT-A.
Ø5Ø9ØØ Ø1  COM-A-REC.
Ø51ØØØ     Ø2  FILLER      PIC  X(78).
Ø511ØØ     Ø2  END-KEY-A   PIC  X.
Ø512ØØ         88  PART-SEG    VALUE "Ø".
Ø513ØØ         88  END-SEG     VALUE "1".
Ø514ØØ         88  END-MSG     VALUE "2".
Ø515ØØ         88  END-TRANS   VALUE "3".
Ø516ØØ     Ø2  FILLER      PIC  X(8).
Ø517ØØ CD  COM-B-IN FOR INPUT.
Ø518ØØ Ø1  COM-B-REC.
Ø519ØØ     Ø2  QUEUE-B      PIC  X(12).
Ø52ØØØ     Ø2  SUB-QUEUE-B  PIC  X(12).
Ø521ØØ     Ø2  FILLER       PIC  X(38).
Ø522ØØ     Ø2  SYM-SRC-B    PIC  X(12).
Ø523ØØ     Ø2  TXT-LGTH-B   PIC  9(4).
Ø524ØØ     Ø2  END-KEY-B    PIC  X.
Ø525ØØ     Ø2  STAT-KEY-B   PIC  XX.
Ø526ØØ     Ø2  QUEUE-CNT-B  PIC  9(6).
Ø527ØØ CD  COM-OUT FOR OUTPUT.
Ø528ØØ Ø1  COM-OUT-REC.
Ø529ØØ     Ø2  DEST-CNT     PIC  9(4).
Ø53ØØØ     Ø2  TXT-LGTH-OUT PIC  9(4).
Ø531ØØ     Ø2  STAT-KEY-OUT PIC  XX.
Ø532ØØ     Ø2  DEST-TBL  OCCURS   1Ø TIMES.
Ø533ØØ         Ø4  ERR-KEY  PIC  X.
Ø534ØØ         Ø4  SYM-DEST PIC  X(12).
```

Figure 5—6. Sample Communication Section Entries

6. Procedure Division

6.1. GENERAL

The procedure division of a COBOL program contains the procedures needed to solve a data processing problem. These procedures are written in COBOL statements that may be combined to form sentences. Groups of sentences may form paragraphs, and paragraphs may be grouped to form sections.

The procedure division is required for every COBOL source program. The division begins with the division header PROCEDURE DIVISION followed, optionally, by declaratives, which are followed by nondeclarative procedures.

6.1.1. Declaratives

Declaratives specify those conditions that normally cannot be tested by the programmer and the associated procedures to be executed when the specified conditions occur.

Declaratives consist of one or more sections grouped at the beginning of the procedure division. The declarative sections are preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES. A declarative section consists of a section header followed by a USE compiler-directing sentence followed by a set of zero, one, or more associated paragraphs (6.1.3).

6.1.2. Procedures

A procedure is composed of a paragraph or group of successive paragraphs, or a section or 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 keywords 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 keywords 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.

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.

6.1.3. PROCEDURE DIVISION STRUCTURE

6.1.3.1. 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] ... ].
```

Rules:

1. 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.

2. 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.

3. 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, identifier-2, file-name-1, or cd-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, identifier-3, file-name-2, or cd-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, identifier, file-name, or cd-name may appear more than once in the same USING phrase of a CALL statement.

4. If the USING phrase is specified, the INITIAL clause must not be present in any CD entry.

6.1.3.2. 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] ... } ...
```

Rules:

1. The procedure division must be divided into sections when the program is to be segmented or when declaratives are present.

2. Format 2 is used when the entire procedure division is composed of paragraphs only. However, if one paragraph is in a section, then all paragraphs must be in sections.

3. If sections are used, section-names must be unique within a program and paragraph-names must be unique within a section.

   If sections are not used, paragraph-names must be unique within a program.

4. When program segmentation is used, sections are classified by segment-numbers. The segment-number must be an integer ranging in value from 0 through 99.

   All sections with the same segment-number constitute a program segment. In Level 1, sections with the same segment-number must be contiguous in the source program.

   In Level 2, sections with the same segment-number need not be physically contiguous in the source program.

5. Segments with segment-numbers 0 through 49 belong to the fixed portion of the object program. In Level 1, all sections with segment-numbers 0 through 49 must be together in the same program.

   Segments with segment-numbers 50 through 99 are independent segments.

6. If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

7. Sections in the declaratives must contain segment-numbers less than 50.

Example:

An example of the procedure division is given in Figure 6-1.

```
Seq.   A   B      Text
No.
1      8   12
------------------------------------------------------------------------
071010 PROCEDURE DIVISION.
071020 DECLARATIVES.
071030 ALPHA SECTION. USE AFTER STANDARD ERROR PROCEDURE ON FILE-A.
071040 A-1.
071050     ADD 1 TO ERROR-COUNT.
071060     IF ERROR-COUNT > 10 GO TO A-4.
071070     IF INDICATOR NOT EQUAL 1 GO TO A-3.
071080 A-2.
071090     DISPLAY "HAD A "TYPE-ERROR" ERROR. RECOVERED".
071100     GO TO A-5.
071110 A-3.
071120     DISPLAY "UNRECOVERABLE ERROR ON FILE-A" ERROR-TYPE.
071130     STOP RUN.
071140 A-4.
071150     DISPLAY "MORE THAN TEN ERRORS ON FILE-A. TERMINATING.".
071160     STOP RUN.
071170 A-5. EXIT.
071180 END DECLARATIVES.
072010 MAIN SECTION.
072020 HOUSEKEEPING.
072030     ACCEPT CURRENT-NAME FROM MSG-DEVICE.
072040     OPEN INPUT-FILE-A.
072050     OPEN OUTPUT FILE-B.
072060     MOVE "C" TO B-SWITCH.
072070 BASIC-ROUTINE.
072080     READ FILE-A, AT END GO TO END-ROUTINE.
072090     MOVE CORRESPONDING RECORD-A TO RECORD-B.
072100     ADD NUMBER-A TO HASH-TOTAL.
072110     GO TO BASIC-ROUTINE.
072120 END-ROUTINE.
072130     DISPLAY "FINAL HASH TOTAL WAS "HASH-TOTAL"."
072140         UPON MSG-DEVICE.
072150     CLOSE FILE-A, FILE-B.
072160         STOP RUN.
```

Figure 6—1. Sample Procedure Division

6.2. CATEGORIES OF STATEMENTS

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

6.2.1. Imperative Statements

An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement may consist of a sequence of imperative statements.

The COBOL verbs used in imperative statements are:

ACCEPT
ADD
ALTER
CALL
CANCEL
CLOSE
COMPUTE
DELETE
DISABLE
DISPLAY
DIVIDE
ENABLE
EXHIBIT
EXIT
GO TO
INSPECT
MERGE
MOVE
MULTIPLY
OPEN
PERFORM
READ
RECEIVE
RELEASE
REWRITE
SEND
SET
SORT
START
STOP
STRING
SUBSTRACT
TRACE
TRANSFORM
UNSTRING
WRITE

6.2.2. Conditional Statements

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.

The COBOL verbs used in conditional statements are listed as follows. The optional phrase in parentheses, when included with the statement, causes otherwise imperative statements to become conditionals.

ADD (SIZE ERROR)
CALL (OVERFLOW)
COMPUTE (SIZE ERROR)
DELETE (INVALID KEY)
DIVIDE (SIZE ERROR)
EXHIBIT (CHANGED)
IF
MULTIPLY (SIZE ERROR)
ON
READ (END or INVALID KEY)
RECEIVE (NO DATA)
RETURN
REWRITE (INVALID KEY)
SEARCH
START (INVALID KEY)
STRING (OVERFLOW)
SUBTRACT (SIZE ERROR)
UNSTRING (OVERFLOW)
WRITE (INVALID KEY or END-OF-PAGE)

6.2.3. Compiler-Directing Statements

A compiler-directing statement causes the compiler to take a specific action during compilation. COBOL verbs used in compiler-directing statements are:

COPY
USE
*DEBUG

6.3. ARITHMETIC EXPRESSIONS

Arithmetic expressions are used as operands of certain conditional and arithmetic statements.

An arithmetic expression can consist of any of the following:

- An identifier of a numeric elementary item
- A numeric literal
- A combination of item 1 and 2 identifiers and literals separated by arithmetic operators
- Two arithmetic expressions separated by an arithmetic operator
- An arithmetic expression enclosed in parentheses

Any arithmetic expression may be preceded by a unary operator. The identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

In an arithmetic expression:

- if one of the two operands in a simple operation is a floating-point item, the intermediate resultant item is floating point; or
- if an exponentiation is specified, the intermediate resultant item is floating point.

Floating-point operations preserve high-order digit accuracy but lose low-order digit precision. (See 6.5.1, 6.5.2, and Appendix G.)

6.3.1. 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

6.3.2. Formation and Evaluation Rules

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first; 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:

First - Unary plus and minus
Second - Exponentiation
Third - Multiplication and division
Fourth - Addition and subtraction

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.

Example 1:

In the expression

```
A + B - C * D
```

C and D are multiplied first, A is then added to B, and the product of C * D is subtracted from the result of A + B.

Example 2:

In the expression

```
A + (B - C) * D
```

C is first subtracted from B, (B - C) is then multiplied by D, and the total is added to A.

Example 3:

In the expression

```
A + (B / C) + ((D * E) ** F) - G
```

The order of evaluation is (1) division, (2) multiplication, (3) exponentiation, and (4) addition and subtraction from left to right.

Operators, variables, and parentheses that may be combined in an arithmetic expression are summarized in Table 6-1.

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.

Arithmetic expressions allow the user to combine arithmetic operations without restrictions on composite of operand and/or receiving data items. (See 6.6.2, rule 3, and Appendix E.)

Table 6—1. Permissible Symbol Combinations in Arithmetic Expressions

First Symbol	Second Symbol				
	Variable	*** / * * — +**	**Unary + or —**	**(**	**)**
Variable	—	P	—	—	P
* / * * + —	P	—	P	P	—
Unary + or —	P	—	—	P	—
(	P	—	P	P	—
)	—	P	—	—	P

LEGEND:

P	Indicates that the two symbols may appear consecutively
—	Indicates that the two symbols may not appear consecutively
Variable	Represents an identifier or a literal

6.4. 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.

6.4.1. Simple Conditions

The simple conditions are:

- Relation condition
- Class condition
- Condition-name condition
- Switch-status condition
- Sign condition

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.

6.4.1.1. 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. See Table 6-2 for a summary of permissible comparisons.

Table 6—2. Permissible Comparisons for Relation Conditions

First Operand	**Second Operand**													
	GR	**AL**	**AN**	**ANE**	**NE**	**FC①** **NNL**	**ZR** **NL**	**ED**	**BI**	**ID**	**EF**	**IF**	**IN**	**IDI**
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Alphanumeric edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Numeric edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN			NN			
Figurative constant (FC) ① and nonnumeric literal (NNL)	NN	NN	NN	NN	NN			NN			NN			
Figurative constant ZERO (ZR) and numeric literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU	NU	IN②	
External decimal (ED)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	IN②	
Binary (BI)							NU	NU	NU	NU	NU	NU	IN②	
Internal decimal (ID)							NU	NU	NU	NU	NU	NU	IN②	
External floating point (EF)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU		
Internal floating point (IF)							NU	NU	NU	NU	NU	NU		
Index name (IN)							IN②	IN②	IN②	IN②			TI	ID
Index data item (IDI)													ID	ID

NOTES:

① FC includes all figurative constants except ZERO.
② Valid only if the numeric item is an integer.

LEGEND:

NN	=	comparison as described for nonnumeric operands
NU	=	comparison as described for numeric operands
TI	=	comparison as described between two index-names
IN	=	comparison as described between index-name and numeric integer
ID	=	comparison as described between index data item and index-name or other index data item

Format:

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

NOTE:

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

The first operand is the subject of the condition; the second operand is the object of the condition. The subject and object may not both be literals.

The relational operator specifies the type of comparison to be made in a relation condition. The relational operators and their meanings are 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 = cre not underlined to avoid confusion with other symbols such as ≥ (greater than or equal to).

A space must precede and follow each reserved word in the relational operator. When used, NOT and the next keyword or relation character form 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.

6.4.1.1.1. Comparison of Numeric Operands

For numeric class operands, a comparison is made of 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.

6.4.1.1.2. 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 4.3.2, 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:

- 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 6.6.20, the MOVE statement, and 5.3.3.4, the PICTURE character P.)

- 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 6.6.20, the MOVE statement, and 5.3.3.4, the PICTURE character P.)

- A noninteger 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.

- Operands of equal size

  If the operands are of equal size, characters in corresponding character positions are compared 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.

- 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.

6.4.1.1.3. Comparisons Involving Index-Names or Index Data Items

The comparison of two index-names is equivalent to the comparison of their corresponding occurrence numbers.

The comparison of an index-name with a numeric item (data item or literal) is permitted if the numeric item is an integer. The numeric integer is treated as an occurrence number.

In the comparison of an index data item with an index-name or with another index data item, the actual values are compared without conversion.

Other comparisons involving an index-name or index data item are not allowed.

6.4.1.2. Class Condition

The class condition determines whether the operand is numeric (consists entirely of the characters 0 through 9, with or without the operational sign) or alphabetic (consists entirely of the characters A through Z and space).

Format:

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

The USAGE of the operand must be described explicitly or implicitly as DISPLAY or COMPUTATIONAL-3. When used, NOT and the next keyword 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 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.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic, as external floating-point, or as a group item composed of elementary items whose data descriptions indicate the presence of operational signs.

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 the operational sign position contains a hexadecimal value of F.

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 –. Valid operational signs for data items not described with the SIGN IS SEPARATE clause are the hexadecimal values C or F and D. (See 5.3.3.6, the SIGN clause.)

Examples:

PICTURE	Data	Data-Item Is Considered
S99	X'F1F2'	NUMERIC
S99	X'F1C2'	NUMERIC
S99	X'F1D2'	NUMERIC
99	X'F1F2'	NUMERIC
99	X'F1C2'	NOT NUMERIC
99	X'F1D2'	NOT NUMERIC

6.4.1.3. Condition-Name Condition

In a condition-name condition, a conditional variable is tested to determine whether its value is equal to one of the values associated with a condition-name.

Format:

```
condition-name
```

If the condition-name is associated with a range or ranges of values, the conditional variable is tested to determine whether 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 (6.4.1.1).

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.

6.4.1.4. Switch-Status Condition

A switch-status condition determines the on or off status of a system task switch. The switch name and the on or off value associated with the condition are named in the SPECIAL-NAMES paragraph of the environment division (4.3.3).

Format:

```
condition-name
```

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

6.4.1.5. Sign Condition

The sign condition determines whether the algebraic value of an arithmetic expression is less than, greater than, or equal to zero.

Format:

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

The arithmetic expression must contain at least one reference to a variable.

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.

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.

6.4.2. Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions, or complex conditions. The conditions are either connected logically with the logical operators AND or OR, or negated logically with the logical operator NOT.

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.

The truth value of a complex condition, whether parenthesized or not, is the truth value that results from: (1) the interaction of all the stated logical operators on the individual truth values of simple conditions, or (2) the intermediate truth values of conditions logically connected or logically negated.

Table 6-3 shows the relationship between the logical operators and simple conditions A and B.

Table 6—3. Logical Operators and the Resultant Values

Value of A	Value of B	NOT A	A AND B	A OR B	NOT (A AND B)	NOT A AND B	NOT (A OR B)	NOT A OR B
True	True	False	True	True	False	False	False	True
False	True	True	False	True	True	True	False	True
True	False	False	False	True	True	False	False	False
False	False	True	False	False	True	False	True	True

6.4.2.1. Negated Simple Conditions

A simple condition (6.4.1) 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.

Format:

NOT simple-condition

6.4.2.2. Combined and Negated Combined Conditions

A combined condition results from connecting conditions with one of the logical operators AND or OR.

Format:

$$\text{condition}\ \left\{\left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\}\ \text{condition}\right\}\ \ldots$$

The condition may be one of the following:

1. A simple condition
2. A negated simple condition
3. A combined condition
4. A negated combined condition, i.e., the NOT logical operator followed by a combined condition enclosed within parentheses
5. Combinations of the first four conditions specified according to the rules summarized in Table 6-4

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 6-4, and paragraph 6.4.3.)

Table 6-4 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.

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.

Table 6—4. Combinations of Conditions, Logical Operators, and Parentheses

Element	Location (left-to-right)			
	First	Last	Intermediate Position	
			Allowable Preceding Elements	Allowable Following Elements
C*	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	C,)	C, NOT, (
NOT	Yes	No	OR, AND, (	C, (
(	Yes	No	OR, NOT, AND, (	C, NOT, (
)	No	Yes	C,)	OR, AND,)

*C = simple-condition

6.4.2.3. 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.

Format:

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

Within a sequence of relation conditions, both 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 6–4. This insertion of an omitted subject or omitted subject and 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 operand 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)))

6.4.3. Condition Evaluation Rules

Parentheses may be used to specify the order in which individual conditions of complex condition 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 6.3.)

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

   a. relation (following the expansion of any abbreviated relation condition)

   b. class

   c. condition-name

   d. switch-status

   e. 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.

6.5. COMMON PHRASES AND GENERAL RULES FOR STATEMENT FORMATS

In the statement descriptions in 6.6, several phrases appear frequently: the ROUNDED phrase, the SIZE ERROR phrase, and the CORRESPONDING phrase.

In the following discussion, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

6.5.1. 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 1 whenever the most significant digit of the excess is greater than or equal to 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.

The ROUNDED phrase is not applicable to a floating-point resultant-identifier.

6.5.2. 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 exits. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and not to intermediate rsults except for the MULTIPLY and DIVIDE statements. 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.

- If the SIZE ERROR phrase is not specified and a size error condition occurs, the value of those resultant-identifiers affected is undefined. Values of resultant-identifiers for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifiers during execution of this operation.

- If the SIZE ERROR phrase is specified and a size error condition occurs, then the values of resultant-identifiers affected by the size errors are not altered. Values of resultant-identifiers for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifiers during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

The SIZE ERROR phrase is not applicable to floating-point resultant-identifiers, except division by zero which always causes a size error condition.

If any of the individual operations of an ADD or SUBTRACT statement with the CORRESPONDING phrase 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.

6.5.3. The CORRESPONDING Phrase

In the ADD, SUBTRACT, or MOVE statement with the CORRESPONDING phrase, both identifier-1 and identifier-2 must refer to group items. In the following discussion, d_1 and d_2 refer to identifier-1 and identifier-2, respectively.

A data item from d_1 and one from d_2 correspond under the following conditions:

- 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.

- 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 or SUBTRACT statement with the CORRESPONDING phrase.

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

- 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 5.3.3.7, the OCCURS clause.)

Example:

```
SUBTRACT CORRESPONDING EMPLOYEE-RECORD FROM PAYROLL-CHECK
```

Data Division Entries:

```
Ø1 EMPLOYEE-RECORD            Ø1 PAYROLL-CHECK
    Ø2 EMPLOYEE-NUMBER            Ø2 EMPLOYEE-NUMBER
       Ø3 FILLER                     Ø3 CLOCK-NUMBER
       Ø3 PLANT-LOCATION             Ø3 FILLER
       Ø3 CLOCK-NUMBER            Ø2 DEDUCTIONS
          Ø4 SHIFT-CODE              Ø3 FICA-RATE
          Ø4 CONTROL-NUMBER          Ø3 WITHHOLDING-TAX
    Ø2 INCOME                        Ø3 PERSONAL-LOANS
       Ø3 HOURS-WORKED            Ø2 INCOME
       Ø3 PAY-RATE                   Ø3 HOURS-WORKED
    Ø2 FICA-RATE                     Ø3 PAY-RATE
    Ø2 DEDUCTIONS                 Ø2 NET-PAY
                                  Ø2 EMPLOYEE-NAME
                                     Ø3 SHIFT-CODE
```

In the example, the corresponding items are:

HOURS-WORKED
PAY-RATE

The following items are not corresponding in the example for the reasons stated:

Item	Reason
EMPLOYEE-NUMBER	Items not elementary
FILLER	FILLER not considered corresponding items
CLOCK-NUMBER	Item not elementary in one group
SHIFT-CODE	Qualifications not identical
INCOME	Items not elementary
FICA-RATE	Qualifications not identical
DEDUCTIONS	Item not elementary in one group

6.5.4. The Arithmetic Statements

The operands of the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements have several common features:

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

- The maximum size of each operand is 18 decimal digits. The composite of operands, which is hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points, must not contain more than 18 decimal digits.

6.5.5. 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.

6.5.6. 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 that 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 compiler.

6.5.7. WHEN-COMPILED Special Register

The reserved word WHEN-COMPILED is the name of a compiler-generated 17-byte alphanumeric field. It makes the date and time of the compilation available to the object program. The format of this field is yy/mm/dd△hh:mm:ss.

6.6. COBOL VERBS

The COBOL verbs listed in 6.2 are explained in detail in this paragraph. The verbs are presented alphabetically, with formats and rules.

6.6.1. ACCEPT Statement

Function:

The ACCEPT statement causes low-volume data to be made available to the specified data item.

Format 1:

```
ACCEPT identifier [FROM mnemonic-name]
```

Format 2:

```
ACCEPT identifier FROM ⎧DATE⎫
                       ⎨DAY ⎬
                       ⎩TIME⎭
```

Format 3:

```
ACCEPT cd-name MESSAGE COUNT
```

Format 4:

```
ACCEPT identifier-1 [,identifier-2] ...
       FROM [SPECIFIC] mnemonic-name
       ⎡USING⎧identifier-3⎫⎤
       ⎣     ⎩literal     ⎭⎦
       [ON EXCEPTION imperative-statement]
```

Format 5:

```
ACCEPT identifier-1 FROM mnemonic-name
       [ON EXCEPTION imperative-statement]
```

Rules:

1. The size of a data transfer is defined as follows:

SYSIN	80, 90, or 128 characters. If the length of a record is other than 80 or 96 characters, then 128 is used as the size of a data transfer. Records other than 80 or 90 characters in length commonly occur when a job stream file is created or updated by the general editor.
SYSCONSOLE	60 characters
SYSCOM	12 characters
SYSSWCH	8 characters
SYSSWCH-n	1 character
SYSTEM-SHUTDOWN	1 character
SYSWORK	1 – 1920 characters
SYSFORMAT	1 – 1920 characters
SYSTERMINAL	60 characters

If the size of the data being transferred exceeds the appropriate size of a data transfer, the excess data is lost during a data transfer.

NOTE:

Rules 2 through 10 pertain to format 1 only.

2. The identifier must be defined implicitly or explicitly as USAGE IS DISPLAY.

3. The mnemonic-name must also be specified in the SPECIAL-NAMES paragraph of the environment division and must be associated with SYSIN, SYSCONSOLE, SYSCOM, SYSSWCH, SYSTERMINAL, SYSSWCH-n, SYSTEM-SHUTDOWN, or SYSWORK.

4. The ACCEPT statement causes the transfer of data from a system logical device. This data replaces the contents of the data item named by the identifier. No editing or error checking of the incoming data is performed.

5. If the mnemonic-name is associated with SYSIN or SYSCONSOLE and:

   a. If the length of the receiving data item is less than or equal to the appropriate size of a data transfer, the transferred data is stored in the receiving data item left-aligned with space-fill or truncation to the right when appropriate.

   b. If the size of the receiving data item exceeds the appropriate size of a data transfer, the transferred data is stored in the receiving data item left-aligned. Additional data is requested and stored contiguously in the remaining portion of the receiving data item. When the size of the remaining portion is less than or equal to the appropriate size of a data transfer, additional data is requested once more. The transferred data is stored in the remaining portion with space-fill or truncation to the right when appropriate.

6. If the mnemonic-name is associated with SYSCOM, the 12-byte information in the communications region of the job preamble is moved to the 12-byte area described by the identifier.

7. If the mnemonic-name is associated with SYSSWCH, the information in the user program switch indicator (UPSI) byte is expanded to eight bytes. Each byte represents an individual switch. If the mnemonic-name is associated with SYSSWCH-n, the appropriate switch is expanded to one byte.

8. If the mnemonic name is associated with SYSTEM-SHUTDOWN, the shutdown indicator in the system information block (SIB) is expanded to one byte, with a character value of 0 or 1 (hexadecimal F0 or F1). Hexadecimal F1 indicates that the system operator entered a shutdown command through the console and plans to terminate all system processing.

9. If the FROM phrase is not specified, the system logical device SYSIN is assumed.

10. The /* is not accepted as an end statement into the program when accepting embedded data.

NOTE:

Rules 11 through 14 pertain to format 2 only.

11. 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.

12. 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.

13. 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.

14. 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.

    NOTE:

    It is possible for the maximum number of hours to be 99 if the system generation parameter TIMER is set to NO or MIN.

NOTE:

Rules 15 through 17 pertain to format 3 only.

15. Cd-name must reference an input CD.

16. 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, ..., sub-queue-3.

17. 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.

NOTE:

Rules 18 through 32 pertain to format 4 only.

18. Format 4 is used to accept data from a workstation terminal that calls screen format services. The FROM phrase must be specified.

19. The data description of identifier-1 or identifier-2, . . . must not contain a subordinate entry that specifies an OCCURS DEPENDING clause.

20. Mnemonic-name must also be specified in the SPECIAL-NAMES paragraph of the environment division and must be associated with SYSFORMAT.

21. The literal must be a nonnumeric literal.

22. The literal or the contents of identifier-3 is made up of a 1- to 8-character name of the screen format.

23. More than one receiving data item may be specified. Identifier-1 or identifier-2 need not be described explicitly or implicitly as USAGE IS DISPLAY. If USAGE other than DISPLAY is specified, no data conversion is performed by the COBOL generated object code. If data conversion is required, it must be specified in the controlling screen format.

24. The SPECIFIC phrase is meaningful only for a multivolume workstation. The SPECIFIC phrase indicates that data is to be accepted from a particular workstation terminal; that is, the terminal indicated in the WS-ID field, if the CONTROL AREA clause is specified with the mnemonic-name in the SPECIAL-NAMES paragraph, or the terminal that participated in the most recently executed ACCEPT or DISPLAY statement that references the same mnemonic-name.

25. The ON EXCEPTION phrase must be specified if the mnemonic-name is declared in the SPECIAL-NAMES paragraph with the CONTROL AREA clause. The ON EXCEPTION phrase must not be specified if the mnemonic-name is declared without the CONTROL AREA clause.

26. The ON EXCEPTION phrase is executed when the execution of the ACCEPT statement is unsuccessful. (See key code 1, 2, 3, or 9 in Status Key 1 in Table 4–1.)

27. A screen format must be specified for a given workstation before data can be accepted or displayed. A screen format may be specified via job control language or by the USING phrase of an ACCEPT or DISPLAY statement.

28. When the USING phrase of an ACCEPT statement specifies a format different from the current screen format on the last used terminal, the new format must be an input-only screen format.

29. Another way of specifying a different screen format is to use the DISPLAY statement with the USING phrase referencing the new input-only screen format. The new format is displayed on the terminal but no data is transmitted to the screen since it is an input-only format.

30. For a multivolume workstation, the USING phrase of an ACCEPT statement that references a new screen format changes the screen only on one terminal; that is, the terminal indicated in the WS-ID field if the CONTROL AREA clause is specified, or the terminal most recently accessed if the CONTROL AREA clause is not specified.

31. For a multivolume workstation, the terminal that responds to an ACCEPT statement whose USING phrase references a new screen format could be different from the terminal whose screen format has been changed by the very same ACCEPT statement.

32. After an ACCEPT statement referencing a screen format that is erased after input (that is, option 3 of the ERASE/UNLOCK function was selected at screen format generation), the next ACCEPT or DISPLAY statement accessing the same terminal must include a USING phrase.

NOTE:

Rules 33 through 40 pertain to format 5 only.

33. Format 5 is used to accept data from a workstation terminal without using screen format services.

34. Identifier-1 must be specified explicitly or implicitly with the USAGE IS DISPLAY phrase.

35. The data description of identifier-1 or identifier-2, . . . must not contain a subordinate entry that specifies an OCCURS DEPENDING clause.

36. The FROM phrase is required. Mnemonic-name must also be specified in the SPECIAL-NAMES paragraph of the environment division and must be associated with SYSWORK.

37. The ON EXCEPTION phrase must be specified if the mnemonic-name is declared in the SPECIAL-NAMES paragraph with the CONTROL AREA clause. The ON EXCEPTION phrase must not be specified if the mnemonic-name is not described with the CONTROL AREA clause.

38. The ON EXCEPTION phrase is executed when the execution of the ACCEPT statement is unsuccessful. (See key code 1, 2, 3, or 9 in Status Key 1 of Table 4–1.)

39. If the length of the receiving data item (identifier-1) exceeds 1920 characters, the transferred data is stored in the receiving data item left-aligned and space-filled. No additional data is requested. If the length of the receiving data item is less than 1920 characters, the transferred data is stored in the receiving data item left-aligned with space-fill or truncation to the right when appropriate.

40. After the execution of a format 5 ACCEPT statement, the cursor is positioned at the start of the next line.

6.6.2. ADD Statement

Function:

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

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]
```

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 fixed-point operands must not contain more than 18 digits. (See 6.5.4.)

   - In format 1, the composite of operands is determined by using all of the fixed-point operands in a given statement.

   - In format 2, the composite of operands is determined by using all of the fixed-point operands in a given statement excluding the data items that follow the word GIVING.

   - 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. See 6.5.1 ROUNDED phrase; 6.5.2, the SIZE ERROR phrase; 6.5.3, the CORRESPONDING phrase; 6.5.4, arithmetic statements; 6.5.5, overlapping operands; and 6.5.6, multiple results in arithmetic statements.

6. 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.

7. 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.

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

6.6.3. ALTER Statement

Function:

The ALTER statement modifies a predetermined sequence of operations.

Format:

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2
      [, procedure-name-3 TO [PROCEED TO] procedure-name-4] ...
```

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.

3. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ... so that subsequent execution 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 10.2.2.)

   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 10.)

6.6.4. CALL Statement

Function:

The CALL statement causes control to be transferred from one object program to another, within the run unit.

Format:

```
CALL {identifier-1}  [USING {data-name-1 }  [, {data-name-2 }] ... ]
     {literal-1   }         {cd-name-1   }      {cd-name-2   }
                            {identifier-2}      {identifier-3}
                            {file-name-1 }      {file-name-2 }
     [;ON OVERFLOW imperative-statement]
```

Rules:

1. Literal-1 must be a nonnumeric literal.

2. Identifier-1 must be defined as an alphanumeric data item.

3. The value of literal-1 or identifier-1 represents a 1- to 6-character load module name, or the value of literal-1 represents the name of the entry point if the called program is statically bound with the calling program.

4. 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. The number of operands in each USING phrase must be identical.

5. Each of the operands in the USING phrase must have been defined as a data item in the file section, working-storage section, linkage section, or communication section; or as a file-name in the file section, or a CD-name in the communication section. Data-name-1, data-name-2,..., may be qualified when they reference data items defined in the file section or the communication section.

6. 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.

7. The execution of a CALL statement causes control to pass to the called program.

8. 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.

9. If, during the execution of a CALL statement, it is determined that the available portion of object time storage 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, in addition, the ON OVERFLOW phrase is not specified, the calling program is terminated and the disposition of the run unit is handled by the operating system.

10. Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

11. 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 that 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 indexes.

12. The CALL statement may appear anywhere within a segmented program. 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. When using parameter CALLST=YES in a segmented program, the linkage editor commands must ensure that the called program can be accessed from the overlay that contains the CALL statement (see 10.5.4).

13. Literal-1 or the contents of the data item referenced by identifier-1 are used to identify the called program.

14. When the called program is a COBOL program, each of the operands in the USING phrase of the calling program must be defined as a data item in the file section, working-storage section, or linkage section. If the called program is written in a language other than COBOL, the operands of the USING clause may also be file-names and the address of the data management keyword attributes (DTF or RIB) is passed to the called program.

15. Programs called by the literal-1 option exclusively may be either linked with the calling program or dynamically loaded. (Refer to Appendix A for the CALLST compiler option parameter.) Programs called by identifier-1 option are always dynamically loaded.

6.6.5. CANCEL Statement

Function:

The CANCEL statement releases the main storage areas occupied by the referenced program.

Format:

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

Rules:

1. Literal-1, literal-2, ... must each be a nonnumeric literal.

2. Identifer-1, identifier-2, ... must each be defined as an alphanumeric data item such that its value can be a program name.

3. Refer to the CALL statement, 6.6.4, for a description of the value of identifier-1 and literal-1.

4. Literal-1, literal-2, ... must refer to called programs that are dynamically loaded. (Refer to Appendix A for the CALLST compiler option parameter.)

5. Subsequent to the execution of a CANCEL statement, the program referenced 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 main storage areas associated with the named programs are released so as to be made available for disposition by the operating system.

6. A program named in the CANCEL statement must not refer to any program that has been called but has not yet executed an EXIT PROGRAM statement.

7. A logical relationship to a canceled subprogram is established only by executing a subsequent CALL statement.

8. A called program is canceled either by being referred to as the operand of a CANCEL statement or by the terminal of the run unit of which the program is a member.

9. 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 canceled. Control passes to the next statement.

10. A called subprogram must not contain a CANCEL statement that directly or indirectly cancels the calling program, or any other program higher than itself in the calling hierarchy.

11. A program may CANCEL a program that it did not call, providing that in the calling hierarchy it is higher than or equal to the program it is canceling.

12. Literal-1 or the contents of the data item referenced by identifier-1 are used to identify the canceled program.

6.6.6. CLOSE Statement

Function:

The CLOSE statement terminates the processing of reels/units and files with optional rewind or lock or removal where applicable.

Format 1 (Sequential and SAM* Files):

```
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     ⎭      ⎦⎦
```

Format 2 (Relative, Indexed, and ISAM* Files):

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

Rules:

1. The REEL or UNIT phrase must only be used for sequential or SAM* files.

2. The files referenced in the CLOSE statement need not all have the same organization or access.

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

3. Except where otherwise stated, the terms reel and unit are synonymous and completely interchangeable in the CLOSE statement. Treatment of mass storage sequential or SAM* files is logically equivalent to the treatment of a file on tape or analogous sequential media.

4. A CLOSE statement may only be executed for a file in an open mode.

5. To show 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 or SAM* file that is entirely contained on one reel/unit.

   c. Sequential multiple reel/unit - A sequential or SAM* file that is contained on more than one reel/unit.

   d. Nonsequential single/multiple reel/unit - A relative, indexed, or ISAM* file contained on one or more units.

6. The results of executing each type of CLOSE statement for each category of files are given in Table 6-5.

NOTE:

The symbols used in Table 6—5 are defined in the text following the table. Definitions apply to all input, output, and input/output files except where noted.

Table 6—5. Relationship of Categories of Files and the Options of the CLOSE Statement

CLOSE Statement Format	**File Category**			
	Non-Reel/Unit	**Sequential Single Reel/Unit**	**Sequential Multiple Reel/Unit**	**Nonsequential Single/Multiple Reel/Unit**
CLOSE	C	C,G	C,G,A	C
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A	C,E
CLOSE WITH NO REWIND	X	C,B	C,B,A	X
CLOSE REEL/UNIT	X	X	F,G	X
CLOSE REEL/UNIT FOR REMOVAL	X	X	F,D,G	X
CLOSE REEL/UNIT WITH NO REWIND	X	X	F,B	X

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

The symbols used in Table 6-5 are defined as follows:

Symbol Definition

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 system 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 system 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 (Sequential Access Mode)

  If the file is positioned at its end and standard system label records are specified for the file, the system labels are processed according to the system standard label convention. Closing operations specified by the system are executed. If the file is positioned at its end and standard system label records are not specified for the file, label processing does not take place but other closing operations specified by the system are executed. If the file is positioned other than at its end, the closing operations specified by the system 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 standard system label records are specified for the file, the labels are processed according to the system standard label convention. Closing system operations specified by the system are executed. If standard system label records are not specified for the file, label processing does not take place but other closing operations specified by the system are executed.

D Reel/Unit Removal

The current reel or unit is rewound when applicable, and 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

The operating system is notified to ensure that this file cannot be opened again during this execution of this run unit.

Symbol Definition

F Close Reel/Unit

- Input Files

  The following operations take place:

  - A reel/unit swap
  - 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:

  - For output files only – The standard ending reel/unit label procedure is executed.
  - A reel/unit swap
  - 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 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.

7. If the file is in the open mode when a STOP RUN statement is executed, the file is to be closed by the compiler-generated object code. The result is unpredictable if 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.

8. 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.

9. If a CLOSE statement without the REEL or UNIT phrase has been executed for a file, no other statement (except the SORT statement 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.

10. The WITH NO REWIND and FOR REMOVAL phrases have no effect at object time if they do not apply to the storage media on which the file resides.

11. 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.

12. With the CLOSE WITH LOCK phrase, single reel tape files are rewound but are not unloaded.

6.6.7. COMPUTE Statement

Function:

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

Format:

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

Rules:

1. Identifiers that appear only to the left of = must refer to either an elementary numeric item or an elementary numeric edited item.

2. See 6.5.1, ROUNDED phrase, 6.5.2, SIZE ERROR phrase; 6.5.4, arithmetic statements; 6.5.5, overlapping operands; and 6.5.6, multiple results in arithmetic statements.

3. 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 6.3.)

4. 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.

5. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE. (See Appendix E.)

6.6.8. COPY Statement

Function:

The COPY statement incorporates text into a COBOL source program.

Format:

```
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           }
```

Rules:

1. Text-name or library-name must follow the rules for formation of a user-defined word; however, only the first eight characters of a text-name or library-name are used by the operating system. A text-name is used to identify a COBOL library text. A library-name is used as the LFD name to identify a COBOL library file.

2. If more than one COBOL library is available during compilation, text-name can be qualified by the library-name identifying the COBOL library in which the text associated with text-name resides.

   If the library-name is not specified, the file-names given in the LIN parameter are used. (See Appendix A, Compiler Options.)

   If the library-name is omitted in the COPY statement and the LIN parameter is not given, the default name COPY$ is used as the library-name.

3. The COPY statement must be preceded by a space and terminated by the separator period.

4. 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. The word COPY appearing in any comment entry is treated as a comment.

5. Pseudo-text-1 must not be null, nor may it consist solely of the character space, spaces, or comment lines.

6. Pseudo-text-2 may be null.

7. 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 2.7.)

8. Word-1 or word-2 may be any single COBOL word.

9. 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.

10. 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, inclusively.

11. 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. Pseudo-text-1, identifier-1, word-1, and literal-1 must not be a prefix or a suffix.

12. 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.

13. The comparison operation to determine text replacement occurs in the following manner:

    Any separator comma, semicolon, 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.

14. A comment line occuring 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.

15. 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.

16. 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.

17. Library text must conform to the rules for COBOL reference format.

18. Text-words after replacement are placed in the source program listing according to the rules for reference format.

19. Comment lines immediately following a COPY statement are placed in the source program listing immediately following the COPY statement and then are followed by the copied library text.

    If the comment lines are intended to follow the copied library text, then a blank line should be placed in the source program between the COPY statement and the comment lines.

6.6.9. DELETE Statement

Function:

The DELETE statement logically removes a record from a mass storage file.

Format:

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

Rules:

1. File-name must be the name of a relative or indexed file.

2. The INVALID KEY phrase must not be specified for a DELETE statement that refers to a file in the sequential access mode.

3. The INVALID KEY phrase must be specified for a DELETE statement that refers to a file not in sequential access mode and for which an applicable USE procedure is not specified.

4. The associated file must be open in the I-O mode at the time of the execution of this statement. (See 6.6.19, the OPEN statement.)

5. For files in the sequential access mode, the last input/output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The operating system logically removes from the file the record that was accessed by that READ statement.

6. For a file in random or dynamic access mode, the operating system logically removes from the file that record identified by the contents of the RELATIVE KEY or 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 8.2.5, the INVALID KEY condition.)

7. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

8. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

9. The current record pointer is not affected by the execution of a DELETE statement.

10. The excecution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated. (See 8.2.3, I-O status.)

6.6.10. DISABLE Statement

Function:

The DISABLE statement notifies the message control system (MCS) to inhibit data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

Format:

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

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 data item referenced by identifier-1 must be defined as alphanumeric, and its length must not exceed 10 characters.

4. 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.

5. When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all queues and subqueues 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.

6. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queues and subqueues 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.

7. 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.

8. Literal-1 or the contents of the data item referenced by identifier-1 are matched with a password built into the system. The DISABLE statement is honored only if literal-1 or the contents and the size of the data item referenced by identifier-1 match the system password. When literal-1 or the contents and the size 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 length of a password ranges from 1 to 10 characters inclusive.

9. The execution of a DISABLE statement causes the logical disconnection at the earliest time the source or destination is inactive. The execution of the DISABLE statement never causes the remaining portion of the message to be terminated during transmission to or from a terminal.

6.6.11. DISPLAY Statement

Function:

The DISPLAY statement causes low-volume data to be transferred to an appropriate system logical device.

Format 1:

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

Format 2:

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

        UPON mnemonic-name

        [USING {identifier-3}]
        [      {literal-3   }]

        [ON EXCEPTION imperative-statement]
```

Format 3:

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

        UPON mnemonic-name

        [ON EXCEPTION imperative-statement]
```

Rules:

NOTE:

Rules 1 through 5 pertain to all formats.

1. The DISPLAY statement causes the contents of each operand to be transferred to the device in the order listed.

2. Each literal may be any figurative constant except ALL.

3. If a figurative constant is specified as one of the operands, only one occurrence of the constant is displayed.

4. If the literal is numeric, it must be an unsigned integer.

5. The size of a data transfer is defined as follows:

Logical Device	Number of Characters
SYSLST or SYSOUT	120
SYSLOG	55
SYSCONSOLE	55
SYSTERMINAL	55
SYSCOM	12
SYSSWCH	8
SYSSWCH-n	1
SYSWORK	1–1920
SYSFORMAT	1–1920

NOTE:

Rules 6 through 14 pertain to format 1 only.

6. Mnemonic name is associated with a system logical device in the SPECIAL-NAMES paragraph of the environment division and must be associated with SYSLST, SYSOUT, SYSLOG, SYSCONSOLE, SYSTERMINAL, SYSCOM, SYSSWCH, or SYSSWCH-n.

7. If the UPON phrase is not specified, SYSLST is used.

8. If mnemonic-name is associated with SYSCOM, SYSSWCH, or SYSSWCH-n, only one operand is permitted in the statement.

9. If the mnemonic-name is associated with SYSLST or SYSOUT, and:

   a. if the length of the data item being transferred is less than or equal to 120 characters, the data is transferred to the associated system logical device.

   b. if the size of the data item being transferred exceeds 120 characters, the data, beginning with the leftmost character and up to the limit of 120 characters, is stored left-aligned in the associated system logical device. The remaining data is transferred sequentially in a like manner until all data is transferred.

10. If the mnemonic-name is associated with SYSLOG or SYSCONSOLE, the length of data to be displayed is limited to 55 characters.

11. If mnemonic-name is associated with SYSSWCH, eight characters are transferred. If mnemonic-name is associated with a single switch, SYSSWCH-n, one character is transferred.

12. 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.

13. If the identifiers are described implicitly or explicitly as USAGE other than DISPLAY, the contents of the data items, when transferred, are converted to DISPLAY format.

14. For numeric data items described with an operational sign without the SIGN IS SEPARATE clause, the operational sign is displayed as a separate character immediately following the data.

NOTE:

Rules 15 through 23 pertain to format 2 only.

15. Format 2 is used to display data on a workstation terminal calling screen format services. The UPON phrase must be specified.

16. Mnemonic-name must also be specified in the SPECIAL-NAMES paragraph of the environment division and must be associated with SYSFORMAT.

17. Literal-3 must be a nonnumeric literal.

18. Literal-3 or the content of identifier-3 is made up of a 1- to 8-character name of the screen format.

19. If identifier-1 or identifier-2, . . . is a group item, the data description entry of any subordinate item in the group must not contain an OCCURS DEPENDING clause.

20. If identifier-1 or identifier-2 is described implicitly or explicitly as USAGE other than DISPLAY, no data conversion is performed by the COBOL generated object code. If data conversion is required, it must be specified in the controlling screen format.

21. The ON EXCEPTION phrase must be specified if the mnemonic-name is declared in the SPECIAL-NAMES paragraph with the CONTROL AREA clause. The ON EXCEPTION phrase must not be specified if the mnemonic-name is not described with the CONTROL AREA clause.

22. The ON EXCEPTION phrase is executed when the execution of the DISPLAY statement is unsuccessful. (See key code 1, 2, 3, or 9 in Status Key 1 of Table 4-1.)

23. A screen format must be specified for a given workstation before data can be displayed. A screen format may be specified via job control language or by the USING phrase of a DISPLAY statement.

NOTE:

Rules 24 through 29 pertain to format 3 only.

24. Format 3 is used to display data on a workstation terminal without using screen format services. The UPON phrase must be specified.

25. Mnemonic-name must also be specified in the SPECIAL-NAMES paragraph of the environment division and must be associated with SYSWORK.

26. If identifier-1 or identifier-2, . . . is a group item, the data description entry of any subordinate item in the group must not contain an OCCURS DEPENDING clause.

27. The ON EXCEPTION phrase must be specified if the mnemonic-name is declared in the SPECIAL-NAMES paragraph with the CONTROL AREA clause. The ON EXCEPTION phrase must not be specified if the mnemonic-name is declared without the CONTROL AREA clause.

28. The ON EXCEPTION phrase is executed when the execution of the DISPLAY statement is unsuccessful. (See key code 1, 2, 3, or 9 in Status Key 1 of Table 4-1.)

29. After the execution of a format 3 DISPLAY statement, the cursor is positioned at the start of the next line.

6.6.12. DIVIDE Statement

Function:

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

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]
```

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 or any floating-point items) of a given statement aligned on their decimal points, must not contain more than 18 digits.

4. For a description of these functions, see 6.5.1, ROUNDED phrase; 6.5.2, SIZE ERROR phrase; 6.5.4, arithmetic statements; 6.5.5, overlapping operands; and 6.5.6, multiple results in arithmetic statements. See also rules 8 through 10 for a discussion of the ROUNDED phrase and the SIZE ERROR phrase as they pertain to formats 4 and 5.

5. 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.

6. 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.

7. 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.

8. 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. When the REMAINDER phrase is specified, none of the operands may be floating-point.

9. In formats 4 and 5, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described in rule 8. Appropriate decimal alignment and truncation (not rounding) will be performed for the content of the data item referenced by identifier-4, as needed.

10. 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 determine which situation has actually occurred.

6.6.13. ENABLE Statement

Function:

The ENABLE statement notifies the message control system (MCS) to allow data transfer between specified output queues and destinations for output or between specified sources and input queues for input.

Format:

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

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 data item referenced by identifier-1 must be defined as alphanumeric, and its length must not exceed 10 characters.

4. 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.

5. When the INPUT phrase with the optional word TERMINAL is specified, the logical path between the source and all associated queues and subqueues 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.

6. When the INPUT phrase without the optional word TERMINAL is specified, the logical paths for all of the sources associated with the queue and subqueues 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.

7. 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.

8. Literal-1 or the contents of the data item referenced by identifier-1 are matched with a password built into the system. The ENABLE statement is honored only if literal-1 or the contents and the size of the data item referenced by identifier-1 match the system password. When literal-1 or the contents and the size 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 length of a password ranges from 1 to 10 characters inclusive.

6.6.14. EXHIBIT Statement

Function:

The EXHIBIT statement displays the current values of data items at selected points in the program.

Format:

```
EXHIBIT ⎧NAMED        ⎫ ⎧identifier        ⎫ ...
        ⎨CHANGED NAMED⎬ ⎩nonnumeric-literal⎭
        ⎩CHANGED      ⎭
```

Rules:

1. An identifier length may not exceed 256 character positions.
2. An identifier may not be an index-data-item.
3. An EXHIBIT statement may appear anywhere in the procedure division or in a debugging packet.
4. Variable-length identifiers are not permitted with the CHANGED or CHANGED NAMED options.
5. The NAMED option displays the names of the identifiers specified with their current values and any nonnumeric literals specified.
6. The CHANGED NAMED option displays the names of the identifiers specified with their current value only if the value has changed since the EXHIBIT statement was last encountered. Any nonnumeric literals are displayed on every encounter.
7. The CHANGED option displays the value of the identifier specified, but only if the value has changed since the EXHIBIT statement was last encountered. Any nonnumeric literals are displayed on every encounter.
8. The first time an EXHIBIT statement is executed, all identifier values are considered changed.
9. Values of identifiers are displayed on SYSLST (4.3.3).
10. If two EXHIBIT statements each specify either the CHANGED or CHANGED NAMED option and the same identifier, the change in value of the identifier is associated independently with each of the two statements.

6.6.15. EXIT Statement

Function:

The EXIT statement provides a common end point for a series of procedures or marks the logical end of a called program.

Format:

```
EXIT [PROGRAM]
```

Rules:

1. The EXIT statement must appear in a sentence by itself. It must be preceded by a paragraph-name and be the only sentence in the paragraph.

2. An EXIT statement without the optional word PROGRAM serves only to enable the user to assign a procedure-name to a given point in a program. It has no other effect on the compilation or execution of the program.

3. An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. An EXIT PROGRAM statement in a program that is not called is executed like an EXIT statement without the PROGRAM phrase.

6.6.16. GO TO Statement

Function:

The GO TO statement causes control to be transferred from one part of the procedure division to another.

A format 3 GO TO statement is used as a special exit from a USE LABEL procedure.

Format 1:

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

Format 2:

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

Format 3:

```
GO TO MORE-LABELS
```

Rules:

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

2. A paragraph that is referenced by an ALTER statement must consist of only a paragraph header followed by a format 1 GO TO statement.

3. A format 1 GO TO statement without procedure-name-1 must be the only statement in the 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. 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.

6. 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.

7. 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.

8. A format 3 GO TO statement can appear only within a USE LABEL procedure.

9. When an input tape file is being processed, a format 3 GO TO statement is a request to the input/output control system to make the next standard use label record available and return control to the beginning of the same USE LABEL procedure for further checking of labels. The USE LABEL procedure is reentered only if there is another standard use label to be processed. Hence, there need not be a program path that flows through the last statement in the USE LABEL procedure.

10. When an output tape file is being processed, a format 3 GO TO statement requests the input/output control system to write the standard use label and return control to the beginning of the same USE LABEL procedure for further label creation. After the last standard use label is created, a program path must be provided that flows through the last statement of the USE LABEL procedure.

6.6.17. IF Statement

Function:

The IF statement causes a condition to be evaluated.(See 6.4.) The subsequent action of the object program depends on whether the value of the condition is true or false.

Format:

```
IF condition [THEN]  {statement-1  }  {;ELSE statement-2     }
             [  ;  ]  {NEXT SENTENCE}  {;ELSE NEXT SENTENCE   }
```

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.

3. When an IF statement is executed, the following transfers of control occur:

   a. If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not contain a procedure branching or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

   b. If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

c. If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching or conditional statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

d. If the condition is false, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

4. IF statement-1 or statement-2 contains an IF statement, the IF statement is said to be nested.

5. 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.

6.6.18. INSPECT Statement

Function:

The INSPECT statement provides the capability 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.

Format 1:

```
INSPECT identifier-1 TALLYING

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

Format 2:

```
INSPECT identifier-1 REPLACING
 ( CHARACTERS BY {identifier-6} [ {BEFORE} INITIAL {identifier-7} ]                         )
 (               {literal-4   } [ {AFTER }         {literal-5   } ]                         )
 (                                                                                          )
 {            ( , {ALL    } ( , {identifier-5} BY {identifier-6}  ) [...] ) [...]           }
 (            (   {LEADING} (   {literal-3   }    {literal-4   }  )       )                 )
 (            {   {FIRST  } {                                     }       }                 )
 (            (             ( [ {BEFORE} INITIAL {identifier-7} ] )       )                 )
 (            (             ( [ {AFTER }         {literal-5   } ] )       )                 )
```

Format 3:

```
INSPECT identifier-1 TALLYING

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

REPLACING

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

Rules:

1. Identifier-1 must refer to 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 refer to either an elementary alphabetic, alphanumeric, or numeric item described either implicitly or explicitly as USAGE IS DISPLAY.

3. Each literal must be nonnumeric and may be any figurative constant except ALL.

4. Literal-1, literal-2, literal-3, literal-4 and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7 must be one character in length in Level 1. Except as specifically noted in the rules, this restriction on length does not apply to Level 2.

5. Identifier-2 must refer to an elementary numeric data item (formats 1 and 3).

6. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit 1-character data item (formats 1 and 3).

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 (formats 2 and 3).

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 (formats 2 and 3).

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 (formats 2 and 3).

10. Inspection, which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying 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 rules 13 through 15.

11. 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 are treated as follows:

    a. If an identifier is described as alphanumeric edited, the INSPECT statement treats the contents of the identifier as a character-string.

    b. If an identifier is described as alphanumeric edited, numeric edited, or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (refer back to rule a) and the INSPECT statement had been written to reference the redefined data item.

    c. If an identifier is described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and rule b had been applied. (See 6.6.20, MOVE statement.)

12. In rules 13 through 20, 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.

13. 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 each properly matched occurrence of literal-3 is replaced by literal-4 (formats 2 and 3).

14. The comparison operation to determine the occurrences of literal-1 to be tallied 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 charcters, 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 or replacing takes place as described in rules 17 through 19. 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 charcter 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 1-character operand participates in the cycle described in rules 14a through 14d, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied charcter 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.

15. The comparison operation defined in rule 14 is affected by the BEFORE AND AFTER phrases as follows:

    a. If the BEFORE and 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 rule 14.

    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 that 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 rule 14 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-1, 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 that 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 the first occurrence is determined before the first cycle of the comparison operation described in rule 14 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.

NOTE:

Rules 16 and 17 pertain to format 1 only.

16. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.

17. 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 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 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 1 for each character matched (rule 14e) within the contents of the data item referenced by identifier-1.

NOTE:

Rules 18 and 19 pertain to format 2 only.

18. The required words ALL, LEADING, and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

19. The rules for replacement are as follows:

    a. When the CHARACTERS phrase is specified, each character matched (rule 14e) 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.

NOTE:

Rule 20 pertains to format 3 only.

20. 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 rules given for matching and counting apply to the format 1 statement and the rules given for matching and replacing apply to the format 2 statement.

Example 1:

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.
word = ANALYST, count = 0, count-1 = 1.

Example 2:

INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING "A" BY "E" AFTER INITIAL "L".

where:

word = CALLAR, count = 2, word = CALLAR.
word = SALAMI, count = 1, word = SALEMI.
word = LATTER, count = 1, word = LETTER.

Example 3:

INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

where:

word = ARXAX, word = GRXAX.
word = HANDAX, word HGNDGX.

Example 4:

INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J" REPLACING ALL "A" BY "B".

where:

word = ADJECTIVE, count = 6, word = BDJECTIVE.
word = JACK, count = 3, word = JBCK
word = JUJMAB, count = 5, word = JUJMBB.

Example 5:

INSPECT word REPLACING ALL "X" BY "Y", "B" BY "Z", "W" BY "Q" AFTER INITIAL "R".

where:

word = RXXBQWY, word = RYYZQQY.
word = YZACDWBR, word = YZACDWZR.
word = RAWRXEB, word = RAQRYEZ.

Example 6:

INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

where:

word before: 1 2 X Z A B C D
word after: B B B B B A B C D

6.6.19. MERGE Statement

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.

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                                            }
```

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 defined implicitly or explicitly as having sequential organization in the FILE-CONTROL paragraph and 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 records 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 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:

   - The data items identified by KEY data-names must be described in records associated with file-name-1.

   - KEY data-names may be qualified.

   - The data items identified by KEY data-names must not be variable-length items.

   - If file-name-1 has more than one record description, the data items identified by KEY data-names need be described in only one of the record descriptions.

   - None of the data items identified by KEY data-names can be described by an entry that either contains an OCCURS clause or is subordinate to an entry containing 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. The file-names specified in the USING phrase must not exceed 15.

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.

9. 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.

10. The data-names following the word KEY are listed from left to right in the MERGE statement in order of decreasing significance disregarding 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.

    - 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.

    - When the DESCENDING 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.

11. The collating sequence that applies to the comparison of the nonnumeric key data items specified is determined in the following order of precedence:

    - First, the collating sequence established by the COLLATING SEQUENCE phrase, if specified, in that MERGE statement.

    - Second, the collating sequence established as the program collating sequence.

12. 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 rules for procedural statements within the output procedure are as follows:

    - 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.

    - The output procedures must not contain any SORT, MERGE, or CALL statements.

    - 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.

13. 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.

14. Segmentation (Section 11) can be applied to programs containing the MERGE statement under the following conditions:

    - If the MERGE statement appears in a section that is not in an independent segment, the output procedure referenced by that MERGE statement must appear:

        a. totally within nonindependent 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 nonindependent segments; or

        b. wholly within the same independent segment as the MERGE statement.

15. 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.

16. 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.

17. 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.

18. The mode specified in the implementor-name of the ASSIGN clause for file-name-2, file-name-3, file-name-4, or file-name-5 must be the same as the mode specified for file-name-1.

6.6.20. MOVE Statement

Function:

The MOVE statement transfers data to one or more data areas in accordance with the rules of editing.

Format 1:

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

Format 2:

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

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 5.3.3.5, USAGE clause.)

5. 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 6.5.3. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements.

6. 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 area.

7. Any MOVE operation 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, or alphanumeric edited (5.3.3.4). 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 noninteger numeric literal or a noninteger 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 rule 8.

8. 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 in 2.5. 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 (5.3.3.6), 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 in 2.5, except where zeroes are replaced because of editing requirements.

      - When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item (5.3.3.6). The representation of the sign is converted as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

      - When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

      - When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

   c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place (2.5). If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right.

9. Any move that is not an elementary move is treated as 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 rule 15 of the OCCURS clause (5.3.3.7).

10. Table 6-6 summarizes the validity of the various types of MOVE statements. The notes indicate the rules that prohibit the move or describe the behavior of a legal move.

Table 6—6. Permissible MOVE Statement Data Transfers

Source Field	Receiving Field									
	GR	AL	AN	ED	BI	NE	ANE	ID	EF	IF
Group (GR)	Y	Y	Y	Y①	Y①	Y①	Y①	Y①	Y①	Y①
Alphabetic (AL)	Y	Y	Y	N	N	N	Y	N	N	N
Alphanumeric (AN)	Y	Y	Y	Y④	Y④	Y④	Y	Y④	Y④	Y④
External decimal (ED)	Y①	N	Y②	Y	Y	Y	Y②	Y	Y	Y
Binary (BI)	Y①	N	Y②	Y	Y	Y	Y②	Y	Y	Y
Numeric edited (NE)	Y	N	Y	N	N	N	Y	N	N	N
Alphanumeric edited (ANE)	Y	Y	Y	N	N	N	Y	N	N	N
ZEROS (numeric or alphanumeric)	Y	N	Y	Y③	Y③	Y③	Y	Y③	Y③	Y③
SPACES (AN)	Y	Y	Y	N	N	N	Y	N	N	N
ALL "character"	Y	Y	Y	Y⑤	Y⑤	Y⑤	Y	Y⑤	N	N
Numeric literal	Y①	N	Y②	Y	Y	Y	Y②	Y	Y	Y
Nonnumeric literal	Y	Y	Y	Y⑤	Y⑤	Y⑤	Y	Y⑤	N	N
Internal decimal (ID)	Y①	N	Y②	Y	Y	Y	Y②	Y	Y	Y
External floating-point (EF)	Y①	N	N	Y	Y	Y	N	Y	Y	Y
Internal floating-point (IF)	Y①	N	N	Y	Y	Y	N	Y	Y	Y
Floating-point literal	Y①	N	N	Y	Y	Y	N	Y	Y	Y

LEGEND:

Y - Denotes valid move
N - Denotes invalid move

NOTES:

① Move without conversion (like AN to AN)
② Only if the decimal point is at the right of the least significant digit
③ Numeric move
④ The alphanumeric field is treated as an ED (integer) field
⑤ The literal must consist only of numeric characters

6.6.21. MULTIPLY Statement

Function:

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

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]
```

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 fixed-point receiving data items of a given statement aligned on their decimal points, must not contain more than 18 digits.

4. See 6.5.1, ROUNDED phrase; 6.5.2, SIZE ERROR phrase; 6.5.4, arithmetic statements; 6.5.5, overlapping operands; and 6.5.6, multiple results in arithmetic statements.

5. 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.

6. When format 2 is used, the value of identifier-1 or literal-1 is muiltiplied by identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

6.6.22. ON Statement

Function:

The ON statement is a conditional statement that specifies both the condition to be met and the statements to be executed.

Format:

```
ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]

    {statement-1   } [ELSE {statement-2   }] .
    {NEXT SENTENCE } [     {NEXT SENTENCE }]
```

Rules:

1. Integer-1, integer-2, and integer-3 are positive numeric literals.
2. Statement-1 and statement-2 represent imperative statements.
3. The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal period of the sentence.
4. A counter is associated with each ON statement. Each time the path of control reaches the ON statement, the counter is advanced by one and the count condition is evaluated. Statement-1 is executed when the value of the counter is equal to integer-1 or integer-1 + (m * integer-2), but less than integer-3 (where m is any positive integer or zero). If the counter is not equal, statement-2 is executed.
5. If the ELSE phrase is omitted, or ELSE NEXT SENTENCE is specified and the counter is unequal, statement-1 is ignored and control passes to the sequence following the ON statement.
6. When integer-3 is not specified, no upper limit is assumed.
7. When integer-2 is omitted, but integer-3 is specified, integer-2 is assumed to have the value 1.
8. When integer-2 and integer-3 are both omitted, statement-1 is executed only once.

6.6.23. OPEN Statement

Function:

The OPEN statement initiates the processing of files. It also performs checking and writing of labels and other input/output operations.

Format 1 (Sequential and SAM* Files):

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

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

Format 2 (Relative, Indexed, and ISAM Files):

```
OPEN  ( INPUT file-name-1 [,file-name-2] ...   )  ...
      < OUTPUT file-name-3 [,file-name-4] ...  >
      ( I-O file-name-5 [,file-name-6] ...     )
```

Rules:

1. The OPEN statement must not reference a sort or merge file.

2. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

3. The successful execution of an OPEN statement makes the associated record area available to the program, but does not obtain or release the first data record.

4. 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.

   For ISAM files, the OPEN OUTPUT statement indicates that the file is loaded or extended. The creation of a file (LOAD) is assumed unless the file already exists, in which case file extension is implied.

5. An OPEN statement must be successfully executed before execution of any of the permissible input/output statements.

6. Table 6–7 indicates the permissible input/output statements for each OPEN mode for the various file organizations and access modes.

7. If standard system 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 system labels to be checked in accordance with the system-specified conventions for input label checking.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the system labels to be written in accordance with the system-specified conventions for output label writing.

8. For files being opened with the INPUT or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. 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 so the next executed format 1 or format 2 READ statement for the file will result in an AT END condition.

9. When the I–O phrase is specified and the LABEL RECORDS STANDARD clause is present, the execution of the OPEN statement includes the following steps:

   a. The system labels are checked in accordance with the system-specified conventions for input/output label checking.

   b. The new system labels are written in accordance with the system-specified conventions for input/output label writing.

Table 6—7. Permissible Input/Output Statement for Each OPEN Mode

File Organization	File Access Mode	Statement	OPEN Mode			
			INPUT	OUTPUT	I-O	EXTEND
Sequential and SAM	Sequential	READ	X		X	
		WRITE		X		X
		REWRITE			X	
Relative, Indexed, and ISAM	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*	

*Not permitted for ISAM files.

NOTE:

Rules 10 through 26 pertain to sequential and SAM files only.

10. The REVERSED and NO REWIND phrases apply only to single REEL/UNIT tape files.

11. The I-O phrase can be used only for mass storage files.

12. The EXTEND phrase can be used only for sequential files assigned to tape or mass storage devices, and for SAM files.

13. The EXTEND phrase must not be specified for multiple file reels. (See the I-O-control paragraph, 4.4.2.)

14. The files referenced in the OPEN statement need not all have the same organization or access.

15. Prior to the successful execution of an OPEN statement for a given file, no statement (except a SORT statement with the USING or GIVING phrase) can be executed that references the file, either explicitly or implicitly.

16. 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.

17. 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.

18. 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.

19. The files referenced in the OPEN statement need not all have the same organization or access.

20. 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.

21. 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, 6.6.25.)

22. The REVERSED phrase will be ignored if it does not apply to the storage media on which the file resides.

23. If the storage medium for the file permits reverse processing, the following rules apply:

    a. When neither the REVERSED nor the EXTEND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

    b. When the REVERSED phrase is specified, the file is positioned at its end by execution of the OPEN statement.

24. 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.

25. 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.

NOTE:

Rules 26 through 30 pertain to relative, indexed, and ISAM files.

26. The files referenced in the OPEN statement need not all have the same organization or access.

27. 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.

28. 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.

29. 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.

30. 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.

6.6.24. PERFORM Statement

Function:

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly when execution of the specified procedure is complete.

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   }                 ]
```

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

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    }                        ]           ]
```

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:

   - the identifier in the associated FROM and BY phrase must be an integer data item;
   - the literal in the associated FROM phrase must be a positive integer; and
   - the literal in the associated BY phrase must be a nonzero integer.

5. If an index-name is specified in the FROM phrase, then:

   - the indentifier in the associated VARYING or AFTER phrase must be an integer data item;
   - the identifier in the associated BY phrase must be an integer data item; and
   - 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 in 6.4.

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.

9. The data items referenced by identifier-4, identifier-7, and identifier-10 must not have a zero value.

10. 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.

11. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1, except as indicated in rules 15, 16, and 17. 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:

    - 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.

    - 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.

    - If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

    - 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.

12. 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.

13. If control passes to the procedures mentioned in rule 12 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.

14. 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.

15. 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. The value of integer-1 or the initial contents of identifier-1 may not exceed 32,767. 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.

16. 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.

17. 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 FORM (current value) phrases also refers to index-names. When index-name appears in a VARYING or AFTER phrase, it is initialized and subsequently augmented 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 in the following paragraphs.

    a. In format 4, when one identifier is varied , Figure 6–2, 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.

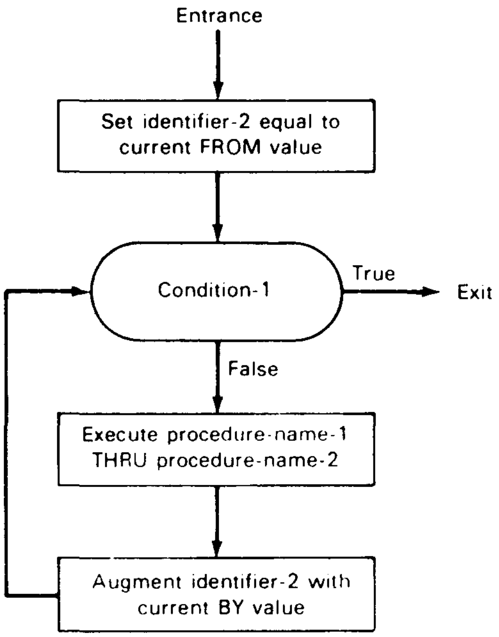

Entrance
Set identifier-2 equal to current FROM value
Condition-1
True
Exit
False
Execute procedure-name-1 THRU procedure-name-2
Augment identifier-2 with current BY value

Figure 6—2. Flowchart for the VARYING Phrase Having One Condition

b. In format 4, when two identifiers are varied (Figure 6–3), 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 reevaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycle continues 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.

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.

c. In format 4, when three identifiers are varied (Figure 6–4), 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.

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.

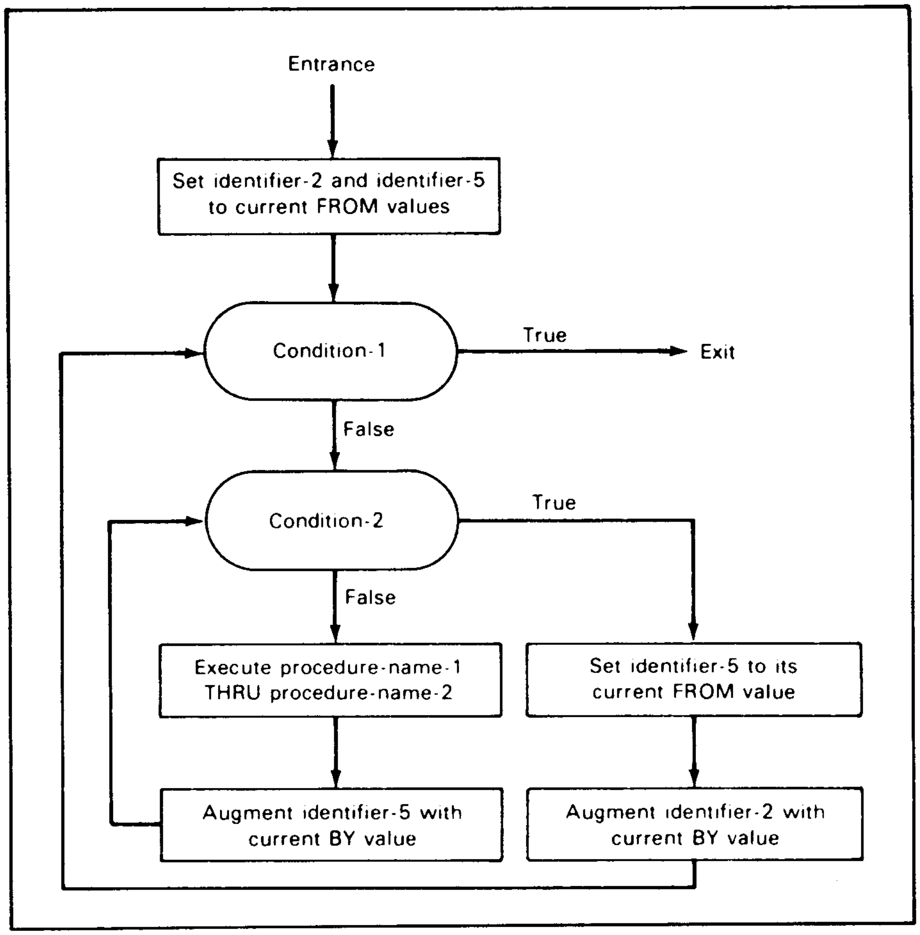

Entrance
Set identifier-2 and identifier-5 to current FROM values
Condition-1
True
Exit
False
Condition-2
True
False
Execute procedure-name-1 THRU procedure-name-2
Set identifier-5 to its current FROM value
Augment identifier-5 with current BY value
Augment identifier-2 with current BY value

Figure 6—3. Flowchart for the VARYING Phrase Having Two Conditions

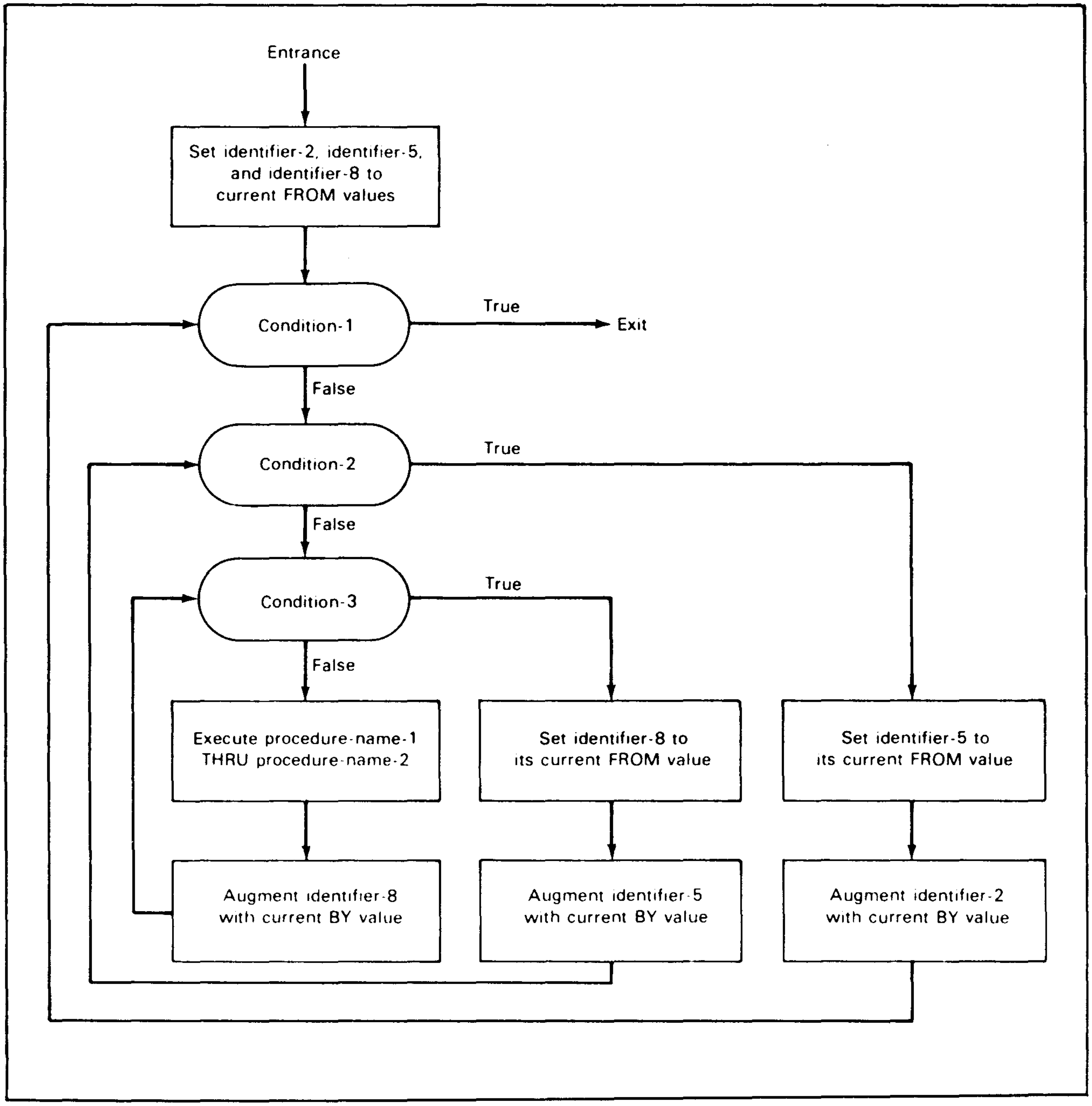
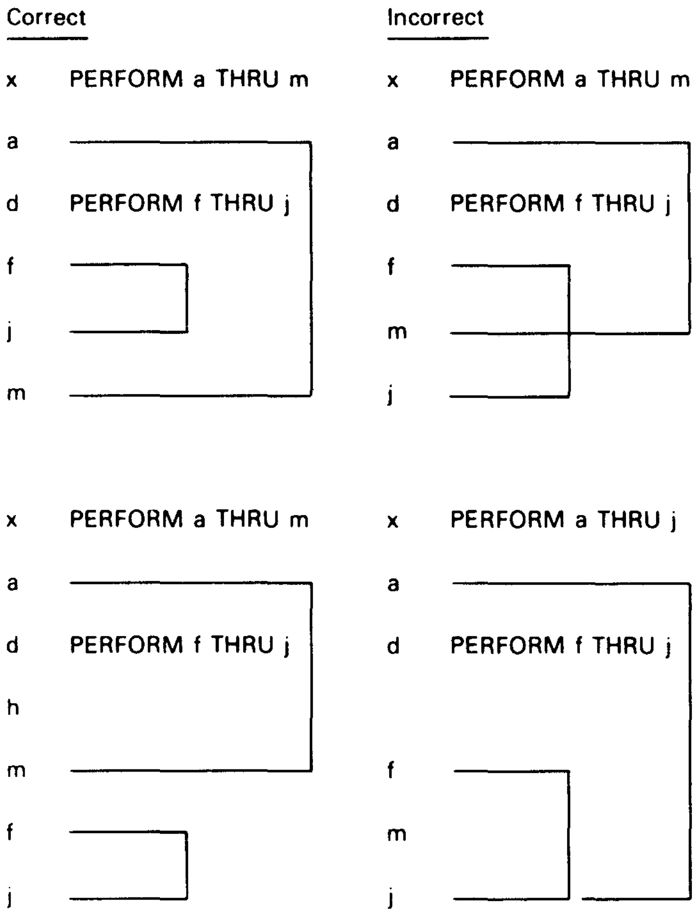

Entrance
Set identifier-2, identifier-5, and identifier-8 to current FROM values
Condition-1
True
Exit
False
Condition-2
True
False
Condition-3
True
False
Execute procedure-name-1 THRU procedure-name-2
Set identifier-8 to its current FROM value
Set identifier-5 to its current FROM value
Augment identifier-8 with current BY value
Augment identifier-5 with current BY value
Augment identifier-2 with current BY value

Figure 6—4. Flowchart for the VARYING Phrase Having Three Conditions

18. If a sequence of statements referenced 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. For example:



19. 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 or paragraphs wholly contained in one or more nonindependent segments

b. Sections or paragraphs wholly contained in a single independent segment

20. A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sectioins whose execution is caused within that range, only one of the following:

a. Sections or paragraphs wholly contained in one or more nonindependent segments

b. Sections or paragraphs wholly contained in the same independent segment as that PERFORM statement

6.6.25. READ Statement

Function:

The READ statement makes available the next logical or specified record from a file.

Format 1 (Sequential and SAM* Files):

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

Format 2 (Relative, Indexed, and ISAM* Files):

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

Format 3 (Relative and ISAM* Files):

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

Format 4 (Indexed Files Only):

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

Rules:

1. The associated file must be open in the INPUT or I-O mode at the time the statement is executed. (See the OPEN statement, 6.6.23.)

2. A record is available to the object program immediately after the execution of the READ statement.

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 8.2.3.)

4. 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.

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 is 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.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

8. If, at the time of execution of a format 1 or format 2 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 or format 2 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 8.2.3, I-O status description.)

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 8.2.4.)

    b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any format 1 USE procedure specified for this file is not executed.

    c. If the AT END phrase is not specified, a format 1 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 that 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.

NOTE:

Rules 12 through 16 pertain to sequential and SAM files only.*

12. The AT END phrase must be specified if no applicable format 1 USE procedure is specified for file-name.

13. 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 pointer is advanced to the next record in the file and then that record is made available.

14. 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.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

15. 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 4.4.1, FILE-CONTROL paragraph; 6.6.23, OPEN statement; 6.6.41, USE statement; and 8.2.3, I-O status.) Execution of the program then proceeds as specified in rule 10.

16. When the AT END condition is recognized, a READ statement for that file must not be executed until a successful CLOSE statement followed by a successful OPEN statement for that file is executed.

17. Format 1 must be used for all files in sequential access mode.

18. For printer-destined files (files assigned to PRINTER or defined with an FC, UC, or VC mode in the implementor-name of the ASSIGN clause), the READ statement referencing this file does not make available any record that contains only vertical positioning control information. (See 8.3.3.)

NOTE:

Rules 19 through 33 pertain to relative, indexed, and ISAM files.*

19. Format 2 must be used for all files in sequential access mode. The NEXT phrase must be specified for files in dynamic access mode when records are to be retrieved sequentially.

20. Format 3 is used for relative or ISAM* files in random access mode or in dynamic access mode when records are to be retrieved randomly.

21. Format 4 is used for indexed files in random access mode or in dynamic access mode when records are to be retrieved randomly.

22. The KEY phrase may be specified only for indexed files. Data-name must be the name of a data item specified as a record key associated with file-name. Data-name may be qualified.

23. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

24. The record to be made available by a format 2 READ statement is determined as follows:

- For relative or ISAM* files:

  a. The record pointed to by the current record pointer (see 8.2.2) is made available if 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, possibly 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.

- For indexed files:

  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.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

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.

25. When the AT END condition has been recognized, a format 2 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; or

    c. a successful format 3 or format 4 READ statement for that file.

26. For a file for which dynamic access mode is specified, a format 2 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from the file as described in rule 24.

27. If the RELATIVE KEY phrase is specified for a relative file, the execution of a format 2 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

28. For a relative file, the execution of a format 3 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 8.2.5, INVALID KEY condition.)

29. 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 that create such duplicate values.

30. For an indexed file, if the KEY phrase is specified in a format 4 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 2 READ statements for the file until a different key of reference is established for the file.

31. If the KEY phrase is not specified in a format 4 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 2 READ statements for the file until a different key of reference is established for the file.

32. Execution of a format 4 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 8.2.5.)

33. For an ISAM* file, the execution of a format 3 READ statement causes the record key 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 8.2.5, INVALID KEY condition.)

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

6.6.26. RECEIVE Statement

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 (MCS). The RECEIVE statement allows for a specific imperative statement when no data is available.

Format:

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

Rules:

1. Cd-name must reference an input CD.

2. 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 5.6.1.)

3. 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.

4. 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.

5. 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 rule 6), 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 subqueues is unknown to the MCS, control passes to the next executable statement whether or not the NO DATA phrase is specified. (See Table 5-11.)

6. The data items identified by the input CD (SYMBOLIC SOURCE, TEXT LENGTH, END KEY, STATUS KEY) are appropriately updated by the MCS at each execution of a RECEIVE statement. (See 5.6.1.)

7. 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.

8. 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. The remainder of the message can be transferred to the area referenced by identifier-1 with subsequent RECEIVE statements referring to the same queue and subqueues. The remainder of the message, for the purposes of applying rules 8a, 8b, and 8c, is treated as a new message.

9. 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 and subqueues. The remainder of the segment, for the purposes of applying rules 9a, 9b, and 9c, 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 rule 9.

10. 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.

11. After the execution of a STOP RUN statement, the remaining portion of a message partially obtained in that run unit is lost.

6.6.27. RELEASE Statement

Function:

The RELEASE statement transfers records to the initial phase of a SORT operation.

Format:

```
RELEASE record-name [FROM identifier]
```

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 (SD) entry contains record-name. (See 6.6.33, SORT statement.)
2. Record-name must be the name of a logical record in the associated SD entry and may be qualified.
3. Record-name and identifier must not refer to the same storage area.
4. The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.
5. 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.
6. After the execution of the RELEASE statement, the logical record is no longer available in the record area unless the associated sort 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 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 that were placed in it by the execution of RELEASE statements.

6.6.28. RETURN Statement

Function:

The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records during a MERGE operation.

Format:

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

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. 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.

5. 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.

6. 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 is returned and immediately before it is moved to the data item.

7. When the INTO phrase is used, the data is available in both the input record area and the data area associated with identifier.

8. If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs. When the AT END condition occurs, the contents of the record areas associated with the file are undefined. After the execution of the imperative-statement in an AT END phrase, no RETURN statement may be executed as part of the current output procedure.

6.6.29. REWRITE Statement

Function:

The REWRITE statement logically replaces a record existing in a mass storage file.

Format 1 (Sequential and SAM* Files):

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

Format 2 (Relative, Indexed, and ISAM* Files):

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

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 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 6.6.23, OPEN statement.)

4. 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.

5. 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.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

6. 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.

7. The current record pointer is not affected by the execution of a REWRITE statement.

8. **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 8.2.3, I-O status description.)**

9. For sequential files, the last input/output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The operating system logically replaces the record that was accessed by the READ statement.

NOTE:

Rules 10 through 20 pertain to relative, indexed, and ISAM files only.*

10. The INVALID KEY phrase must not be specified for a REWRITE statement that references a relative file in sequential access mode.

11. The INVALID KEY phrase must be specified in the REWRITE statement for relative files in the random or dynamic mode and for indexed or ISAM* files in all access modes if an appropriate USE procedure is not specified for the file.

12. For files in the sequential access mode, the last input/output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement. The operating system logically replaces the record that was accessed by the READ statement.

13. For a relative file accessed in either random or dynamic access mode, the operating system logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists. (See 8.2.5.) The updating operation does not take place and the data in the record area is unaffected.

14. For an indexed 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.

15. For an indexed file in the random or dynamic access mode, the record to be replaced is specified by the prime record key data item.

16. 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.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

17. For an indexed file, 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;

    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 8.2.5.)

18. For an ISAM* file in the sequential access mode, the record to be replaced is specified by the value contained in the record key. When the REWRITE statement is executed, the value contained in the record key data item of the record to be replaced must be equal to the value of the record key of the last record read from this file.

19. For an ISAM* file in the random or dynamic access mode, the record to be replaced is specified by the record key data item.

20. For an ISAM* file, the INVALID KEY condition exists when:

    a. the access mode is sequential and the value contained in the record key data item of the record to be replaced is not equal to the value of the record key of the last record read from this file; or

    b. the value contained in the record key data item does not equal that of any record stored in the file.

    The updating operation does not take place and the data in the record area is unaffected. (See 8.2.5.)

6.6.30. SEARCH Statement

Function:

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

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         }]
```

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

Format 2:

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

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

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

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

NOTE:

The required relational character = is not underlined to avoid confusion with other symbols.

Rules:

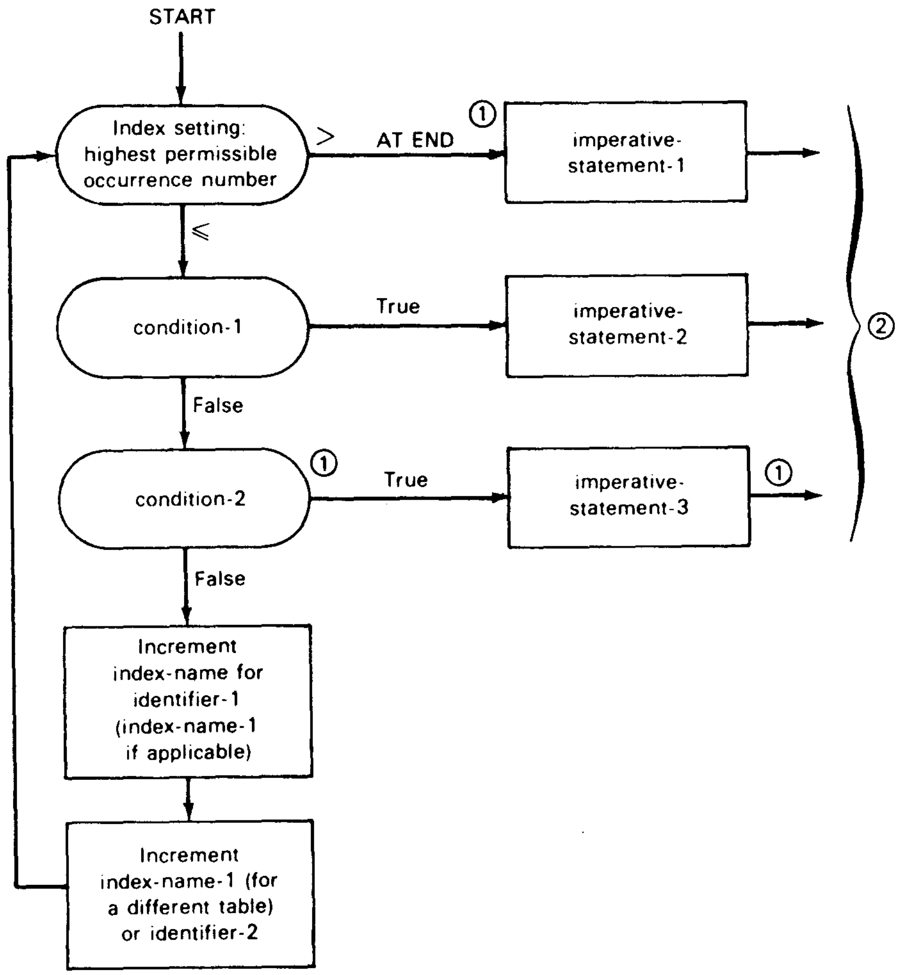
1. In both formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY clause. The description of identifier-1 in format 2 must also contain the KEY IS phrase in its OCCURS clause.

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 6.4. Refer to Figure 6-5 for the logic of a SEARCH statement containing two WHEN phrases.

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 indexes 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.


START
Index setting: highest permissible occurrence number
>
AT END
①
imperative-statement-1
⩽
condition-1
True
imperative-statement-2
②
False
condition-2
①
True
imperative-statement-3
①
False
Increment index-name for identifier-1 (index-name-1 if applicable)
Increment index-name-1 (for a different table) or identifier-2

NOTES:

① These operations are options included only when specified in the SEARCH statement.

② Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

Figure 6—5. Flowchart for Format 1 Search Operation Containing Two WHEN Phrases

5. 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 5.3.3.7, 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 (as explained in 5.3.3.7), the SEARCH statement evaluates 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 step a. 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.

6. 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 keys referenced in the WHEN clause are sufficient to identify a unique table element.

7. If format 2 of the SEARCH statement is used, a binary search operation takes place. The initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation so that it is never set to a value that exceeds the value corresponding to the last element of the table or is less than the value corresponding to the first element of the table. The length of the table is discussed in the OCCURS clause (5.3.3.7). 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.

8. After execution of an imperative-statement that does not terminate with a GO TO statement, control passes to the next executable sentence.

9. 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.

10. 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.

11. 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 not, 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 1 at the same time as the index referenced by the index-name associated with identifier-1 is incremented.

12. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a 2- or 3-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 2 or 3-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.

6.6.31. SEND Statement

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 (MCS).

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 } ADVANCING { {integer     } [LINES] } ]
    [                   { PAGE                   } ]
```

Rules:

1. Cd-name must reference an output CD.

2. Identifier-2 must reference a 1-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. Integer or the value of the data item referenced by identifier-3 may be zero, but may not exceed 255.

5. When a receiving communication device (printer, display screen, teletypewriter terminal, 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; and

   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.

6. When a receiving communication device (another program, 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.

7. 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 leftmost 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 Table 5-11.)

8. 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 5.6.2.)

9. The effect of having special control characters within the contents of the data item referenced by identifier-1 is the user's responsibility.

10. 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.

11. During the execution of the run unit, a portion of a message not terminated by an EMI or EGI is not sent to a destination, since the message does not logically exist for the MCS.

    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.

12. 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.

13. When an incomplete message is sent (format 1 or format 2 with ESI), the output CD referenced by the SEND statement can only be used to add to or complete the message. Until a message is completed, the contents of data-name-5 (SYMBOLIC DESTINATION table) cannot be changed.

NOTE:

Rules 14 through 19 pertain to format 2 only.

14. 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 identifier-2 is	then the contents of 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 treated as an EMI

    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.

15. 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 is treated by the MCS as the equivalent of an EMI.

    The MCS recognizes these indications and maintains group, message, and segment control.

16. 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.

17. 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.

18. 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.

19. On a device where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing will be provided to act as if the user had specified AFTER ADVANCING 1 LINE.

20. 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 the BEFORE phrase is used, the message or message segment is represented on the communication device before vertical repositioning.

    c. If the AFTER phrase is used, the message or message segment is represented on the communication device after vertical repositioning.

    d. 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 to act as if the user had specified BEFORE or AFTER (depending upon the phrase used) ADVANCING 1 LINE.

6.6.32. SET Statement

Function:

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

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   }
```

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.

5. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY phrase of the OCCURS clause.

6. 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 6.6.30, SEARCH statement; and 6.6.24, PERFORM statement.)

7. 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.

8. 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.

9. Table 6-8 indicates the validity of various operand combinations in the SET statement and the applicable rule reference.

Table 6—8. Valid Uses of the Format 1 SET Statement

Sending Item	Receiving Item		
	Integer Data Item	Index-name	Index Data Item
Integer literal	No (rule 7c)	Valid (rule 7a)	No (rule 7b)
Integer data item	No (rule 7c)	Valid (rule 7a)	No (rule 7b)
Index-name	Valid (rule 7c)	Valid (rule 7a)	Valid (rule 7b)*
Index data item	No (rule 7c)	Valid (rule 1a)*	Valid (rule 7b)*

*No conversion takes place

6.6.33. SORT Statement

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.

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                                           }
```

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. The file names specified in the USING phrase must not exceed 15.

4. File-name-2, file-name-3, and file-name-4 must be defined implicitly or explicitly as having sequential organization in the FILE-CONTROL paragraph and 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 records described for file-name-2, file-name-3, and file-name-4 must be equal to the actual size of the logical records described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, the programmer then must describe the corresponding records so that equal amounts of character positions are allocated for the corresponding records.

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, the data items identified by KEY data-names need by 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 that either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

6. The words THRU and THROUGH are equivalent.

7. 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.

8. Only one file-name from a multiple file reel can appear in the SORT statement.

9. In Level 1, the procedure division of a program contains one SORT statement and a STOP RUN statement in the first nondeclarative portion. Other sections consist of only the input and output procedures associated with the SORT statement.

10. 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.

11. 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 sort sequence is 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.

    c. When the KEY data-names are described as DISPLAY floating-point items, the sort sequence is based on the rules for comparison of alphanumeric operands in a relation condition.

12. 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.

13. 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. 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, MERGE, or CALL 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.

14. 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.

15. 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. 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, MERGE, or CALL 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.

16. If an output procedure is specified, control passes to it after file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of 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.

17. Segmentation as defined in Section 10 can be applied to programs containing the SORT statement in accordance with the following rules.

    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:

       ■ totally within nonindependent segments; or

       ■ 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:

       ■ totally within nonindependent segments; or

       ■ wholly within the same independent segment as that SORT statement.

18. 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 so 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.

19. 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 termines the processing of file-name-4. These implicit functions are performed so 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.

20. The mode specified in the implementor-name of the ASSIGN clause for file-name-2, file-name-3, or file-name-4 must be the same as the mode specified for file-name-1.

6.6.34. START Statement

Function:

The START statement provides a basis for logical positioning within a relative, indexed, or ISAM* file for subsequent sequential retrieval of records.

Format 1 (Relative and Indexed Files):

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

        [; INVALID KEY imperative-statement]
```

Format 2 (ISAM Files):

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

      [; INVALID KEY imperative-statement]
```

NOTE:

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

Rules:

1. File-name must be the name of a file with sequential or dynamic access.

2. Data-name may be qualified.

3. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed. (See 6.6.23, the OPEN statement.)

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 8.2.3, I-O status description.)

5. The INVALID KEY phrase must be specified if no applicable format 1 USE procedure is specified for file-name.

6. If the KEY phrase is not specified, the relational operator IS EQUAL TO is implied.

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

NOTE:

Rules 7 through 10 pertain to relative files only.

7. File-name must be the name of a relative file.

8. Data-name, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

9. The type of comparison specified by the relational operator in the KEY phrase occurs betwen a key associated with a record in the file referenced by file-name and a data item as specified in rule 10.

   - The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

   - 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 8.2.5, the INVALID KEY condition.)

10. The comparison described in rule 9 uses the data item referenced by the RELATIVE KEY clause associated with file-name.

NOTE:

Rules 11 through 17 pertain to indexed files only.

11. File-name must be the name of an indexed file.

12. If the KEY phrase is specified, data-name may reference a data item specified as a record key associated with file-name. The data-name may also reference any alphanumeric data item subordinate to the data-name specified as a record key associated with file-name. However, when a data-name references a subordinate record key data item, the leftmost character position of the data-name must correspond to the leftmost character position of that record key data item.

13. 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 rule 14. 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 6.4.1.1.2.)

    - The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

    - 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 8.2.5.)

14. If the KEY phrase is specified, the comparison described in rule 9 uses the data item referenced by data-name.

15. If the KEY phrase is not specified, the comparison described in rule 9 uses the data item referenced in the RECORD KEY clause associated with file-name.

16. Upon completion of the successful execution of the START statement, a key of reference is established and used in subsequent format 2 READ statements (6.6.25) as follows:

    - If the KEY phrase is not specified, the prime record key specified for file-name becomes the key of reference.

    - 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.

    - If the KEY phrase is specified, the 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.

17. If the execution of the START statement is not successful, the key of reference is undefined.

NOTE:

Rules 18 through 20 pertain to ISAM files only.*

18. File-name must be the name of an ISAM file.

19. Data-name, if specified, must be the data item specified in the RECORD KEY clause of the associated file control entry.

20. The type of comparison specified or implied occurs between a key associated with a record in the file referenced by file-name and the data item specified in the RECORD KEY clause of the associated file control entry. All nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE clause will have no effect on the comparison. (See 6.4.1.1 for a description of the comparison of nonnumeric operands.)

    - The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

    - 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 8.2.5.)

6.6.35. STOP Statement

Function:

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

Format:

$$\underline{\texttt{STOP}} \left\{ \begin{array}{l} \underline{\texttt{RUN}} \\ \texttt{literal} \end{array} \right\}$$

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

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.

4. If the RUN phrase is used, execution of the run unit is terminated and control is transferred to the operating system.

5. If STOP literal is specified, the literal is communicated to the operator and execution is suspended. When the operator acknowledges the communication, execution resumes at the next executable statement in sequence.

6.6.36. STRING Statement

Function:

The STRING statement provides juxtaposition of the partial or complete contents of two or more data items into a single data item.

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]
```

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.

6. 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.

7. Identifier-1, literal-1, identifier-2, literal-2, represent the sending items. Identifier-7 represents the receiving item.

8. Literal-3, identifier-3, indicate the characters 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.

9. When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit 1-character data item whose usage is DISPLAY.

10. 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 form 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 6.6.20, 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 characters specified by literal-3, or by the contents of identifier-3 are encountered. The characters specified by literal-3 or by the data item referenced by identifier-3 are not transferred.

    c. If DELIMITED BY SIZE is specified, 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.

11. If the POINTER phrase is specified, identifier-8 is explicitly available to the programmer, who is responsible for setting its initial value. The initial value must not be less than 1.

12. If the POINTER phrase is not specified, rules 13 through 16 apply as if the user had specified identifier-8 with an initial value of 1.

13. When characters are transferred to the data item referenced by identifier-7, the moves behave has 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.

14. After 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 contain data that was present before this execution of the STRING statement.

15. 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 1 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.

16. If the ON OVERFLOW phrase is not specified when the conditions described in rule 15 are encountered, control is transferred to the next executable statement.

6.6.37. SUBTRACT Statement

Function:

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

Format 1:

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

Format 2:

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

Format 3:

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

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, 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 6.5.4, the arithmetic statements.)

   a. In format 1, the composite of operands is determined by using all the fixed-point operands in a given statement.

   b. In format 2, the composite of operands is determined by using all the fixed-point 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. See 6.5.1, the ROUNDED phrase; 6.5.2, the SIZE ERROR phrase, 6.5.3, the CORRESPONDING phrase; 6.5.4, the arithmetic statements; 6.5.5, overlapping operands; and 6.5.6, multiple results in arithmetic statements.

6. 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. This process is repeated for each operand following the word FROM.

7. 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.

8. If format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

6.6.38. TRACE Statement

Function:

The TRACE statement initiates or terminates the trace function.

Format:

```
{READY} TRACE
{RESET}
```

Rules:

1. The trace function displays, on SYSLST (4.3.3), the name and line number of a section or paragraph at the start of its execution.

2. The statement READY TRACE initiates trace activity when the flow of program control passes to it.

3. The RESET TRACE statement terminates trace activity.

4. The TRACE statement may appear anywhere within the procedure division or the debugging packet.

6.6.39. TRANSFORM Statement

Function:

The TRANSFORM statement may be used to alter characters of an identifier according to a user-defined transformation rule or table.

Format 1:

```
TRANSFORM identifier-1 [,identifier-2] ... CHARACTERS

    FROM (identifier-3          ) TO (identifier-4          )
         {nonnumeric-literal-1  }    {nonnumeric-literal-2  }
         (figurative-constant-1 )    (figurative-constant-2 )
```

Format 2:

```
TRANSFORM identifier-1 [,identifier-2] ... CHARACTERS

    {ON} identifier-5
    {BY}
```

Rules:

1. All identifiers must be described either explicitly or implicitly as USAGE IS DISPLAY. Identifier-3, identifier-4, and identifier-5 may not be variable-length operands.

2. In format 1, identifier-3 and identifier-4 must not exceed 256 characters in length. The length of identifier-4 must be either equal to the length of identifier-3 or have a length of one character.

3. In format 1, all figurative constants are permitted except ALL nonnumeric literal.

4. In format 1, a character must not be duplicated in identifier-3 or in nonnumeric-literal-1.

5. In format 2, identifier-5 must have a length of 256 characters.

6. The least significant digit position of a signed, decimal numeric display item without a SEPARATE SIGN clause is treated as a single character, not a signed digit. The most significant digit position of a signed decimal numeric display item with a SIGN IS LEADING CHARACTER clause is treated as a single character, not a signed digit.

7. For format 1, the following rules, summarized in Table 6-9, describe the various FROM/TO combinations:

   - identifier-3 TO identifier-4

     identifier-3 TO nonnumeric-literal-2

     identifier-3 TO figurative-constant-2

     nonnumeric-literal-1 TO identifier-4

     nonnumeric-literal-1 TO nonnumeric-literal-2 nonnumeric-literal-1 TO figurative-constant-2

- If the FROM and the TO operands have the same length, any occurrence in identifier-1, identifier-2, etc, of a character (or the single character) in operand-1 is replaced by the character (or the single character) in the corresponding position of operand-2.

- If the FROM operand exceeds one character and the TO operand is only one character, any occurrence in identifier-1, identifier-2, etc. of any character in operand-1 is replaced by the single character in operand-2.

■ figurative-constant-1 TO identifier-4

figurative-constant-1 TO nonnumeric-literal-2

figurative-constant-1 TO figurative-constant-2

- Length of operand-1 and operand 2 is one character. Any occurrence in identifier-1 of the single character in operand-1 is replaced by the single character in operand-2.

8. For format 2, identifier-5 is a 0–255 binary-value positional translate table. Any character in identifier-1 with a binary value of 0 will be transformed to the character in the first position of identifier-5; any character in identifier-1 with a binary value of 1 will be transformed to the character in the second position of identifier-5, etc.

Table 6—9. Combination of FROM and TO Options in a TRANSFORM Statement (Part 1 of 2)

Operands	Rule	Identifier-1 Before	FROM	TO	Identifier-1 After
FROM figurative-constant-1 TO figurative-constant-2	All occurrences of figurative-constant-1 in the item represented by identifier-1 are replaced by figurative-constant-2. (Each operand must be a single character.)	1"2""3	QUOTE	ZERO	102003
FROM figurative-constant-1 TO nonnumeric-literal-2	All occurrences of figurative-constant-1 in the item represented by idnetifier-1 are replaced by nonnumeric-literal-2. (Each operand must be a single character.)	1 △2 △3	SPACE	"7"	17273
FROM figurative-constant-1 TO identifier-4	All occurrences of figurative-constant-1 in the item represented by identifier-1 are replaced by the item represented by identifier-2. (Each operand must be single character.)	1 △2 △3	SPACE	ALPHA (current value of ALPHA = B)	1B2B3
FROM nonnumeric-literal-1 TO figurative-constant-2	All occurrences of any character of nonnumeric-literal-1 in the item represented by identifier-1 are replaced by the single-character figurative-constant-2.	AB12X7P	"1234567890"	SPACE	AB △△X △P
FROM nonnumeric-literal-1 TO nonnumeric-literal-2	Nonnumeric-literal-1 and nonnumeric-literal-2 must be equal in length, or nonnumeric-literal-2 must be a single character. If the operands are equal in length, any character in the item represented by identifier-1 that is replaced by the character in the corresponding position of nonnumeric-literal-2.	ABCD12X	"ABCDEFGHIJ"	"1234567890"	123412X
	If nonnumeric-literal-2 is a single character, then all occurrences of any character of nonnumeric-literal-1 in the item represented by identifier-1 are replaced by the single character in nonumeric-literal-2.	AB21X73	"1234567890"	"L"	ABLLXLL

Table 6—9. Combination of FROM and TO Options in a TRANSFORM Statement (Part 2 of 2)

Operands	Rule	Identifier-1 Before	FROM	TO	Identifier-1 After
FROM nonnumeric-literal-1 TO identifier-4	The two operands must be equal in length, or identifier-4 must represent a single-character item. If the operands are equal in length, any character in the item represented by identifier-1 that is equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of the item represented by identifier-4.	1Δ2ΔDEF	"Δ12DEF"	BETA (current value of BETA = FED21Δ)	EFDF21Δ
	If identifier-4 is a single character, then all occurrences of any character of nonnumeric-literal-1 in the item represented by identifier-1 are replaced by the character represented by identifier-4.	ABC	ADE	GAMMA (current value of GAMMA=1)	1BC
FROM identifier-3 TO figurative-constant-2	All occurrences of any character of the item represented by identifier-3 in identifier-1 are replaced by the single character figurative-constant-2.	A12B	GAMMA (current value of GAMMA = ABC.)	QUOTE	"12"
FROM identifier-3 TO nonnumeric-literal-2	The two operands must be equal in length, or nonnumeric-literal-1 must be a single-character item. If the operands are equal in length, any character in the item represented by identifier-1 that is equal to a character in the item represented by identifier-3 is rpelaced by the character in the corresponding position of nonnumeric-literal-2.	ABCD	ALPHA (current value of ALPHA = A12B)	"DCBA"	DACD
	If nonnumeric-literal-2 is a single character, then all occurrences of any character of the item represented by identifier-3 in the item represented by identifier-1 are replaced by nonnumeric-literal-2.	ABCD	DELTA (current value of DELTA = ABCDEF)	"6"	6666
FROM identifier-3 TO identifier-4	Any character in the item represented by identifier-1 that is equal to a character in the item represented by identifier-3 is replaced by the character in the corresponding position of the item represented by identifier-4. Both operands must be of equal length. Each of the operands may contain one or more characters.	1AB4	ITEM-A (current value of ITEM-A = 1234)	ITEM-B (current value of ITEM-B = ABCD)	AABD

6.6.40. UNSTRING Statement

Function:

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format:

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

Rules:

1. Each literal must be a nonnumeric literal, or any figurative constant without the optional word ALL.

2. Identifier-1, identifier-2, identifier-3, identfier-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.

7. 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.

8. Identifier-1 represents the sending area.

9. Identifier-4 represents the data receiving area. Identifier-5 represents the receiving area for delimiters.

10. Literal-1 or the data item referenced by identifier-2 specifies a delimiter.

11. 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 characters.

12. The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1.

13. 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.

14. A figurative constant used as the delimiter represents 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 only one occurrence. This occurrence is moved to the receiving data item according to rule 19d.

15. 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.

16. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the computer character set.

17. Each literal-1 or the data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all the characters must be present in contiguous positions of the sending item and in the order given to be recognized as a delimiter.

18. 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 characters in the sending field are considered to be a single delimiter. No characters 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.

19. 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 rule 17.) 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 characters if any) treated as an elementary alphanumeric data item and are moved into the current receiving area according to the rules for the MOVE statement.

    d. If the DELIMITER IN phrase is specified, the delimiting characters 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. 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 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 referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The behavior described in rules 19b through 19f 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.

20. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

21. The contents of the data item referenced by identifier-10 will be incremented by 1 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.

22. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of the data item referenced by identifier-11 contain a value equal to its initial value plus the number of data receiving items acted upon.

23. 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. 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.

24. 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.

25. The evaluation of subscripting and indexing for the identifiers is as follows:

    a. Any subscripting or indexing associated with identifier-1, identifier-10, or 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.

6.6.41. USE Statement

Function:

- The USE ERROR PROCEDURE statement specifies procedures for input/output error handling that are in addition to the standard procedures provided by the input/output control system.

- The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

- The LABEL PROCEDURE statement specifies procedures for input/output tape label handling that are in addition to the standard label procedures provided by the input/output control system.

Format 1:

```
USE AFTER STANDARD {EXCEPTION}
                   {ERROR    }

     PROCEDURE ON ( file-name-1 |[,file-name-2] ...| )
                  ) INPUT                            (
                  < OUTPUT                           > .
                  ) I-O                              (
                  (|EXTEND|                          )
```

Format 2:

```
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                      ]
```

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
| Format 3:                                                  |
|                                                            |
|   USE {AFTER } STANDARD [BEGINNING] [FILE]                 |
|       {BEFORE}          [ENDING   ] [REEL]                 |
|                                                            |
|       LABEL PROCEDURE ON {file-name-1[,file-name-2] ...}.  |
|                          {INPUT                        }   |
|                          {OUTPUT                       }   |
└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Rules:

1. A USE statement 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 paragrahs 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. Appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. A sort or merge file may only be referenced in a format 2 USE statement.

5. 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 format 1 or 3 USE statement or to the procedures associated with such a USE statement.

6. 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.

NOTE:

Rules 7 through 11 pertain to format 1 only.

7. The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

8. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

9. A file-name may not be explicitly referred to in more than one format 1 USE statement.

10. 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 or INVALID KEY condition when the AT END phrase or INVALID KEY phrase, respectively, has not been specified in the input/output statement.

11. After execution of a USE procedure, control is returned to the invoking routine.

NOTE:

Rules 12 through 46 pertain to format 2 only.

12. Debugging sections, if specified, must appear together immediately after the DECLARATIVES header.

13. Except in the USE FOR DEBUGGING statement itself, there must be no reference to any nondeclarative procedure within the debugging section.

14. Statements appearing outside the set of debugging sections must not reference procedure-names defined within the set of debugging sections.

15. 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.

16. Procedure-names defined within debugging sections or debugging packets must not appear within USE FOR DEBUGGING statements.

17. Any given identifier, file-name, or procedure-name may appear in only one USE FOR DEBUGGING statement and may appear only once in that statement.

18. The ALL PROCEDURES phrase can appear only once in a program.

19. When the ALL PROCEDURES phrase is specified, procedure-name-1, procedure-name-2, ... must not be specified in any USE FOR DEBUGGING statement.

20. 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.

21. References to the special register DEBUG-ITEM are restricted to references from within a debugging section.

22. Procedure-names defined within debugging packets must not appear within USE FOR DEBUGGING statements.

23. In the following 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.

24. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

25. 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;

    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.

26. 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; and

    b. immediately after the execution of an ALTER statement that references procedure-name-1.

27. The ALL PROCEDURES phrase causes the effects described in rule 26 to occur for every procedure-name in the program, except those appearing within a debugging section.

28. 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. Immediately before the execution of a WRITE or REWRITE statement after the execution of any implicit move resulting from the presence of the FROM phrase

    b. For 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. For 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. For 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.

29. When identifier-1 is specified without the ALL REFERENCES OF phrases, that debugging section is executed at each of the following times:

    a. For a WRITE or REWRITE statement that explicitly references identifier-1, immediately before the execution of the statement and after the execution of any implicit move resulting from the presence of the FROM phrase

    b. For 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 identifier-1 and causes the contents of the referenced data item to be changed.

    If identifier-1 is specified in a phrase that is not executed or evaluated, the associated debugging section is not executed.

30. 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 that 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.

31. A reference to cd-name-1, file-name-1, identifier-1, or procedure-name-1 as a qualifier does not constitute reference to that item for the debugging described in the preceding rules.

32. 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)*.
```

**The size of DEBUG-CONTENTS ranges from 30 to 4096 characters depending on the size of the largest data item being monitored.*

33. 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 rules 34 through 46, immediately before control is passed to that debugging section. The contents of any data item not specified in the rules remain 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 as an alphanumeric-to-alphanumeric elementary move with no conversion of data from one form of internal representation to another.

34. The contents of DEBUG-LINE is the compiler-generated line number that identifies a particular source statement.

35. 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/indexes, if any, are not entered into DEBUG-NAME.

36. 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.

37. DEBUG-CONTENTS is a data item that is large enough to contain the data required by the following rules.

38. 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.

39. 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 the applicable procedure-name associated with the TO phrase of the ALTER statement.

    c. DEBUG-CONTENTS contains the applicable procedure-name associated with the TO phrase of the ALTER statement.

40. 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.

41. If reference to procedure-name-1 in the INPUT or OUTPUT phrase of a SORT statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the SORT statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains:

        - If the reference to procedure-name-1 is in the INPUT phrase of a SORT statement, SORT INPUT

        - If the reference to procedure-name-1 is in the OUTPUT phrase of a SORT statement, SORT OUTPUT

42. If the transfer of 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.

43. 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.

44. 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.

45. If references to file-name-1 cause the debugging section to be executed, then:

    a. DEBUG-LINE identifies the source statement that references file-name-1.

    b. DEBUG-NAME contains the name of file-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.

46. 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 rules 17 and 18).

NOTE:

Rules 47 through 55 pertain to format 3 only.

47. The files implicitly or explicitly referenced in a format 3 USE statement must all be magnetic tape files.

48. The same file-name can appear in a different specific arrangement of format 3. However, the appearance of a file-name in a USE statement must not cause simultaneous requests for execution of more than one USE procedure.

49. If the file-name phrase is used, the file description entry for file-name-1, file-name-2, etc, must specify a LABEL RECORDS ARE data-name clause.

50. If neither BEGINNING nor ENDING is specified, the designated procedures are executed for both beginning and ending labels.

    If neither REEL nor FILE is included, the designated procedures are executed for both reel and file labels.

51. If the INPUT or OUTPUT phrase is specified, the USE procedures do not apply respectively to input or output files that are described with the LABEL RECORDS ARE OMITTED or STANDARD clause.

52. The BEFORE phrase is not applicable to standard label procedures. If the BEFORE phrase is specified, it is treated as if the AFTER phrase were specified.

53. For files opened for input, the designated USE procedure is executed if a standard user label is encountered after the standard system label processing is completed. If more than one standard user label exists, they can be accessed by a GO TO MORE-LABELS statement. (See 6.6.16.) The USE LABEL procedure is not reentered if there are no more standard user labels to be processed.

54. For files opened for output, the designated USE procedure is executed after standard system label processing is completed. A standard user label is written after execution of the last statement in the associated USE procedure. A standard user label is also written upon execution of a GO TO MORE-LABELS statement, in which case, control is transferred to the beginning of the same USE procedure.

55. Input/output statements and the STOP literal statement are not allowed in the USE LABEL procedures except for the ACCEPT statement (other than from SYSCONSOLE or SYSIN) and DISPLAY statement.

6.6.42. WRITE Statement

Function:

The WRITE statement releases a logical record for an output or input/output file. It also provides control of the vertical positioning of each line on a page for sequential files.

Format 1 (Sequential and SAM Files):

```
WRITE record-name [FROM identifier-1]
    ┌{BEFORE} ADVANCING ( {identifier-2} [LINE ] )┐
    │{AFTER }           { {integer     } [LINES] }│
    │                   { {mnemonic-name}        }│
    └                   ( {PAGE         }        )┘
     ┌──────────────────────────────────────────────┐
     │ [;AT {END-OF-PAGE} imperative-statement]     │
     │      {EOP        }                           │
     └──────────────────────────────────────────────┘
```

Format 2 (Relative, Indexed, and ISAM Files):

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

Rules:

1. Record-name and the identifier of the FROM phrase must not refer to the same storage area.

2. The record-name is the name of a logical record in the file section of the data division and may be qualified.

3. The 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 or 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.

4. The result 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 rule 3.)

5. The current record pointer is unaffected by the execution of a WRITE statement.

6. 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 8.2.3, I-O status description.)

7. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

8. 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.

9. The execution of the WRITE statement releases a logical record to the operating system.

NOTE:

Rules 10 through 21 pertain to sequential and SAM files only.*

10. When mnemonic-name is specified, the name is associated with a particular feature SYSCHAN-n and is defined in the SPECIAL-NAMES paragraph of the environment division.

11. When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

12. Integer or the value of the data item referenced by identifier-2 may be zero, but may not exceed 255.

13. If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry for the associated file.

14. The words END-OF-PAGE and EOP are equivalent.

15. The ADVANCING mnemonic-name phrase cannot be specified when writing a record to a file whose file description entry contains any LINAGE clause, except the LINAGE IS SYSTEM LINES clause.

16. The associated file must be open in the OUTPUT or EXTEND mode at the time of the execution of this statement. (See 6.6.23, the OPEN statement.)

17. 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 is provided 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 to the line specified by SYSCHAN-n. (See 4.3.3, SPECIAL-NAMES paragraph.)

    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.

*Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

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.

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 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 device is repositioned to the first line of the next page as defined by the operating system. If PAGE has no meaning in conjunction with a specific device, then the compiler-generated code advances the device as if the user had specified BEFORE or AFTER (depending on the phrase used) ADVANCING 1 LINE.

18. 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.

19. 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. If the SYSTEM LINES option of the LINAGE clause is specified, an end-of-page condition occurs when the LINAGE-COUNTER equals or exceeds the line on which the operating system reports that the overflow line position of the vertical format buffer has been crossed. In these cases, 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. By definition, the automatic page overflow condition cannot arise when the SYSTEM LINES option of the LINAGE clause is specified. (See 5.3.1.6., the LINAGE clause.)

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.

20. For printer-destined files opened for output (files assigned to PRINTER or defined with mode FC, VC, or UC in the ASSIGN clause), a command to skip to the home-paper position is issued when the first WRITE statement is executed. If the first WRITE statement executed specifies an initial blank page, either by the AFTER PAGE phrase or by a record containing all blanks with the BEFORE PAGE phrase, that initial blank page is deleted from the output file (its function is accomplished by the command to skip to the home-paper position).

21. 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 8.2.3, I-O status description.)

    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.

22. At the end of a reel/unit of a multi-reel/unit output file, 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

NOTE:

Rules 23 through 25 pertain to relative, indexed, and ISAM files.*

23. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

24. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement. (See 6.6.23, the OPEN statement.)

25. 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, in 8.2.5, the INVALID KEY condition. (See 8.2.3, I-O status description.)

NOTE:

Rules 26 through 28 pertain to relative files only.

26. When a relative 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 causes a record to be released to the operating system. The first record has a relative record number of 1 and subsequent record released 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 related, is placed into the RELATIVE KEY data item by the operating system during execution of the WRITE statement.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

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 operating system by execution of the WRITE statement.

27. When a relative 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 operating system.

28. The INVALID KEY condition exists:

a. when the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record that already exists in the file; or

b. when an attempt is made to write beyond the externally defined boundaries of the file.

NOTE:

Rules 29 through 35 pertain to indexed files only.

29. The execution of the WRITE statement causes the contents of the record area to be released. The operating system utilizes the content of the record keys in such a way that subsequent access of the record key may be made based upon any of those specified record keys.

30. The value of the prime record key must be unique within the records in the file.

31. 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 rule 4.)

32. If sequential access mode is specified for the file, records must be released to the operating system in ascending order of prime record key values.

33. If random or dynamic access mode is specified, records may be released to the operating system in any program-specified order.

34. 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 nonunique only if the DUPLICATES phrase is specified for that data item. In this case, the operating system provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the operating system.

35. The INVALID KEY condition exists under the following circumstances:

- 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;

- 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;

- 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

- when an attempt is made to write beyond the externally defined boundaries of the file.

NOTE:

*Rules 36 through 40 pertain to ISAM** files only.*

36. The value of the record key must be unique within the records in the file.

37. The data item specified as the record key must be set by the program to the desired value prior to the execution of the WRITE statement. (See rule 4.)

38. When a file is being created, sequential or dynamic access mode must be specified. Records must be released to the operating system in ascending order of record key values, even if dynamic mode is specified.

39. If random or dynamic access mode is specified, records may be released to the operating system in any program-specified order.

40. The INVALID KEY condition exists under the following circumstances:

    - when sequential or dynamic access mode is specified for a file opened in the output mode and the value of the record key is not greater than the value of the record key of the previous record; or

    - when the file is opened in the output or I-O mode and the value of the record key is equal to the value of a record key of a record already existing in the file; or

    - when an attempt is made to write beyond the externally defined boundaries of the file.

6.6.43. *DEBUG Statement

Function:

The *DEBUG statement indicates the location of the program at which a debugging packet is to be executed. (See 12.4.3.4.)

Format:

```
*DEBUG procedure-name
```

Rules:

1. The word *DEBUG must begin at margin L; however, procedure-name may appear anywhere between margin A and margin R.

2. Procedure-name may be qualified.

3. Procedure-name may not appear within the group of debugging packets, nor may it appear in more than one *DEBUG statement.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

4. A *DEBUG statement is required as a header for each debugging packet.

5. Debugging packets are placed immediately behind the last source statement in the procedure division for compilation, but the debugging packets are executed at object time as though each packet appeared immediately following the referenced procedure-name in the program but before the source statements (procedure) associated with the procedure-name.

6. Statements in the debugging packets must not refer to the DEBUG-ITEM of a USE FOR DEBUGGING section.

7. Table Handling Summary

7.1. GENERAL

The table handling module provides a means of defining contiguous data items in a tabular form and accessing any item regardless of its position in the table.

Table handling Level 1 provides a capability for accessing items in up to 3-dimensional, fixed-length tables. This level also provides series options and the capability to vary the contents of indexes by an increment or decrement.

Table handling Level 2 provides a capability for accessing items in up to 3-dimensional, variable-length tables. This level also provides the additional facilities for specifying ascending or descending keys and for searching a dimension of a table for an item satisfying a specified condition.

7.2. LANGUAGE CONCEPTS

COBOL tables are defined structurally by including the OCCURS clause in the data description entries. The OCCURS clause specifies the number of times an item is to be repeated. An item described by an OCCURS clause is called a table element, and the name and description of the table element applies to each repetition or occurrence.

Because the data-name is the same for each occurrence of a table element, a reference to a desired occurrence can be made only by specifying the data-name of the table element with the occurrence number of the desired table element. The occurrence number is specified by either subscripting or indexing.

7.2.1. Table Definition

To define a 1-dimensional table, an OCCURS clause is written as part of the data description of the table element. The OCCURS clause, however, must not appear in the description of group items that contain the table element.

Example:

```
01  TABLE-1.
    02  TABLE-ELEMENT OCCURS 20 TIMES.
        03  NAME .........
        03  ADDRESS ......
```

Defining a 1-dimensional table within each occurrence of an element of another 1-dimensional table gives rise to a 2-dimensional table. To define a 2-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 3-dimensional table, the OCCURS clause should appear in the data description of the element of the table and in the description of two group items that contain the element. In COBOL, tables of up to three dimensions are permitted.

Example:

```
Ø1  CORPORATION-TABLE.
    Ø2  COMPANY-TABLE OCCURS 5 TIMES.
        Ø4  COMPANY-NAME PIC X(12).
        Ø4  DIVISION-TABLE OCCURS 1Ø TIMES.
            Ø6  DIVISION-CODE PIC X(4).
            Ø6  DEPARTMENT-TABLE OCCURS 1ØØ TIMES.
                Ø8  DEPARTMENT-CODE PIC 9(3).
                Ø8  EMPLOYEES PIC 9(4).
```

This example defines a table of one dimension for COMPANY-NAME, two dimensions for DIVISION-CODE, and three dimensions for DEPARTMENT-CODE and EMPLOYEES.

The table consists of 10,055 data items:

5 for COMPANY-NAME

50 for DIVISION-CODE

5000 for DEPARTMENT-CODE

5000 for EMPLOYEES

Within the table there are 5 occurrences of COMPANY-NAME. Within each COMPANY-NAME there are 10 occurrences of DIVISION-CODE, and within each DIVISION-CODE there are 100 occurrences of DEPARTMENT-CODE and EMPLOYEES.

7.2.2. References to Table Items

When referring to a table element, the reference must indicate the intended occurrence of the element. For access to a 1-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.

Occurrence numbers may be specified either by subscripting or by indexing. However, data-name subscript and index-name must not be mixed within a single reference to a table element that requires more than one occurrence number.

7.2.2.1. Subscripting

Subscripts are used only to refer to an individual element within a table of like elements that have not been assigned individual data-names.

Format:

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

A subscript is an integer that identifies the occurrence number of a particular table element. 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.

The subscript, or set of subscripts, is enclosed in parentheses and appears immediately following the space that terminates the data-name of the table element. When more than one subscript is specified within a pair of parentheses, each subscript must be separated from the next by a space, and the subscripts are written in the order of successively less inclusive dimensions of the data organization, that is, in the same order as the OCCURS clauses.

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.

7.2.2.2. 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 of the OCCURS clause in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name must be initialized before it is used as a table reference. An index-name can be given an initial value by a SET, SEARCH ALL, or format 4 PERFORM statement.

Format:

```
{data-name      } / {index-name-1 [{±} literal-2]}                  \
{condition-name } | {literal-1                   }                  |
                  | [,{index-name-2 [{±} literal-4]}               ] |
                  | [ {literal-3                   }               ] |
                  | [ [,{index-name-3 [{±} literal-6]}]           ] |
                  \ [ [ {literal-5                   }]           ] /
```

Direct indexing is specified by using an index-name like a subscript. Relative indexing is specified when the index-name is followed by the operator + or – and an unsigned integer numeric literal, all enclosed in parentheses following the space that terminates the data-name of the table element. The occurrence number resulting from relative indexing is determined by incrementing (+ operator) or decrementing (– operator), 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 that refers to an indexed table element, the value contained in the index referenced by the index-name associated with the table element must not correspond to a value less than 1 or 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.

7.2.2.3. Range Checking

Normally the values of subscripts and indices are not checked to ensure that the table reference is within the limits of the OCCURS clause, and results are unpredictable if the subscript or index is out of range. If the compile time parameter SUBCK=YES is specified, subscripts and indices are checked for validity. An out-of-range condition results in an object program termination with a CE58 error message. Refer to Appendix A for a description of the SUBCK parameter. Refer to the system messages operations reference handbook, UP-8076, for an explanation of the CE58 error message.

7.3. DATA DIVISION CONSIDERATIONS

There are two table-handling clauses in the data division – the OCCURS clause and the USAGE IS INDEX clause.

The OCCURS clause indicates the number of elements contained in a table, and also supplies information required for the application of subscripts or indexes. Format 1 of the OCCURS clause indicates the exact number of occurrences of a specified data-item. Format 2 of the OCCURS clause indicates that the item described by the OCCURS clause has a variable number of occurrences. This option gives the minimum and maximum number of occurrences and specifies the data item that controls the number of occurrences.

The length of a table element may not exceed 32,767 bytes. The maximum number of occurrences of a table element may not exceed 65,535.

The USAGE IS INDEX clause specifies that a data item is to be used as temporary storage for the values of an index.

A detailed description of the OCCURS clause is given in 5.3.3.7 and of the USAGE IS INDEX clause in 5.3.3.5.

7.4. PROCEDURE DIVISION CONSIDERATIONS

7.4.1. Table Handling Statements

The table handling statements for the procedure division consist of the SEARCH and SET statements.

The SEARCH statement is used to search a table for a table element that satisfies a specified condition and to adjust the associated index-name to point to that table element. Format 1 of the SEARCH statement is used to perform a serial search of a table. Format 2 is used to perform a nonserial search of a large data table.

The SET statement is used to change the value of an index-name or index data item, or to obtain the occurrence number which corresponds to the current value of an index-name. The index-names can then be used as reference points for table handling operations. Format 1 sets an integer data item, index-name, or index data item to a specified value. Format 2 increments or decrements the value of an index-name, to represent a new occurrence number.

A detailed description of the SEARCH statement is given in 6.6.30 and of the SET statement in 6.6.32.

7.4.2. Comparisons Involving Index-Name or Index Data Items

Relation tests involving index-name or index data items may be made as explained in 6.4.1.1.3.

7.4.3. Overlapping Operands in a SET Statement

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.

8. File Processing Summary

8.1. GENERAL

The organization of a file specifies the logical structure of the file and determines the technique to be used for processing of the file.

Files may be organized in a sequential form or in a nonsequential form. The organization of a file is established at the time the file is created and cannot subsequently be changed.

8.2. LANGUAGE CONCEPTS

8.2.1. File Organization and Access Methods

There are five types of file organization available – sequential, relative, indexed, SAM*, or ISAM*.

8.2.1.1. Sequential Organization

A sequential file 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.

Each record in a sequential 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.

8.2.1.2. Relative Organization

A relative file is stored only on mass storage devices and is accessed sequentially, dynamically, or randomly. Each record in a relative file is uniquely identified by an integer value greater than zero that specifies the logical ordinal position of the record in the file. The file may be considered as composed of a serial string of areas, each capable of holding a logical record. Each area is denominated by a relative record number, and records are stored and retrieved based on this number.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all currently existing records 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. 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 nine 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.

8.2.1.3. Indexed Organization

An indexed file is a mass storage file in which data records are identified by one or more keys within those records. The position of each logical record in the file is determined by indexes created and maintained by the operating system. The indexes are based on keys provided by the data items named in the RECORD KEY and ALTERNATE RECORD KEY clauses of the file control entry for that file.

For 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. For retrieval of records, the value of the prime record key provides a logical path to the data records. The values of alternate record keys, if specified, provide alternate access paths. The values of alternate record keys need not be unique if the DUPLICATE phrase is specified.

An indexed file can be accessed sequentially, dynamically, or randomly. Sequential access provides access to the data 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. 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.

8.2.1.4. SAM* Organization

A SAM file is a sequentially organized mass storage file supported by the OS/3 disk sequential access method (disk SAM). This file organization is a Sperry Univac extension to support user's existing sequential mass storage files created by the OS/3 disk SAM data management.

In concept, a SAM file has all the characteristics of a mass storage file with sequential organization; however, the file structure of a SAM file on a mass storage device differs considerably from the file structure of a sequential mass storage file created by the OS/3 MIRAM (unkeyed) data management. By using the SAM file organization, a compatibility with existing sequential mass storage files is achieved.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

8.2.1.5. ISAM Organization

The ISAM organization is a Sperry Univac extension to support the user's existing indexed sequential files created by the OS/3 indexed sequential access method (ISAM) data management.

An ISAM file is a mass storage file in which data records are accessed based on a key field contained in each record. The position of each logical record in the file is determined by indexes created and maintained by the operating system. The indexes are based on keys provided by the data item named in the RECORD KEY clause of the file control entry for that file.

For inserting and updating records in a file, each record is identified solely by the value of its record key. This value must, therefore, be unique and must not be changed when updating the record.

In the sequential access mode, the sequence in which records are accessed is the ascending order of the record key values.

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.

8.2.2. Current Record Pointer

The current record pointer is a conceptual entity used to indicate the next record to be accessed within a file that is opened in the INPUT or I/O mode. 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.

8.2.3. I/O Status

If the FILE STATUS clause (4.4.1) is specified, a value is placed by the operating system into the specified 2-character data item to indicate to the COBOL program the status of that input/output operation. The value is placed in the FILE STATUS data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement for that file and before the execution of any associated AT END/INVALID KEY imperative statement or any applicable format 1 USE procedure.

The leftmost character position of the FILE STATUS data item is known as status key 1; the rightmost character position is known as status key 2. Status key 1 is set to indicate a specific condition; status key 2 provides further information, if any, about the input/output operation. Table 8–1 lists the status key values and their meanings for each type of file organization. The meanings for the key values are further described in the notes following the table.

Table 8—1. Status Key Values and Meanings

File Organization	Status Key 1	Status Key 2
Sequential	0 – Successful completion	0 – No further information
and	1 – At end	0 – No further information
ISAM	3 – Permanent error	0 – No further information 4 – Boundary violation
Relative	0 – Successful completion	0 – No further information
	1 – At end	0 – No further information
	2 – Invalid key	2 – Duplicate key 3 – No record found 4 – Boundary violation
	3 – Permanent error	0 – No further information
Indexed	0 – Successful completion	0 – No further information 2 – Duplicate key (indexed files only)
and	1 – At end	0 – No further information
ISAM	2 – Invalid key	1 – Sequence error 2 – Duplicate key 3 – No record found 4 – Boundary violation
	3 – Permanent error	0 – No further information

NOTES:

1. At end – A format 1 or format 2 READ statement is unsuccessful because no next logical record exists, or an OPTIONAL file is not available at OPEN time.

2. Boundary violation – An attempt is made to write beyond the externally defined boundaries of a file.

3. Duplicate key – An attempt is made to write a record to a relative file, or to write or rewrite a record to an indexed or ISAM file, which will create a duplicate key in the file.

4. No record found – An attempt is made to access a record, identified by a key, and that record does not exist in the file.

5. Permanent error – The I/O statement is unsuccessful because of an unrecoverable I/O error, or a boundary violation for a sequential file.

6. Sequence error – For a sequentially accessed indexed or ISAM file, the ascending sequence requirements for successive RECORD KEY values are violated, or the prime record key value of an indexed file, or the RECORD KEY value of an ISAM file is changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.

8.2.4. The AT END Condition

The AT END condition can occur as a result of the execution of a format 1 or format 2 READ statement. For details of the causes of the condition, see 6.6.25, the READ statement.

8.2.5. 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 a relative, indexed, or ISAM file. For details of the causes of the condition, see 6.6.34, the START statement; 6.6.25, the READ statement; 6.6.42, the WRITE statement; 6.6.29, the REWRITE statement; and 6.6.9, the DELETE statement.

When the INVALID KEY condition is recognized, the operating system takes these actions in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition.

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 for this file, either explicitly or implicitly, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input/output statement that recognized the condition is unsuccessful and the file is not affected.

8.2.6. 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 for a sequential file. The maximum size of a logical page or the maximum value of LINAGE-COUNTER is 999. See 5.3.1.6, the LINAGE clause, for the rules governing the LINAGE-COUNTER.

8.3. SEQUENTIAL FILE PROCESSING

8.3.1. 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.

8.3.2. Clauses and Statements for Sequential File Processing

The basic clauses and statements for sequential file processing are summarized in the following paragraphs.

8.3.2.1. Environment Division

- ASSIGN TO implementor-name

The ASSIGN clause specifies implementor-name in the form of device type-lfd name mode (record format). For files assigned to CARDREADER, CARDPUNCH, PRINTER, TAPE, DISC, or DISK, the record format may be F, V, U, FC, VC, or UC.

- RESERVE $\begin{Bmatrix}1\\2\end{Bmatrix}$ AREAS

The integer in the RESERVE clause can be only 1 or 2. If the clause is not specified, two areas are reserved.

- ORGANIZATION IS SEQUENTIAL

The ORGANIZATION IS SEQUENTIAL clause states that the file is organized in a sequential manner. Records are accessed one after another. If this clause is omitted, ORGANIZATION SEQUENTIAL is implied. Keys are not allowed with sequential files.

Sequential files assigned to CARDREADER, CARDPUNCH, PRINTER, or TAPE are processed by the SAM data management.

Files specified with the ORGANIZATION IS SEQUENTIAL clause and assigned to DISC or DISK are processed by MIRAM (unkeyed, no RCB) data management.

- ACCESS MODE IS SEQUENTIAL

This clause specifies the manner in which records are to be accessed. For sequential files, the access mode is always sequential. The clause, therefore, is optional.

- FILE STATUS

A value is placed by the operating system in the FILE STATUS data item to indicate the status of an input/output operation.

8.3.2.2. Data Division

- LABEL RECORDS

For mass storage files, LABEL RECORDS STANDARD is required. For magnetic tape files, LABEL RECORDS OMITTED, STANDARD, or data-name is permissible. For card-reader, card-punch, and printer files, LABEL RECORDS OMITTED is required.

- LINAGE

The LINAGE clause defines the size of a logical page of a printer-destined file. If the LINAGE clause is specified, a LINAGE-COUNTER is provided and maintained by the compiler-generated code to indicate the line number within the current page body at any given time during the execution of the object program.

- CODE-SET

The CODE-SET clause may only be specified for tape files. This clause specifies the character code set used to represent data on a sequential tape file. When STANDARD-1 or STANDARD-0 is specified in the SPECIAL NAMES paragraph (4.3.3), all data must be described as usage is DISPLAY, and any signed numeric data must be described with the SIGN IS SEPARATE clause. The compiler assumes that the file has a buffer offset of zero. For an explanation of buffer offset and ASCII tape file formats, see the Basic data management user guide, UP-8068 (current version). Refer also to the PROGRAM COLLATING SEQUENCE clause (4.3.2).

8.3.2.3. Procedure Division

- OPEN INPUT

The OPEN INPUT statement specifies that a file is accessed by READ statements only. The REVERSED and NO REWIND phrases apply only to single-reel files assigned to magnetic tape.

- OPEN OUTPUT

The OPEN OUTPUT statement specifies that the file is to be created by WRITE statements only. The NO REWIND phrase applies only to single-reel files assigned to magnetic tape.

- OPEN I–O

The OPEN I–O statement is for mass storage files only. It indicates that a file is to be updated by pairs of READ and REWRITE statements.

- OPEN EXTEND

The OPEN EXTEND statement specifies that a file is to be extended by adding new records (with the WRITE statement) to the end of the file. The EXTEND phrase may only be specified for tape or mass storage files and must not be specified for files stored on MULTIPLE FILE TAPE.

- CLOSE

The CLOSE statement terminates the processing of a file. The LOCK phrase prevents the file from being opened again during the current execution of this run unit. The CLOSE WITH NO REWIND applies only to single-reel files assigned to magnetic tape.

- READ

The READ statement makes available the next logical record from a file. If the AT END phrase is not specified, an applicable USE ERROR procedure is required.

- WRITE

The WRITE statement releases a logical record for an output file. The ADVANCING phrase is used for vertical positioning of lines within a logical page of a printer-destined file.

- REWRITE

The REWRITE statement replaces a record previously read in a mass storage file.

8.3.3. Printer-Destined Files

Printer-destined files are files that are defined with a mode of FC, VC, or UC in the implementor-name of the ASSIGN clause. Each logical record of a printer-destined file is preceded by a device-independent control character. This control character, however, is not accessible to the COBOL programs.

To accommodate vertical form positioning beyond the device-independent control character limit of 15 lines, records containing control character information are created for form positioning purposes only. These control records are not printed nor made available by a READ statement when the file is opened as INPUT.

The presence of a control character must be considered in computing the block size for the file. (See 5.3.1.1, the BLOCK CONTAINS clause.)

Files assigned to PRINTER are automatically printed. Printer-destined files assigned to devices other than PRINTER require a print routine for printing.

8.3.4. Multivolume Sequential Files

For multivolume mass storage sequential files, only one volume is mounted at a time. After one volume is processed, it must be dismounted, and then the next volume must be mounted before processing continues.

For multireel tape sequential files, one or two reels are mounted at a time. When two reels are mounted, reel swapping is automatic.

8.4. RELATIVE FILE PROCESSING

8.4.1. 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.

8.4.2. Clauses and Statements for Relative File Processing

The basic clauses and statements for relative file processing are summarized in the following paragraphs.

8.4.2.1. Environment Division

- ASSIGN TO implementor-name

  The ASSIGN clause specifies implementor-name in the form of device type-lfd-name mode (record format). The device type permitted is DISC or DISK, and the record format must be F or V.

- RESERVE $\left\{\begin{matrix}1\\2\end{matrix}\right\}$ AREAS

  The integer in the RESERVE clause can be only 1 or 2. For relative files, if the ACCESS MODE is sequential, either one or two areas may be specified; if the ACCESS MODE is random or dynamic, only one area is allocated regardless of the value specified in the RESERVE clause. If this clause is not specified, two areas are allocated when the ACCESS MODE is sequential and one area is reserved when the ACCESS MODE is random or dynamic.

- ORGANIZATION IS RELATIVE

  The ORGANIZATION IS RELATIVE clause designates the file as relatively organized. Each record in the file is identified by a relative record number. This clause is required. If this clause is not specified, ORGANIZATION IS SEQUENTIAL is assumed. Relative files are processed by MIRAM (unkeyed, RCB) data management.

- ACCESS MODE IS $\left\{\begin{matrix}\text{SEQUENTIAL}\\\text{RANDOM}\\\text{DYNAMIC}\end{matrix}\right\}$

  Sequential access processes the file in a sequential manner. Random access indicates that the sequence in which the records are accessed is based on the contents of the RELATIVE KEY data item provided by the COBOL program.

  Dynamic access indicates that the file may be processed sequentially, randomly, or both, depending on the appropriate input/output statements.

  If the ACCESS MODE clause is not specified, ACCESS IS SEQUENTIAL is implied.

- RELATIVE KEY

  The value placed in the RELATIVE KEY data item by the COBOL program represents the logical ordinal position of the intended record. The RELATIVE KEY data item, therefore, must not be defined as a part of the data record of the file. For files accessed randomly, the RELATIVE KEY data item specifies the record to be processed. For files accessed sequentially, the record number processed is returned in the RELATIVE KEY data item.

  In the sequential access mode, the RELATIVE KEY phrase is optional; in the random or dynamic access mode, the phrase is required.

- FILE STATUS

  A value is placed by the operating system in the FILE STATUS data item to indicate the status of an input/output operation.

8.4.2.2. Data Division

- LABEL RECORDS ARE STANDARD

  Standard labels are required for relative files.

8.4.2.3. Procedure Division

- OPEN INPUT

  The OPEN INPUT statement indicates that a file is to be accessed by the READ or START statement, and standard labels are checked by the operating system.

- OPEN OUTPUT

  The OPEN OUTPUT statement indicates that a file is to be created by the WRITE statement either sequentially or randomly. Standard labels are written by the operating system.

- OPEN I-O

  The OPEN I-O statement indicates that a file to be processed for both input and output operations. Standard labels are checked by the operating system.

- START

  The START statement positions a file to the desired record for subsequent sequential retrieval. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- READ

  The READ statement makes available either the next logical record for sequential access or the specified record for random access. The NEXT phrase must be specified for sequential retrieval in the dynamic access mode. The AT END or INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- WRITE

  The WRITE statement releases a logical record for an output or input/output file. In the sequential access mode, the sequence in which records are released constitutes the logical ordinal positions of the records in the file. In the random or dynamic access mode, each record is placed in file according to the relative record number provided in the RELATIVE KEY data item by the COBOL program. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- REWRITE

  In the sequential access mode, a REWRITE statement replaces the last logical record read by a READ statement. In the random or dynamic access mode, a logical record is stored in the file based on the relative record number supplied by the COBOL in the RELATIVE KEY data item. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- DELETE

  In the sequential access mode, the last logical record read by a READ statement is deleted. In the random or dynamic mode, the logical record identified by the relative record number supplied by the COBOL program is logically removed. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- CLOSE

  The CLOSE statement terminates the processing of a file. The LOCK phrase, if specified prevents the file from being opened again during the current execution of this run unit. Standard labels are processed by the operating system.

8.5. INDEXED FILE PROCESSING

8.5.1. 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.

8.5.2. Clauses and Statements for Indexed File Processing

The basic clauses and statements for indexed file processing are summarized in the following paragraphs.

8.5.2.1. Environment Division

- ASSIGN TO implementor-name

  The ASSIGN clause specifies implementor-name in the form of device type-lfdname-mode (record format). The device type permitted is DISC or DISK. The record format must be F or V.

- RESERVE $\left\{\begin{matrix}1\\2\end{matrix}\right\}$ AREAS

  Only one area is allocated regardless of the value specified in the RESERVE clause. If the clause is omitted, one area is reserved by the compiler.

- ORGANIZATION IS INDEXED

  The ORGANIZATION IS INDEXED clause is required to indicate that the file organization is indexed. If this clause is not specified, ORGANIZATION IS SEQUENTIAL is assumed. Indexed files are supported by MIRAM (keyed, RCB) data management.

- ACCESS MODE IS $\left\{\begin{matrix}\text{SEQUENTIAL}\\\text{RANDOM}\\\text{DYNAMIC}\end{matrix}\right\}$

  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.

- RECORD KEY

  The RECORD KEY clause is required. It specifies the prime record key for the file. The values of the prime record key must be unique among records of the file. The data item named in the RECORD KEY clause must be described in the record description of the file.

- ALTERNATE RECORD KEY

  The ALTERNATE RECORD KEY clause is optional. It specifies a record key that is an alternate record key for the file. Up to four alternate record keys may be specified for the file. The values of alternate record keys need not be unique if the DUPLICATE phrase is specified. The data item named in an ALTERNATE RECORD KEY clause must be described in the record description of the file.

- FILE STATUS

  A value is placed by the operating system in the FILE STATUS data item to indicate the status of an input/output operation.

8.5.2.2. Data Division

- LABEL RECORDS ARE STANDARD

  Standard system labels are required.

8.5.2.3. Procedure Division

- OPEN INPUT

  The OPEN INPUT statement indicates that a file is to be accessed by the READ or START statements. The standard labels are checked by the operating system.

- OPEN OUTPUT

  The OPEN OUTPUT statement indicates that a new file is to be created by the WRITE statement either sequentially or randomly. Standard labels are written by the operating system.

- OPEN I/O

  The OPEN I/O statement indicates that a file is to be processed for both input and output operations. Standard labels are checked by the operating system.

- START

  The START statement positions a file to the desired area for subsequent sequential retrieval. Any relational operator may be specified in the KEY phrase of this statement.

- READ

The READ statement makes available either the next logical record for sequential access or the next specified record for random access. The NEXT phrase must be specified for sequential retrieval in the dynamic access mode. The AT END or INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- WRITE

The WRITE statement releases a logical record for an output or input/output file. In the sequential access mode, records are released in the ascending order of the record key values. In the random or dynamic access mode, records may be released in any program-specified order. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- REWRITE

In the sequential access mode, a REWRITE statement replaces the logical record last read by a READ statement. In the random or dynamic access mode, the record to be released is specified by the key value in the RECORD KEY data item. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- DELETE

In the sequential access mode, the logical record last read by a READ statement is deleted. In the random or dynamic mode, the record identified by the contents of the RECORD KEY data item is deleted. The INVAlID KEY phrase is required if no applicable USE ERROR procedure is specified.

- CLOSE

The CLOSE statement terminates the processing of a file. The LOCK phrase, if specified, prevents the file from being opened again during the current execution of this run unit. Standard labels are processed by the operating system.

8.6. SAM* FILE PROCESSING

The SAM file processing facility is provided for compatibility with files created by the disk sequential access method (disk SAM) of the OS/3 data management.

The basic clauses and statements for SAM file processing are summarized in the following paragraphs.

8.6.1. Environment Division

- ASSIGN TO implementor-name

The ASSIGN clause specifies implementor-name in the form of device type-lfdname-mode (record format). SAM files must be assigned to DISC or DISK. The record format may be F, V, FC, or VC.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

- RESERVE $\left\{ \begin{matrix} 1 \\ 2 \end{matrix} \right\}$ AREAS

  The integer in the RESERVE clause can be only 1 or 2. If the clause is omitted, two areas are reserved.

- ORGANIZATION IS SAM

  The ORGANIZATION IS SAM clause specifies that the file will be supported by the disk sequential access method (disk SAM). If this clause is omitted, ORGANIZATION IS SEQUENTIAL (MIRAM unkeyed) is implied. Keys are not allowed with SAM files.

- ACCESS MODE IS SEQUENTIAL

  This clause specifies the method used to access records. For SAM files, the access mode is always sequential. The clause, therefore, is optional.

- FILE STATUS

  The operating system places a value in the FILE STATUS data item to indicate the status of an input/output operation.

8.6.2. Data Division

- LABEL RECORDS

  For SAM files, the LABEL RECORDS STANDARD clause is required.

- LINAGE

  The LINAGE clause defines the size of a logical page of a printer-destined file. If the LINAGE clause is specified, a LINAGE-COUNTER is provided and maintained by the compiler-generated code to indicate the line number within the current page body at any given time during the execution of the object program.

8.6.3. Procedure Division

- OPEN INPUT

  The OPEN INPUT statement specifies that a file is accessed by READ statements only. Specification of this statement causes the operating system to check standard system labels.

- OPEN OUTPUT

  The OPEN OUTPUT statement specifies that the file will be created by WRITE statements only. Specification of this statement causes the operating system to create system labels.

- OPEN I/O

  The OPEN I/O statement indicates that an existing file will be updated by pairs of READ and REWRITE statements.

- OPEN EXTEND

  The OPEN EXTEND statement specifies that a file will be extended by adding new records (with the WRITE statement) to the end of the file.

- READ

  The READ statement makes available the next logical record from a file. If the AT END phrase is not specified, an applicable USE ERROR procedure is required.

- WRITE

  The WRITE statement releases a logical record for an output file. The ADVANCING phrase is used for vertical positioning of lines within a logical page of a printer-destined file.

- REWRITE

  The REWRITE statement replaces a record previously read.

- CLOSE

  The CLOSE statement terminates the processing of a file. The LOCK phrase prevents the file from being opened again during the current execution of this run unit.

8.6.4. Multivolume SAM Files

For multivolume mass storage SAM files, only one volume is mounted at a time. After one volume is processed, it must be dismounted, and then the next volume must be mounted before processing continues.

8.7. ISAM* FILE PROCESSING

The ISAM file processing facility is provided for compatibility with files created by the indexed sequential access method (ISAM) of the OS/3 operating system.

The basic clauses and statements for ISAM file processing are summarized in the following paragraphs.

8.7.1. Environment Division

- ASSIGN TO implementor-name

  The ASSIGN clause specifies implementor-name in the form of device type-lfdname-mode (record format). The device types permitted are DISC and DISK. The record format must be F or V.

- RESERVE $\left\{\begin{matrix}1\\2\end{matrix}\right\}$ AREAS

  The integer in the RESERVE clause can be only 1 or 2. If the clause is not specified, two areas are reserved.

- ORGANIZATION IS ISAM

  The ORGANIZATION IS ISAM clause is required to indicate that the file organization is ISAM. ISAM files are processed by ISAM data management.

Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems

- ACCESS MODE IS $\left\{\begin{array}{l}\text{SEQUENTIAL}\\ \text{RANDOM}\\ \text{DYNAMIC}\end{array}\right\}$

  In the sequential access mode, records are accessed in the ascending order of the record key values. For random access, the desired record is accessed based on the value in the record key data item provided by the COBOL program. In the dynamic access mode, a file may be processed sequentially, randomly, or both, depending on the appropriate input/output statements.

- RECORD KEY

  The RECORD KEY data item is the key field contained in each record. This is a required clause. The data item named in the RECORD KEY clauses must be described in the record description of the file.

- FILE STATUS

  A value is placed by the operating system in the FILE STATUS data item to indicate the status of an input/output operation.

8.7.2. Data Division

- LABEL RECORDS ARE STANDARD

  Standard labels are required for ISAM files.

8.7.3. Procedure Division

- OPEN INPUT

  The OPEN INPUT statement indicates that a file is to be accessed by the READ or START statements. The standard labels are checked by the operating system.

- OPEN OUTPUT

  The OPEN OUTPUT statement indicates that a new file is to be sequentially created by the WRITE statement. Standard labels are written by the operating system.

- OPEN I-O

  The OPEN I-O statement indicates that a file is to be processed for both input and output operations. Standard labels are checked by the operating system.

- START

  The START statement positions a file to the desired area for subsequent sequential retrieval. For an ISAM file, the relational operators permitted in the KEY phrase are NOT LESS THAN or NOT <, and EQUAL TO or =. The INVALID KEY clause is required if no applicable USE ERROR procedure is specified.

- READ

  The READ statement makes available either the next logical record for sequential access or the next specified record for random access. The NEXT phrase must be specified for sequential retrieval in the dynamic access mode. The AT END or INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- WRITE

  The WRITE statement releases a logical record for an output or input/output file. In the sequential access mode, records are released in the ascending order of the record key values. In the random access mode, records may be released in any program-specified order. In the dynamic access mode, if the file is opened as OUTPUT, records must be released in ascending order of the record key values. If the file is opened as I-O, records may be released in any program-specified order. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- REWRITE

  In the sequential access mode, a REWRITE statement replaces the logical record last read by a READ statement. In the random or dynamic access mode, the record to be released is specified by the key value in the RECORD KEY data item. The INVALID KEY phrase is required if no applicable USE ERROR procedure is specified.

- CLOSE

  The CLOSE statement terminates the processing of a file. The LOCK phrase, if specified, prevents the file from being opened again during the current execution of this run unit. Standard labels are processed by the operating system.

9. Sort/Merge Summary

9.1. GENERAL

The COBOL sort/merge facility 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 or after the records are ordered by the SORT statement or after the records have been combined by the MERGE statement. The standard system utility sort/merge subroutine is used to perform the sort and merge operations.

In general, a sort operation proceeds as follows:

1. Control passes to a SORT statement. The SORT statement specifies the sort file to be created and the data keys that guide the sort operation. It either identifies the input procedure and output procedure or names the source of the unsorted input records and that file which is to receive the sorted output records.

2. The input procedure, if named in the SORT statement, is executed. This input procedure must contain at least one RELEASE statement. If no input procedure is specified, the input file is named in the USING phrase of the SORT statement. The effect of either option is to make input records available to the sort operation.

3. The records made available to the sort operation are sorted on a set of specified keys as shown in the KEY phrase according to the EBCDIC collating sequence, or a user-specified collating sequence.

4. The SORT statement passes control to the output procedure, if one is named. The output procedure must contain at least one RETURN statement to return the sorted record from the sort file to the COBOL program. If no output procedure is used, the GIVING phrase must specify the output file.

5. The execution of the SORT statement is terminated and control passes to the next statement in sequence.

When the input or output procedure is in control, all transfers of control must refer to procedures contained within that input or output procedure. Conversely, control cannot be transferred into an input or output procedure from points in the procedure division outside the limits of the input or output procedure. Neither an input nor an output procedure may contain a SORT or MERGE statement.

The process of a merge operation is similar to a sort operation except:

- two or more files in identical sequence are merged into one output file; and

- the input procedure is not permitted in a MERGE statement. Files to be merged must be specified in the USING phrase.

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 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.

9.2. LANGUAGE CONCEPTS

9.2.1. Relationship with File Processing Facility

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 or SAM organization. No input/output statement may be executed for the file named in the sort/merge file description.

9.2.2. Sort Special Registers

Two sort special registers, SORT-FILE-SIZE and SORT-MODE-SIZE, provide a means of object time communication between the COBOL program and the sort/merge routine, and they aid in the efficiency of the sort operation.

The registers may be referenced as operands of any statements where signed binary data items are permitted. The information to be contained in the registers must be passed before the SORT statement is executed. The registers are data items generated by the compiler and initialized by the compiler to zeros, but are not reset after a sort operation is completed.

- SORT-FILE-SIZE

  The reserved word SORT-FILE-SIZE is the name of a binary data item whose PICTURE is S9(8). It is used for the estimated number of records in the file to be sorted.

- SORT-MODE-SIZE

  The reserved word SORT-MODE-SIZE is the name of a binary data item whose PICTURE is S9(4). It is used for variable-length records. If the length of most records in the file is significantly different from the average record length, performance is improved by specifying the most frequently occurring record length.

9.3. ENVIRONMENT DIVISION CONSIDERATIONS

SELECT clauses must be included for the following types of files:

- Files used within input and output procedures
- The sort or merge file
- Files named in the USING and GIVING options of the SORT or MERGE statement

9.3.1. File Control Entry

The file control entry names a sort or merge file and specifies the association of the file to a storage medium.

Format:

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

Work files required by sort or merge operations may be assigned to WORK1, WORK2,... WORKn; where n ranges from 1 through 8 for disks, or from 1 through 6 for tapes.

Refer to the SELECT entry in 4.4.1.

9.3.2. I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies the main storage area to be shared by different files.

Format:

```
I-O-CONTROL.
    ⎡;SAME ⎡RECORD     ⎤ AREA FOR file-name-1,file-name-2[,file-name-3] ...⎤ ...
    ⎢      ⎢SORT       ⎥                                                   ⎥
    ⎣      ⎣SORT-MERGE ⎦                                                   ⎦
```

Refer to 4.4.2, the I-O-CONTROL paragraph.

9.4. DATA DIVISION CONSIDERATIONS

The file section of the data division must include the following entries:

- File description entries for all files used within input and output procedures
- File description entries for files named in the USING and GIVING phrases of the SORT or MERGE statement
- Sort/merge file description entries for the sort or merge files
- Record description entries for all files

A sort/merge 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 that the user can control, and the rules for blocking and internal storage are peculiar to the SORT and MERGE statements.

Format:

```
SD file-name
    [;RECORD CONTAINS clause]
    ⎡;DATA ⎧RECORD IS  ⎫ clause⎤
    ⎣      ⎩RECORDS ARE⎭       ⎦
```

Refer to 5.3.2 for details concerning the sort/merge file description entry.

9.5. PROCEDURE DIVISION CONSIDERATIONS

The procedure division must contain a SORT or MERGE statement to control the sorting or merging operation, and, optionally, may contain input and output procedures to process records before and after the sort, or output procedure to make merged records available for processing.

An input procedure must include a RELEASE statement to release records to the sort file, and an output procedure must include a RETURN statement to obtain records from the sort or merge file. In addition, the EXIT statement may be used as a common end point for the input or output procedure used with the sort/merge feature.

9.5.1. RELEASE Statement

The RELEASE statement transfers records to the initial phase of a SORT operation.

Format:

```
RELEASE record-name [FROM identifier]
```

Details for using this statement are given in 6.6.27.

9.5.2. RETURN Statement

The RETURN statement obtains sorted records from the final phase of a sort operation, or merged records during a merge operation.

Format:

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

Details for using this statement are given in 6.6.28.

9.5.3. SORT Statement

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.

Format:

```
SORT file-name-1 ON {ASCENDING } KEY phrase
                    {DESCENDING}
     [COLLATING SEQUENCE] phrase
     {INPUT PROCEDURE} phrase
     {USING          }
     {OUTPUT PROCEDURE} phrase
     {GIVING          }
```

Details for use of this statement are given in 6.6.33.

9.5.4. MERGE Statement

The MERGE statement combines two or more identically sequenced files on a set of specified keys and makes merged records available to an output procedure or to an output file.

Format:

```
MERGE file-name-1 ON {ASCENDING } KEY  phrase
                     {DESCENDING}

      [COLLATING SEQUENCE] phrase

      USING phrase

      {OUTPUT PROCEDURE} phrase
      {GIVING          }
```

Details for use of this statement are given in 6.6.19.

9.6. OBJECT TIME SUBROUTINE SORT/MERGE MAIN STORAGE REQUIREMENTS

When a SORT or MERGE statement is included in a COBOL program, the object code generated by the compiler dynamically invokes the OS/3 subroutine sort/merge to perform a sorting or merging operation.

The minimum main storage required by the subroutine sort/merge is 13,000 ($32C8_{16}$) bytes. This requirement is dynamically requested within the job region by the COBOL run time interface subroutine via a DMEM macro. Therefore, the user must include this main storage requirement in the size of the job region in the // JOB job control statement.

For sort/merge programs containing statically bound subprograms (compiler option CALLST=YES), the minimum job region size is 13,000 bytes plus the size of the load module.

For sort/merge programs involving dynamic subprograms (compiler option CALLST=NO), the minimum job region size is 13,000 bytes plus all active load modules at the time a sort or merge is in operation.

If main storage space more than the sort/merge minimum requirement is available in the job region, the additional space is utilized in the sorting operation to improve efficiency.

Sort/merge is more efficient when 50,000 to 150,000 bytes of main storage are allocated.

10. Segmentation Summary

10.1. GENERAL

COBOL segmentation is a facility that enables the user to communicate with the compiler to specify object program overlay requirements. COBOL segmentation deals only with segmentation of procedures. Only the procedure division and the environment division are considered in determining segmentation requirements for an object program.

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.

Under segmentation Level 1, users can specify permanent and independent segments. 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.

Under segmentation Level 2, users can intermix sections with different segment-numbers and the fixed portion of the source program may contain segments that may be overlaid.

10.2. ORGANIZATION

10.2.1. Fixed Portion

The fixed portion is defined as part of the object program that is logically treated as if it were always in main storage. 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 that cannot be overlaid by any other part of the program. A fixed overlayable segment is a segment in the fixed portion that, although logically treated as if it were always in main storage, can be overlaid by another segment to optimize main storage 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 4.3.2). Such a segment, if called for by the program, is always made available in its last used state.

10.2.2. Independent Segments

An independent segment is defined as part of the object program that 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 statement, in a segment with a different 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 item 2 in the following paragraph).

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 1 and 2 in the preceding paragraph).

2. Control is transferred explicitly to that segment as the result of the execution of an EXIT PROGRAM statement.

10.3. SEGMENTATION CLASSIFICATION

Sections to be segmented are classified by means of a system of segment numbers using the following criteria:

- Logic Requirements

  Sections that must be available for reference at all times or are referred to frequently are normally classifed as belonging to one of the permanent segments; sections that 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.

- 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.

- Relationship to Other Sections

  Sections that frequently communicate with one another should be given the same segment numbers.

10.4. SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. The compiler automatically provides transfers of control from one segment to another.

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.

10.5. STRUCTURE OF PROGRAM SEGMENTS

10.5.1. Segment Numbers

Section classification is accomplished via a system of segment numbers. The segment number is included in the section header.

Format:

```
section-name SECTION [segment-number] .
```

Refer to 6.1.3.2 for details on using segment numbers.

10.5.2. SEGMENT-LIMIT Clause

Ideally, all program segments having segment numbers ranging from 0 through 49 should be specified as permanent segments. However, when insufficient storage 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).

Format:

```
[,SEGMENT-LIMIT] clause
```

Refer to 4.3.2 for details on using this clause.

10.5.3. Object Module Naming Conventions

For a segmented object program, two or more object modules are produced. The naming conventions for these modules are explained here.

One module (the root phase) consists of the initialization code, data areas, and fixed permanent segments of the procedure division code. The name of this module is taken from the first six characters of the program-name in the PROGRAM-ID paragraph. If the program-name is less than six characters, the program-name is used.

One module (overlay) is produced for each fixed overlayable segment and for each independent segment. The module name for each overlay is composed of eight characters: the first six characters of the root module (if the name is less than six characters, zeros are added to the right to form six characters) and a 2-character numeric suffix assigned consecutively from 01 to 99. For example, a segmented program named PAY4A in the PROGRAM-ID paragraph would have object modules named PAY4A, PAY4A001, PAY4A002, etc.

10.5.4. Linkage Editor Control Statement Considerations

Each object module (except the last) produced for a segmented program contains embedded linkage editor control statements that define an overlay point and include the next object module in the series of object modules. For example, a segmented program object module named PAYA4002 would contain the embedded control statements:

```
OVERLAY  PAYA4###
INCLUDE  PAY4AØØ3.
```

An example of the linkage editor commands used to link a segmented program is:

```
LOADM  PAY4A
INCLUDE  PAY4A.
```

The proper overlay structure is automatically generated by the embedded linkage editor control statements.

When linking subprograms with a segmented program, the programmer must ensure that the linkage editor control statements do not conflict with the compiler-generated linkage editor control statements embedded in the object modules of the segmented program. In particular, the subprogram must not be included in an overlay unless it is called only from the same overlay or the root. It is recommended that subprograms be included in the root or in a linkage editor region so that they can be called from any overlay in the segmented COBOL program.

The compiler-generated linkage editor control statements can be suppressed by specifying the compiler option parameter LNKCON=NO. See Appendix A.

10.6. COBOL VERBS AFFECTED BY SEGMENTATION

When segmentation is used, the following restrictions are placed on the ALTER, PERFORM, MERGE, and SORT statements.

10.6.1. 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.

10.6.2. PERFORM Statement

A PERFORM statement 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:

- Sections or paragraphs wholly contained in one or more nonindependent segments

- Sections or paragraphs wholly contained in a single independent segment

A PERFORM statement 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:

- Sections or paragraphs wholly contained in one or more nonindependent segments
- Sections or paragraphs wholly contained in the same independent segment as that PERFORM statement

10.6.3. SORT Statement

If a SORT statement appears in a section that is not an independent segment, any input procedures or output procedures referenced by that SORT statement must appear:

- totally within nonindependent segments; or
- wholly contained in a single independent segment.

If a SORT statement appears in an independent segment, any input procedures or output procedures referenced by that SORT statement must be contained:

- totally within nonindependent segments; or
- wholly within the same independent segment as that SORT statement.

10.6.4. 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:

- totally within nonindependent segments; or
- 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:

- totally within nonindependent segments; or
- wholly within the same independent segment as that MERGE statement.

11. Library Summary

11.1. GENERAL

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.

The compilation summary listing shows the volume (VOL), label (LBL), and LFD information for all library files accessed by the compiler.

COBOL library text is placed on the OS/3 copy library by the system librarian routine (See OS/3 system service programs user guide.)

Library Level 1 facilitates 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 makes more than one COBOL library available at compile time.

11.2. COPY STATEMENT

The COPY statement incorporates text into a COBOL source program.

Format:

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

Details for using this statement are given in 6.6.8.

11.3. SOURCE PROGRAM CORRECTIONS DURING COMPILATION

A complete source program to be compiled from a library via the IN parameter (Appendix A) may be corrected temporarily during the compilation process using the SEQ, REC, and SKI system librarian control statements. (See OS/3 system service program user guide.)

The correction deck is interchangeable between the compiler and the librarian except that the librarian uses the added COR control statement, whereas the correction deck for the compilation always starts with the SEQ control statement as the first card.

The correction deck must be included within the data delimiters, /$ and /*. If there are no data cards between the data delimiters, no source correction is performed.

12. Debugging Language Summary

12.1. GENERAL

The debug module provides a means by which the user can describe a debugging algorithm and 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.

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.

Debug Level 1 provides a basic debugging capability to specify selective or full procedure monitoring, and optionally compiled debugging statements.

Debug Level 2 provides the additional COBOL debugging capability of specifying identifiers and file names for monitoring by the USE FOR DEBUGGING statement.

The *DEBUG statement provides debugging packets.

12.2. LANGUAGE CONCEPTS

The features of the COBOL language that support the debug module are:

- A compile-time switch - WITH DEBUGGING MODE
- An object-time switch - bit 0 of the UPSI byte
- A USE FOR DEBUGGING statement
- A special register - DEBUG-ITEM
- Debugging lines
- *DEBUG - debugging packet

12.2.1. DEBUG-ITEM Register

The reserved word DEBUG-ITEM is the name for a special register generated automatically 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. This register provides information about the conditions that cause the execution of a debugging section. See Table 12-1.

12.2.2. 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. Debugging packets are unaffected by the switch.

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.

12.2.3. Object-Time Switch

An object-time switch dynamically activates the debugging code inserted by the compiler. This switch setting should not be changed by the program; it is controlled outside the COBOL environment. If the switch is on, all the effects of the standard debugging language written in the source program are permitted. If the switch is off, all the effects described in 6.6.41, 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 is bit 0 of the UPSI byte and may be set with the job control statement:

```
// SET UPSI,{0}
           {1}
```

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.

The object time switch has no effect on the execution of the extended debugging facilities.

12.3. ENVIRONMENT DIVISION CONSIDERATIONS

12.3.1. WITH DEBUGGING MODE Clause

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.

Format:

```
SOURCE-COMPUTER: computer-name [WITH DEBUGGING MODE]:
```

Details for using this clause are given in 4.3.1.

12.4. PROCEDURE DIVISION CONSIDERATIONS

12.4.1. USE FOR DEBUGGING Statement

The USE FOR DEBUGGING statement identifies the user items that are to be monitored by the associated debugging section.

Format:

```
                      ┌──────────┐
USE FOR DEBUGGING    {│identifier│     }  phrase
                     {│file-name │     }
                     {└──────────┘     }
                     {procedure-name   }
                     {ALL PROCEDURES   }
```

Details for using this statement are given in 6.6.41.

12.4.2. 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 is 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.

12.4.3. The Extended Debugging Facility

The extended debugging facility consists of the functions provided by ON, EXHIBIT, and TRACE statements and the debugging packets.

12.4.3.1. ON Statement

The ON conditional statement specifies when the statements it contains are to be executed.

Format:

```
ON integer-1 [AND EVERY integer-2]
    [UNTIL integer-3] {statement-1  }
                      {NEXT SENTENCE}
    ELSE {statement-2  }
         {NEXT SENTENCE}
```

Details for using this statement are given in 6.6.22.

12.4.3.2. EXHIBIT Statement

The EXHIBIT statement displays on SYSOUT the current values of data items at selected points in the program.

Format:

```
EXHIBIT {NAMED         } {identifier         } ...
        {CHANGED NAMED } {nonnumeric-literal }
        {CHANGED       }
```

Details for using this statement are given in 6.6.14.

12.4.3.3. TRACE Statement

The TRACE statement initiates or terminates the display, on SYSLST (4.3.3), of the name and line number of a section or paragraph at the start of its execution.

Format:

```
{READY} TRACE
{RESET}
```

Details for using this statement are given in 6.6.38.

12.4.3.4. The Debugging Packet (*DEBUG)

The debugging packet consists of a set of testing statements for a specific procedure in a source program. It is compiled at the end of the source program but is executed as though it were placed immediately following the procedure-name but before the first statement of the source program procedure to be tested. The debugging packet is headed by the *DEBUG statement that names the source procedure it is intended to debug.

Any procedure in the source program may include a debugging packet. All packets are grouped together and placed immediately following the last statement of the source program.

Format:

```
*DEBUG procedure-name
```

Details for using this statement are given in 6.6.43.

Table 12—1. Debug Conditions and Contents of DEBUG-item

Condition	DEBUG-LINE Identifies	DEBUG-NAME Contains	DEBUG-CONTENTS Contains
If debugging section is executed because of:			
1. First execution of the first non-declarative procedure	First statement of that procedure	Name of that procedure	'START PROGRAM'
2. Reference to procedure-name-1 in an ALTER statement	ALTER statement that references procedure-name-1	Procedure-name-1	Applicable procedure-name associated with the TO phrase of the ALTER statement
3. Transfer of control associated with the execution of a GO TO statement	GO TO statement that transfers control to procedure-name-1	Procedure-name-1	Space
4. Refer to procedure-name-1 in INPUT or OUTPUT phrase of a SORT statement	SORT statement that references procedure-name-1	Procedure-name-1	'SORT INPUT' for INPUT phrase 'SORT OUTPUT' for OUTPUT phrase
5. Transfer of control from the control mechanism associated with a PERFORM statement	PERFORM statement that referenced procedure-name-1	Procedure-name-1	'PERFORM LOOP'
6. Implicit transfer of control from previous sequential paragraph to procedure-name-1	Previous statement	Procedure-name-1	'FALL THROUGH'
7. References to file-name-1	Source statement that references file-name-1	Name of file-name-1	1. Entire record read for READ 2. Spaces for all other references to file-name-1
8. Reference to identifier-1	Source statement that references identifier-1	Name of identifier-1	Contents of data item referenced by identifier-1 at time control passes to debugging section
If procedure-name-1 is a USE procedure to be executed and procedure-name-1 is referenced in a USE FOR DEBUGGING statement.	Statement that causes execution of USE procedure	Procedure-name-1	'USE PROCEDURE'

13. Interprogram Communication Summary

13.1. GENERAL

The interprogram communication module provides a facility that enables a program to communicate with other programs. This communication is provided by:

1. the ability to transfer control from one program to another within a run unit; and

2. the ability for both programs to have access to the same data items.

The interprogram communication facility is supported by the compiler in two ways. The first method makes use of the dynamic loading and unloading system facility during program execution and supports full COBOL language. The second method supports only Level 1 of the inter-program communication facility and makes use of static binding of programs by the linkage editor. The second method is invoked by the compiler option parameter CALLST=YES (Appendix A) in conjunction with the literal option of the CALL statement.

13.1.1. Transfer of Control

The CALL statement initiates the transfer of control 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, procedure statements are executed normally 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. 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 a COBOL program, 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.

13.1.2. Access to 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, and, at object time, main storage is allocated for the entire data division. In the called program, common data items are described in the linkage section, but, at object time, main storage 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 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 USING clause is treated as a reference to the corresponding operand in the USING clause of the active CALL statement.

After 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 main storage areas occupied by called programs may be released. In addition, the CANCEL guarantees that the canceled program will be in its initial state when called by a subsequent CALL statement.

13.1.3. Level Characteristics

Interprogram 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 interprogram 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 main storage for the program to which control is being passed.

13.2. DATA DIVISION CONSIDERATIONS

13.2.1. Noncontiguous Linkage Storage

Items in the linkage section that have 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 that 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.

13.2.2. Linkage Records

Data elements in the linkage section that have a hierarchic relationship must be grouped into records according to the rules for formation of record descriptions. Any clause in an input or output record description can be used in a linkage section.

13.2.3. Initial Values

The VALUE clause must not be specified in the linkage section except in condition-name entries (level 88).

13.3. PROCEDURE DIVISION CONSIDERATIONS

13.3.1. Procedure Division Header

Format:

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

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 identifier-2 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 identifier-3,... 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 identifier may appear more than once in the same USING phrase of a CALL statement.

Details for using the header are given in 6.1.3.

13.3.2. CALL Statement

The CALL statement transfers control from one object program to another within the run unit.

Format:

```
CALL {identifier-1} [USING phrase] [ON OVERFLOW phrase]
     {literal-1   }
```

Details for using the statement are given in 6.6.4.

13.3.3. CANCEL Statement

The CANCEL statement releases the main storage areas occupied by the referenced program.

Format:

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

Details for using this statement are given in 6.6.5.

13.3.4. EXIT PROGRAM Statement

The EXIT PROGRAM statement marks the logical end of a called program.

Format:

```
EXIT [PROGRAM].
```

Details for using this statement are given in 6.6.15.

13.4. OBJECT PROGRAM EXECUTION CONSIDERATIONS

A parameter in either SYSGEN or job control must be specified for the execution of a COBOL object program containing code produced for a CALL statement referencing a called program that is to be dynamically loaded.

In SYSGEN, the ROLLOUT=YES parameter plus the size of the buffer and buffer table must be specified. In job control, the DLOAD parameter of the SFT job control statement is included in the job stream. Refer to the current versions of the OS/3 system installation user guide/programmer reference and the job control user guide.

14. Communication Summary

14.1. GENERAL

The communication facility provides the ability to access, process, and create messages or portions of messages. It provides the ability to communicate through a message control system (MCS) with local and remote communication devices.

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.

14.2. MESSAGE CONTROL SYSTEM

The MCS consists of a COBOL message control system (CMCS) and the integrated communications access method (ICAM). The MCS is 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:

- 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.

- To perform line discipline, including such tasks as dial-up, polling, and synchronization.

- To perform device-dependent tasks, such as character translation and insertion of control characters, so that the COBOL user can create device-independent programs.

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 that 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.

14.3. 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 and the various queues 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.

14.4. RELATIONSHIP OF COBOL PROGRAM TO MCS AND COMMUNICATION DEVICES

The interfaces that 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:

- the interface between the COBOL object program and the MCS; and

- the interface between the MCS and the communication devices.

The COBOL source program uses three statements to control the interface with the MCS:

- the RECEIVE statement, which causes data in a queue to be passed to the COBOL object program;

- the SEND statement, which causes data associated with the COBOL object program to be passed to one or more queues; and

- 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:

- the ENABLE statement, which establishes logical connection between the MCS and one or more given communication devices; and

- the DISABLE statement, which breaks a logical connection between the MCS and one or more given communication devices.

These relationships are shown in Figure 14-1.

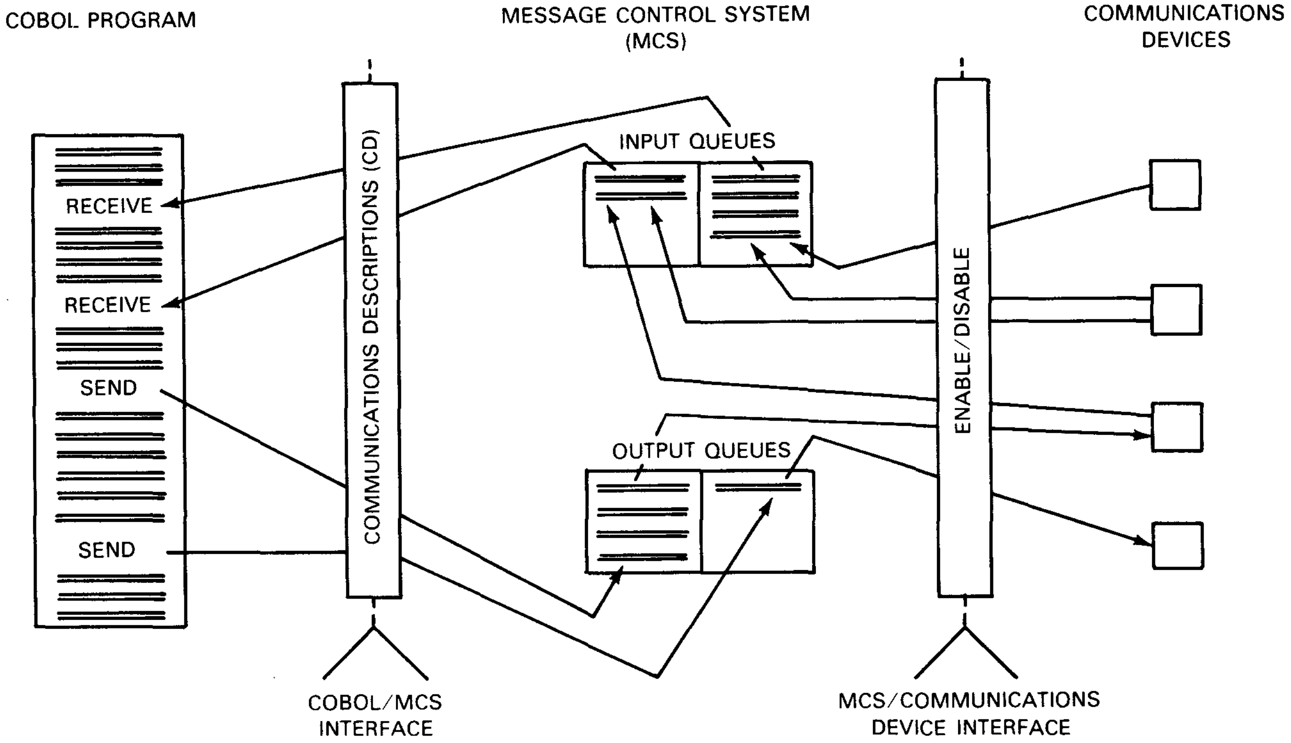


Figure 14—1. COBOL Communication Environment

14.4.1. Invoking the COBOL Object Program

There are two methods of invoking a COBOL communication object program: scheduled initiation and 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.

14.4.1.1. Scheduled Initiation

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:

- the ACCEPT MESSAGE COUNT statement;
- the RECEIVE statement with a NO DATA phrase; and
- the RECEIVE statement without a NO DATA phrase (in which case a program wait is implied if no data is available).

14.4.1.2. MCS Invocation

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 subqueue 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.

14.4.1.3. Determining the Method of Invocation

A COBOL source program can be written so that its object program can operate with either of the two methods of invocation. To determine which method was used to load the COBOL object program, the following is one technique that may be used:

- One CD must contain a FOR INITIAL INPUT clause.

- 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.

14.5. 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.

14.6. 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.

14.6.1. Enabling and Disabling Logical Connectives

Usually, the MCS logically connects and disconnects sources and destinations based on the parameters specified in the CMCS network definition. However, the COBOL program has the ability to perform these functions by using 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.

14.6.2. Enqueueing and Dequeueing Methods

It may be necessary that the user specify to the MCS, prior to execution of programs that 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 always done on a first-in, first-out basis. Thus, messages dequeued from either an input or output queue are those messages that have been in the queue for the longest period of time.

14.6.3. Queue Hierarchy

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 14-2, where queues and subqueues have been named with the letters A through O. Messages have been identified with a letter according to their source (X, Y, or Z) and with a sequential number.

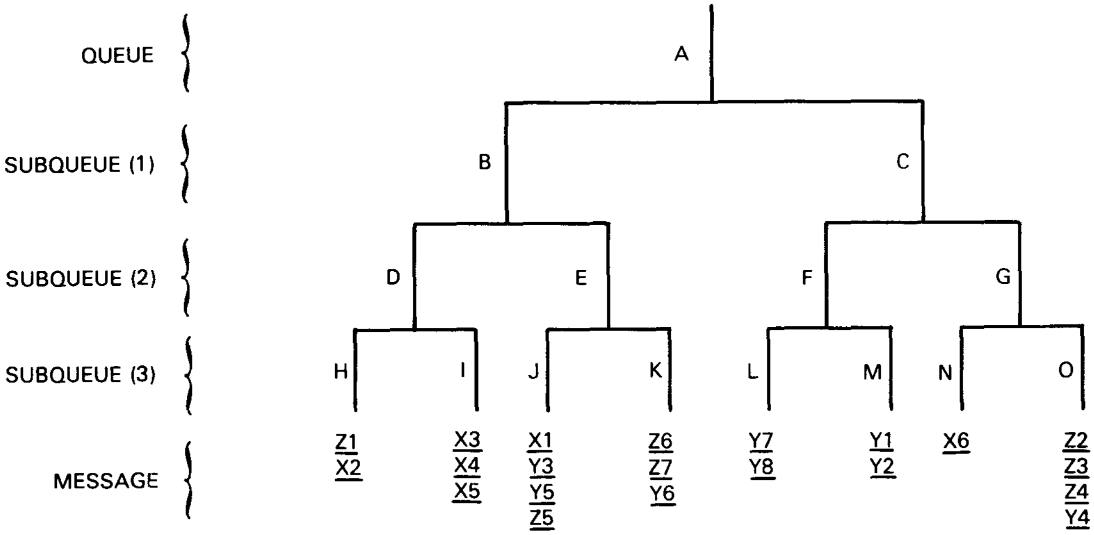


Figure 14—2. Hierarchy of Queues

Let us assume that the MCS is operating under the following queueing algorithm:

- Messages are placed in queues according to the contents of some specified data field in each message.
- With the RECEIVE statement, if the user does not specify a given subqueue level, the MCS will choose the subqueue from that level in alphabetical order; e.g., if subqueue-1 is not specified by the user, the MCS will dequeue from subqueue-1 B.

The following examples, using Figure 14-2, illustrate the effect of these algorithms:

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
   Subqueue-1 C

   MCS returns: Message Y7

3. The program executes a RECEIVE statement, specifying via the CD:

   Queue A
   Subqueue-1 B
   Subqueue-2 E

   MCS returns: Message X1

4. The program executes a RECEIVE statement, specifying via the CD:

   Queue A
   Subqueue-1 C
   Subqueue-2 G
   Subqueue-3 N

   MCS returns: Message X6

If the COBOL programmer wishes to access the next message in a queue, regardless of which subqueue 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 subqueue names via the data items in the associated CD. If, however, he desires the next message in a given subqueue, he must specify both the queue name and any applicable subqueue names.

For output, the COBOL user specifies only the destination of the message, and the MCS places the message in the proper output queue structure.

14.7. MESSAGE CONTROL SYSTEM GENERATION

The MCS consists of a COBOL message control system (CMCS) and the integrated communications access method (ICAM).

The ICAM network needs to be defined and generated. The ICAM system generation defines the lines, terminals, and queues to be used by the COBOL object program. It may also specify the queueing algorithms directing messages to various queues based on text content or terminal names.

A CMCS module also must be generated. The CMCS module generation defines the relationship between the symbolic names of the source, destination, queue, subqueues, etc, specified in the COBOL program and the corresponding ICAM names.

The CMCS module may be statically bound with the COBOL object program, or dynamically loaded at execution time. This option and the CMCS module name are specified by the CMCSST=YES/NO and CMCS=module-name parameters. (See Appendix A.)

After the ICAM and CMCS have been generated, the COBOL object program may then be linked and executed.

Detailed procedures for ICAM system generation and CMCS module generation are described in the OS/3 ICAM utilities user guide/programmer reference.

Appendix A. Compiler Options

A.1. GENERAL

The compiler provides a number of options, described in A.2, that the user may specify. The compiler options may be specified on two levels: SYSGEN specification, and compile-time parameters. The SYSGEN specification may be overridden by the compile-time parameters. The compiler performs a consistency check for all specifications. (See A.3.) If none of the compiler options are specified, the following default options are effected:

1. Accept the highest level of the standard language supported and the OS/3 extensions.

2. Produce a source listing with copied text, if any.

3. Produce a diagnostic listing using a page width of 120 characters.

4. Generate an object module that dynamically loads any called programs.

A.2. COMPILER OPTION SPECIFICATION

The SYSGEN specifications for compiler option parameters are entered at SYSGEN time by the COBGEN label card, followed by individual option specification cards and then by the END card. (See OS/3 system installation user guide/programmer reference, UP-8074 (current version) for more information about SYSGEN operations.)

Example:

```
1        10
___________________________
COBGEN
         OBJLST=YES,AXREF=YES
         .
         .
         .
END
```

The compile-time parameters are specified by the PARAM job control statements. See OS/3 job control programmer reference, UP-8217 (current version). The format for the PARAM statements is:

```
// PARAM option-1,option-2...
```

where:

`option-1,option-2...`
Represents compiler option parameters.

Example:

```
// PARAM CPYTXT=NO,IN=PAYROLL/COB3
// PARAM DIAGWN=NO,OBJLST=YES
```

A parameter must be contained wholly on one PARAM card. Any number of PARAM cards are permitted. Within one card, parameters are separated from each other by commas, and the last parameter on the card is terminated by a space.

The parameters used on PARAM statements are described in Table A-1. Compiler defaults are shaded as follows: ▨

Table A—1. Options of the PARAM Statement (Part 1 of 3)

Parameter	Function

AXNON= {YES
NO}
	Suppresses nonreferenced entries in alphabetically ordered cross-reference listing

AXREF= {YES
NO}
	Specifies an alphabetically ordered cross-reference listing

CALLST= {YES
NO}
	Specifies static CALL of subprograms referenced by the literal option. YES indicates that subprograms named by the literal option of the CALL statements are to be linked with the main program. NO indicates that subprograms named either by the literal or identifier option of the CALL statements are to be dynamically loaded when called.

CDMIO= {**YES**
NO
MI}
	YES specifies that consolidated data management will be used for all files except ISAM and SAM disk files.
	NO specifies that DTF interfaces will be used for all files except workstation files.
	MI specifies that CMD interfaces will be used for MIRAM and workstation files only; DTF interfaces will be used for all other files.
	This parameter is applicable only for a Series 90 system that supports both CDM and DTF interfaces.

CMCS=name	Specifies a 1- to 8-character module name of the COBOL communication control system. If this parameter is not specified for a COBOL communication program, a default name, consisting of six characters of the PROGRAM-ID name (left-justified and zero-filled, if necessary) and a suffix of two characters (CM), is used.

CMCSST= {YES
NO}
	YES indicates that the CMCS module is bound with the COBOL object program.
	NO indicates that the CMCS module will be dynamically loaded at execution time.

CPYTXT= {**YES**
NO}
	Includes COBOL library text in source listing

DIAG= {**YES**
NO}
	Specifies a diagnostic listing

DIAGWN= {**YES**
NO}
	Includes warning diagnostics in the diagnostic listing

Table A—1. Options of the PARAM Statement (Part 2 of 3)

Parameter	Function
ERRFIL=module-name/ lfdname	Specifies generation of an error-file element of compile-time diagnostics. The module-name is the 1- to 8-character module name of the element. The lfdname is the 1- to 8-character name of the MIRAM library where the element will be generated. The ERRFIL parameter is ignored unless the IN parameter is also specified. The error-file element is used by the OS/3 editor error file processing facility (@EFP command).
FIPS= 1 2 3 4 5	Specifies a FIPS PUB 21-1 flagging option. See Appendix D.
IMSCOD= YES NO REN	Specifies IMS compatible code; i.e., COBOL programs are to be executed under control of IMS as action programs. When IMSCOD=YES or IMSCOD=REN is specified, the COBOL language elements restricted by IMS are flagged and deleted. YES indicates generation of a shared code action program. REN indicates generation of a reentrant action program.
IN=m-n/f-n	M-n is a 1- to 8-character source module name in the library. F-n is a 1- to 8-character LFD name identifying the file on which the source module resides. If f-n is omitted, the default name YSRC is used.
LIN=filename/ filename/	Filename is a 1- to 8-character LFD name identifying the file or files where the COPY library resides. A maximum of 10 LFD names can be specified, allowing multiple COPY libraries to be searched. If multiple LFD names are specified, they must be separated by stroke (/) characters. If the library-name is specified in the COPY statement (6.6.8), the library-name takes precedence. If the library-name is omitted in the COPY statement, the filename or filenames in the LIN parameter are used. Multiple file names are searched sequentially in the order specified on the LIN parameter. If the parameter is omitted, the name COPY$ is used as the default name of the LIN parameter.
LIST= YES NO	Specifies a source program listing
LNKCON= YES NO	Specifies generation or suppression of linker control statements in the object module
LSTREF= YES NO	Specifies a source listing with definition references
LSTWTH= nnn	Specifies the page width. nnn ranges from 120 through 160. Default value is 120 characters a line.
MAP= YES NO	Specifies an object program locator/map listing
MXNON= YES NO	Suppresses nonreferenced entries in the map listing with cross-references
MXREF= YES NO	Specifies a map listing with cross-references
OBJ=filename	Filename is a 1- to 8-character LFD name of the file on which the generated object module is to be stored. If the parameter is not specified, the default name YRUN is used.
OBJLST= YES NO	Specifies an object program listing
OBJMOD= YES NO	Specifies object module production
PAGOVF= YES NO	YES provides automatic printer page eject feature in the object program. NO indicates omission of the eject feature in the object program. PAGOVF=YES should not be specified if the LINAGE clause or the ADVANCING PAGE phrase is specified in the source program.

Table A—1. Options of the PARAM Statement (Part 3 of 3)

Parameter	Function

PROVER= { YES
NO }
	YES specifies the production of a listing of procedure-names and verbs with associated source line numbers and object program relative addresses. NO indicates suppression of the listing.

SPRLST= { YES
NO }
	Suppresses all listings unconditionally. This parameter overrides all other listing parameters.

SPROUT= { 1
2
3 }
	Suppresses compiler output (except source listing, diagnostic listing, memory map and cross reference listings, and related options) when severity code level 1, 2, or 3 errors are encountered.

SUBCK= { YES
NO }
	YES specifies range checking of subscripts and indices. If a subscript or index exceeds the table size, a CE58 run-time error message is issued (see the system messages operations reference handbook, UP-8076). When NO is specified, the compiler does not generate range-checking code, and the results are unpredictable.

SYNCHK= { YES
NO }
	Specifies syntax check only or normal compilation. When SYNCHK=YES is specified, only the FIPS and LSTWTH parameters may be specified. A source program listing and a diagnostic listing are produced automatically by the compiler.

TRNADR= { YES
NO }
	YES indicates generation of a transfer address in the object module. NO indicates suppression of a transfer address; in which case, the program cannot be executed unless it is called.

TRUNC= { YES
NO }
	YES indicates that data truncation on binary items is based on the decimal digits specified in the PICTURE character-string. NO indicates that data truncation is based on the actual storage size allocated to the items. In either case, SIZE ERROR detection is based on the decimal digit size specified in the PICTURE clause.

A.3. COMPILER OPTION SPECIFICATION CONSISTENCY CHECK

When SYSGEN specifications and the compile-time parameters are read, the compiler inspects the resulting values for consistencies. If inconsistent specifications exist, the compiler resets the values of certain options to make them logically consistent. The consistency check is performed in the order shown in the left-hand column of Table A-2, and the values of certain parameters are reset as shown in the right-hand column of Table A-2.

Table A—2. Parameter Consistency Checks

User Specifications		Compiler Actions	
Parameter	Value	Parameter	Value
LIST	NO	LSTREF	NO
LIST	NO	CPYTXT	NO
MXNON	YES	MXREF	YES
MXREF	YES	MAP	YES
AXNON	YES	AXREF	YES
IMSCOD	YES	CALLST	YES
IMSCOD	REN	CALLST	YES
OBJMOD	NO	PROVER	NO
SYNCHK	YES	LIST	YES
SYNCHK	YES	DIAG	YES
SYNCHK	YES	OBJLST	NO
SYNCHK	YES	OBJMOD	NO
SYNCHK	YES	MAP	NO
SYNCHK	YES	MXREF	NO
SYNCHK	YES	AXREF	NO
SYNCHK	YES	PROVER	NO
SYNCHK	YES	LNKCOM	NO
SYNCHK	YES	PAGOVF	NO
SYNCHK	YES	TRNADR	NO
SPRLST	YES	LIST	NO
SPRLST	YES	LSTREF	NO
SPRLST	YES	CPYTXT	NO
SPRLST	YES	OBJLST	NO
SPRLST	YES	MAP	NO
SPRLST	YES	MXREF	NO
SPRLST	YES	AXREF	NO
SPRLST	YES	DIAG	NO

Appendix B. Compiler Listings

The compiler produces seven output listings directed to the SYSLST file. The listings, shown in the order in which they are produced by the compiler, are:

Compilation summary
Diagnostic
Source
Object code
Locator/map
Alphanumerically ordered cross-reference
Object code map listing

B.1. COMPILATION SUMMARY LISTING

The compilation summary listing contains the volume (VOL), label (LBL), and LFD information for all library files accessed by the compiler.

B.2. DIAGNOSTIC LISTING

The diagnostic listing contains a diagnostic message for each source program error other than sequence number errors. Detailed information concerning diagnostic listings and messages is given in Appendix C.

B.3. SOURCE LISTING

The source listing shows the source program as it is compiled.

Compiler-generated line numbers are displayed to the left of each line compiled, regardless of whether it came from the basic source or from a COPY statement. The compiler-generated line numbers are used in all diagnostic messages.

The sequence numbers of the input source program are checked for proper order and any sequence errors are flagged with an S to the left of the source line in error. If a source line has spaces in columns 1 through 6, no sequence check is performed on that line. A line is considered to be in sequence if the nonblank value in columns 1 through 6 is higher (in terms of the EBCDIC alphanumeric collating sequence) than the previous nonblank value in columns 1 through 6.

Sequence checking of columns 1 through 6 is not done for text copied in via a COPY statement.

When a COPY statement occurs in the main input source, the COPY statement is printed in its original form; the copied text of the library entry is then printed as it appears after any replacement specified in the COPY statement has taken place. The copied text is flagged with a C in the left-hand margin, or an R if the text has been modified by a REPLACING phrase.

If the SUPPRESS COPIED TEXT option is specified, the copied source is not listed but it is still assigned compiler line numbers.

When the LSTREF option is specified, the source listing will also have, to the right of each line that references a data item or procedure, the compiler-generated line number of the definition of that item.

B.4. OBJECT CODE LISTING

The object code listing shows the compiler-generated line number with the corresponding object code. It also shows the local base register and displacement as well as an object program relative location counter. It shows the DTF expansion code, the machine language instruction or constant, object program relative operand addresses, and assembly language mnemonic corresponding to the machine language op code. It also shows a comments field that contains information useful in relating this portion of the object program to the source program.

Explanatory comments in the fixed-code portion of the object module are especially important in helping the programmer understand the logic of control in the COBOL object program.

B.5. LOCATOR/MAP/CROSS-REFERENCE LISTINGS

The locator/map listing consists of the data division storage map and the procedure division storage map. If the MXREF parameter is specified, the MAP listings also contain the compiler-generated line numbers of the COBOL statements in which the data-name or procedure-name is referenced.

- Data Division Storage Map

  The data division storage map shows the layout of the object program main storage for the data division. It shows the compiler-generated line number, the name, the level number or level indicator, the base register and displacement, the relative address, and the data type, size, and number of occurrences of the data item. It also shows any mnemonic-names defined in the SPECIAL-NAMES paragraph of the environment division.

  The base register field in the listing identifies a permanently assigned machine register (R5 through R8) or a 2-digit number that represents an entry in a table of base register values. This table is called the base address tags table and is shown on the first page of the data division storage map listing.

  The address field in the listing identifies the object program relative address of the data item or, for linkage section data items, the displacement within a level 01 record description. (The address of linkage section data items is determined only at object program execution time.)

- Procedure Division Storage Map

  The procedure division storage map shows the layout of the object program main storage for the procedure division. It contains the compiler-generated line number of the procedure, and an indication as to whether the procedure is a section or paragraph, the segment number of the section, the starting address of the procedure, and indicators as to whether the procedure is referenced by GO TO or PERFORM statements or USE FOR DEBUGGING procedures. Paragraph-names are indented to show their inclusion within a section. The beginning and the end of the declaratives are indicated. The CSECT names of the segments in a segmented program are also shown.

The procedure division storage map also displays PERFORM statement exit bucket addresses. An exit bucket is a save area in main storage that holds return linkage information. The exit buckets are helpful in evaluating object program main storage dumps.

Each procedure-name that is a PERFORM statement exit point is designated by a PX (perform exit point) or SX (sort input/output procedure exit point) on the procedure division storage map listing. Each PX or SX has an exit bucket associated with it. The address of the exit bucket is shown in parentheses next to the PX or SX designation. At the end of the storage map listing, the exit bucket addresses are listed in address order and next to each exit bucket address is the line-number of its associated procedure-name. In a reentrant action program, the address of the exit bucket is actually a displacement into the object program reentrancy control area (see G.4).

Each exit bucket is eight bytes long and has an initial setting of binary zeros. When a PERFORM statement is executed, the return address (usually the address of the statement after the PERFORM statement) is stored in the first four bytes and the local cover register value for the PERFORM statement object code is stored in the next four bytes. When the end of a PERFORM range is reached and program execution returns to the point following the PERFORM statement, the first four bytes are reset to binary zeros.

When an exit bucket statement is all binary zeros, no PERFORM statement was executed for that procedure.

When the first four bytes are binary zeros and the next four are not, a PERFORM statement was executed for the procedure and the procedure reached its exit point.

When all eight bytes are not binary zeros, a PERFORM statement was executed for the procedure and the procedure has not yet reached its exit point. The first four bytes identify the particular PERFORM statement that was executed.

If the MXNON parameter is specified, nonreferenced names are not shown in the listing.

B.6. ALPHABETICALLY ORDERED CROSS-REFERENCE LISTING

The alphabetically ordered cross-reference listing shows the program special registers and user-defined names in ascending alphanumeric sequence by name. It also contains the compiler-generated line number on which the name is defined and the compiler-generated line numbers of COBOL statements in which the name is referenced. If the AXNON parameter is specified, nonreferenced names are not shown in the listing.

B.7. OBJECT CODE MAP LISTING

The object code map listing shows the compiler-generated symbols for each procedure, and each statement within a procedure, as well as the object program relative address assigned to each.

Appendix C. Compiler Diagnostics

C.1. GENERAL

This appendix contains the compile-time diagnostic listing messages. The console and terminal messages relating to the compilation process and the run-time error messages are given in the OS/3 systems messages programmer/operator reference, UP-8076 (current version).

C.2. DIAGNOSTIC LISTING

The diagnostic listing contains a diagnostic message for each error encountered in the source program other than sequence number errors. Each diagnostic message contains the compiler-generated line number on which the error occurred, a diagnostic message number, a severity code associated with the type of error, and a diagnostic message text.

There are four severity codes:

Code   Description

0   This is for a precautionary or warning diagnostic. The source program contains a legal but potentially undesirable situation.

1   This is for a conditional, or changed, diagnostic. An error has been encountered in the source program, but the compiler has been able to make a corrective assumption and continue processing.

2   This is for a serious error for which the compiler is not able to make any corrective assumption. The statement containing the error has been deleted.

3   A fatal error situation has been encountered. The compilation is continued for error checking purposes, but recompilation is necessary.

The diagnostic text consists of a concise description of the error, including any possible compiler recovery action.

At the end of the diagnostic listing, there is a summary report of the number of each type of diagnostic encountered and the number of source sequence errors detected. If there are no errors encountered, an explicit message is given.

The following chart shows compile-time diagnostic messages, their message numbers, and severity codes. The chart also includes message explanations and recovery procedures.

Message Number	Severity Code	Message/Explanation/Action
0001	0	**INPUT RECORD EXCEEDS 80 CHARACTERS. EXCESS CHARACTERS TRUNCATED.** A source statement of over 80 characters was encountered. All characters past position 80 of the input record are deleted.
0002	1	**NONBLANK CHARACTERS APPEAR IN AREA A OF A CONTINUATION LINE. CHARACTERS ACCEPTED.** A nonblank character has been found in area A (columns 8 to 11) when continuation has been specified by a hyphen in column 7. The first nonblank character after column 7 is accepted as the beginning of continuation.
0003	1	**ILLEGAL CHARACTER IN COLUMN 7. SPACE ASSUMED.** An invalid character has been found in column 7. A space is assumed to have been found in column 7.
0004	2	**COPY STATEMENT APPEARS IN COPIED TEXT. IMBEDDED WORD COPY IGNORED.** Text encountered while processing a COPY statement includes the word COPY. The word COPY is ignored.
0005	1	**NONNUMERIC LITERAL IN A CONTINUATION LINE NOT BEGIN WITH QUOTATION MARK. CONTINUATION STARTS WITH FIRST NONBLANK CHARACTER.** The continued portion of a nonnumeric literal did not begin with a quote or apostrophe. Processing continues as if a quote or apostrophe occurred prior to the first nonblank character.
0006	1	**CHARACTER char-string NOT PRECEDED BY A SPACE. A SPACE ASSUMED.** A space that should be used to delimit two characters or character strings was not found. A space is assumed to precede the character.

Message Number	Severity Code	Message/Explanation/Action
0007	1	**CHARACTER STRING char-string EXCEEDS PERMISSIBLE LENGTH. EXCESS CHARACTERS TRUNCATED.** A character string that is greater than its maximum legal size has been detected. The first 30 characters of the string are noted in the diagnostic. Processing continues after the excessive characters are discarded.
0008	1	**INVALID CHARACTER char-string. CHARACTER IGNORED.** A character in the COBOL character set was used incorrectly. The character is ignored.
0009	1	**COLUMN number CONTAINS AN ILLEGAL CHARACTER char-string. CHARACTER IGNORED.** A character not in the COBOL character set was encountered. The character is ignored.
0010	1	**NONNUMERIC LITERAL OF ZERO LENGTH. A LITERAL OF ONE SPACE ASSUMED.** Two quotes or apostrophes with no intervening characters were encountered. A nonnumeric literal of one space character is assumed.
0011	1	**NONNUMERIC LITERAL NOT TERMINATED BY QUOTATION MARK NOR CONTINUED ON NEXT LINE. LITERAL TERMINATED AT LAST NONBLANK CHARACTER OF CURRENT LINE.** There is no terminating quote or apostrophe on the source line and no hyphen in column 7 of the next source line. The nonnumeric literal is terminated at the last nonblank character of the current line.
0012	2	**language-element NOT IMPLEMENTED. LANGUAGE ELEMENT IGNORED.** The COBOL language element encountered is not implemented. The language element is not processed.
0013	0	**LANGUAGE ELEMENT xxx EXCEEDS SPECIFIED FIPS LEVEL. ELEMENT BELONGS TO LEVEL nnn. NO CORRECTIVE ACTION TAKEN.** The language element used in the program exceeds the specified FIPS processing level. The language element is accepted.

Message Number	Severity Code	Message/Explanation/Action
0014	1	**OPTIONAL CLAUSE, DATA-NAME, OR A LEVEL 01 ENTRY NOT SPECIFIED WITH THE CD ENTRY.** If the optional clauses or data-names are not specified in the CD entry, a level 01 data description must follow the CD entry. For input CD, an area of 87 characters with FILLER items is assumed. For output CD, an area of 23 characters with FILLER items is assumed. Source corrections and recompilation are required if the CD area is referenced in the program.
0015	1	**SPECIFIED SEGMENT-LIMIT GREATER THAN 49. SEGMENT-LIMIT 49 ASSUMED.** The value of integer specified in the SEGMENT-LIMIT clause must be within the range from 1 to 49. Segment-limit of 49 is assumed.
0016	2	**ALPHABET-NAME OR CLASS-NAME SPECIFICATION ERROR. CLAUSE INCOMPLETE.** An integer or literal may have been specified more than once. The remainder of this clause is ignored.
0017	1	**THE OPTIONAL PHRASE SPECIFIED FOR ORGANIZATION OTHER THAN SEQUENTIAL OR SAM, OR FOR A DEVICE WHICH CANNOT BE OPENED INPUT. PHRASE IGNORED.** The OPTIONAL phrase was specified for a file with relative, indexed, or ISAM organization, or for a device that cannot be opened inut. The phrase is ignored.
0018	1	**DEVICE NAME IN element CLAUSE NOT VALID. DISC-*DUMMY-V ASSUMED.** The device type in the ASSIGN or RERUN clause is invalid. Processing continues with the assumed name.
0019	2	**IMPLEMENTOR-NAME SPECIFIED IN ASSIGN CLAUSE INCOMPLETE. DISC-*DUMMY-V ASSUMED.** The implementor-name format is device-lfdname-mode. The lfdname-mode was not specified. Processing continues with the assumed name.
0020	1	**LINK NAME IN element CLAUSE EXCEEDS 8 CHARACTERS. FIRST 8 CHARACTERS USED.** The lfdname (same as linkname) may not exceed eight characters. Characters beyond the first eight are truncated.

Message Number	Severity Code	Message/Explanation/Action
0021	1	**MODE INVALID OR NOT SPECIFIED IN ASSIGN CLAUSE. DISC-*DUMMY-V ASSUMED.** The mode in the implementor-name of the ASSIGN clause is not specified or specified incorrectly. The values assumed are DISC for device, *DUMMY for lfdname, and V for mode.
0022	1	**CLASS-NAME CLAUSE SPECIFIED WITH NO VALUE PHRASE OR SOURCE-ALPHABET CLAUSE. COBOL CHARACTER SET USED.** When the VALUE phrase is omitted in the CLASS-NAME clause, the SOURCE-ALPHABET clause must be specified. The COBOL character set is used for the CLASS-NAME mnemonic-name test.
0023	1	**ASSIGN CLAUSE NOT SPECIFIED IN SELECT SENTENCE. NO CORRECTIVE ACTION TAKEN.** The ASSIGN clause is missing in the SELECT sentence. Processing continues as if an ASSIGN clause specifying a DISK device were encountered.
0024	0	**INTEGER SPECIFIED IN RESERVE AREA CLAUSE GREATER THAN 2. TWO AREAS ASSUMED.** The value of the integer in the RESERVE clause may not exceed 2. RESERVE 2 AREAS assumed.
0025	1	**ILLEGAL ACCESS MODE SPECIFIED FOR SEQUENTIAL ORGANIZATION. SEQUENTIAL ACCESS ASSUMED.** RANDOM or DYNAMIC ACCESS MODE was specified for a sequential file. Sequential access mode is assumed.
0026	1	**SPECIFIED KEY(S) NOT VALID FOR THE FILE ORGANIZATION. ILLEGAL KEYS IGNORED.** The key or keys specified for the file is not allowed for the file organization. Illegal key or keys ignored.
0027	1	**RELATIVE KEY NOT SPECIFIED FOR RELATIVE FILE WITH RANDOM OR DYNAMIC ACCESS. NO CORRECTIVE ACTION TAKEN.** The RELATIVE KEY clause must be specified for a relative file with random or dynamic access. Processing continues.

Message Number	Severity Code	Message/Explanation/Action
0028	2	**RECORD KEY NOT SPECIFIED FOR INDEXED OR ISAM FILE. NO CORRECTIVE ACTION TAKEN.** The RECORD KEY clause must be specified for an INDEXED or ISAM file. Processing continues.
0029	2	**NUMBER OF ALTERNATE KEYS SPECIFIED EXCEEDS LIMIT. THIS KEY IGNORED.** Alternate keys for a file may not exceed 4. The key specified on the line indicated by the line number is ignored.
0030	2	**OCCURS CLAUSE SPECIFIED IN LEVEL 01 OR 77 ENTRY. CLAUSE IGNORED.** The OCCURS clause may not be specified for a level 01 or 77 entry. The OCCURS clause is ignored.
0031	2	**INTEGER-1 OR INTEGER-2 IN OCCURS CLAUSE EXCEEDS LIMIT. IF INTEGER-1, IT IS SET TO 1; IF INTEGER-2, IT IS IGNORED.** The value of integer-1 or integer-2 specified in the OCCURS clause exceeds limit. If integer-1 exceeds limit, it is set to 1. If integer-2 exceeds limit, it is ignored.
0032	1	**integer NOT A VALID LEVEL NUMBER. LEVEL 49 ASSUMED.** An invalid level number was encountered. Level number 49 is assumed.
0033	2	**UNSIGNED NONZERO INTEGER EXPECTED WHERE xxx SPECIFIED IN element CLAUSE. CLAUSE IGNORED.** An unsigned nonzero integer is expected in clause analysis. The clause is ignored.
0034	1	**FILE-NAME xxx APPEARS MORE THAN ONCE IN element CLAUSE(S). FIRST SPECIFICATION USED.** The file name appears more than once as a rerun controller, or in the SAME AREA or MULTIPLE FILE TAPE clause. Only the first specification is used.
0035	1	**FILE-NAME xxx IN element CLAUSE NOT SPECIFIED IN SELECT CLAUSE. FILE-NAME IGNORED.** The file name was not found in the SELECT clause. The file name is ignored.

Message Number	Severity Code	Message/Explanation/Action
0036	2	**FILE-NAME xxx ALREADY SPECIFIED IN A SELECT CLAUSE. CLAUSE DELETED.** The same file name was specified in a previous SELECT clause. The SELECT clause is syntax-checked and ignored.
0037	1	**PERIOD MISSING IN** element. **PERIOD ASSUMED.** A character other than a period was found where a period was expected. A period is assumed.
0038	2	**LINK-NAME IN RERUN CLAUSE DOES NOT MATCH ANY SELECT LINK-NAME. RERUN CLAUSE IGNORED.** The lfdname (same as linkname) in the OS/3 RERUN ON lfdname option did not match with a lfdname in any of the SELECT clauses. The RERUN clause is ignored.
0039	2	**RERUN RECEIVER NAME FORMAT ERROR. RERUN CLAUSE IGNORED.** The implementor-name in the RERUN clause must be in the device-lfdname-option format. The RERUN clause is ignored.
0040	1	**OPTION 1 OR 2 NOT SPECIFIED IN RERUN RECEIVER NAME. OPTION 1 ASSUMED.** The value 1 or 2 was not specified in the implementor-name of the RERUN clause. One dedicated receiver file is assumed.
0041	1	**VALUE OF INTEGER IN RERUN CLAUSE EXCEEDS LIMIT. VALUE SET TO 5000.** The value of integer in the RERUN clause may not exceed 8,388,607. The value of integer is set to 5000.
0042	1	**VALUE OF INTEGER IN MULTIPLE FILE POSITION CLAUSE EXCEEDS 256. VALUE SET TO PREVIOUS POSITION PLUS 1.** Self-explanatory. Processing continues as if a value equal to the previous position plus 1 were specified.
0043	2	**FILE-NAME xxx IN MULTIPLE FILE CLAUSE NOT DEFINED AS TAPE FILE. FILE-NAME IGNORED.** The file name in the MULTIPLE FILE TAPE clause was assigned to a device other than tape. The file name is ignored.

Message Number	Severity Code	Message/Explanation/Action
0045	2	**ILLEGAL CHARACTER STRING char-string. CHARACTER STRING IGNORED.** A set of characters that does not form any COBOL word or literal was encountered. The string is ignored.
0046	1	**A FILE WITH ORGANIZATION SAM IS ASSIGNED TO A DEVICE OTHER THAN DISK. ORGANIZATION SEQUENTIAL ASSUMED.** A file specified with ORGANIZATION IS SAM* clause must be assigned to a disk. ORGANIZATION SEQUENTIAL is assumed for the file.
0047	1	**TEXT-NAME OR LIBRARY-NAME char-string EXCEEDS 8 CHARACTERS. FIRST 8 CHARACTERS USED.** A library or file name of a copy module is longer than eight characters. The first eight characters of the name provided are used.
0048	2	**division-name DIVISION HEADER MISSING OR OUT OF SEQUENCE. NO CORRECTIVE ACTION TAKEN.** A division header other than the next one in sequence was encountered. Processing continues with the division header encountered.
0049	3	**type name CANNOT BE ACCESSED. COPY STATEMENT IGNORED.** The text or library used in a COPY statement cannot be accessed. There was an I/O error while trying to open the library file or the text requested was not on the library file. The COPY statement is ignored. Scanning of the source program for other COPY statements and reference format errors is performed, after which compilation is terminated.
0050	2	**FILE file-name IN APPLY CLAUSE NOT DEFINED AS xxx FILE. FILE-NAME IGNORED.** 1. File referenced in APPLY VERIFY clause was assigned to a device other than disk. 2. File referenced in APPLY BLOCK-COUNT clause was assigned to a device other than tape. 3. File referenced in APPLY CYLINDER-OVERFLOW, CYLINDER-INDEX, or INDEXED-AREA clause was specified with an organization other than ISAM*. The file-name in the APPLY clause or the clause itself is ignored.

**Applies only to 90/25, 90/30, 90/30 B, and 90/40 systems*

Message Number	Severity Code	Message/Explanation/Action
0051	2	**WORD xxx DOES NOT BEGIN ANY CLAUSE. PROCESSING CONTINUES WITH NEXT CLAUSE.** A new clause was expected but the word encountered does not begin any clause. All words are ignored until a word that begins an appropriate clause is found.
0052	2	**element CLAUSE NOT IMPLEMENTED. CLAUSE IGNORED.** Report writer feature is not implemented. The language element is ignored.
0053	1	**NUMERIC LITERAL nnnn EXCEEDS 18 DIGITS. TRUNCATED TO nnnn.** A numeric literal may not exceed 18 digits. The literal is truncated from the right.
0054	1	**NUMBER OF RECORDS IN LINKAGE SECTION NOT EQUAL TO NUMBER OF ARGUMENTS IN USING LIST. USING LIST ACCEPTED.** The LINKAGE section should correspond to the arguments in the USING list. There are more records in one than the other. The USING list is accepted and compilation continues. Errors occur if items not mentioned in both lists are used.
0055	2	**SERIOUS ERROR IN USE STATEMENT. item INVALID. SECTION section-name DELETED.** The named item in the USE statement is invalid. The entire section for the USE statement is deleted.
0058	2	**LEVEL 88 condition-name ENTRY NOT PRECEDED BY CONDITIONAL VARIABLE. LEVEL 01 FILLER ENTRY GENERATED.** The level 88 entry is the first entry in the data division. The compiler creates a level 01 named FILLER, length 1, signed for the conditional variable.
0059	2	**LEVEL 66 data-name ENTRY NOT AT END OF HIERARCHY. LEVEL 01 FILLER ENTRY GENERATED.** The level 66 entry was not followed by one of the following: a level 01 entry, an FD or SD entry, a level 77 entry, a level 66 entry, or a PROCEDURE DIVISION header. A level 01 named FILLER is created to follow the level 66 entry.

Message Number	Severity Code	Message/Explanation/Action
0061	2	**LEVEL** number **ENTRY NOT SUBORDINATE TO LEVEL 01 ENTRY. LEVEL 01 FILLER ENTRY GENERATED.** A data entry with a level between 02 and 49 follows a level 77 or DATA DIVISION header. A level 01 named FILLER is created to precede the data entry.
0062	2	**CONSISTENCY ERROR:** element clause **INVALID WHEN USED WITH** element clause. **FIRST CLAUSE DELETED.** Conflict between description clauses of the data entry, i.e., USAGE COMP-3 and alphanumeric PICTURE. The first clause is ignored.
0063	0	**GO TO DEPENDING STATEMENT CONTAINS ONLY ONE PROCEDURE-NAME. NO CORRECTIVE ACTION TAKEN.** At least two procedure names are required in a GO TO statement with the DEPENDING option. Control is transferred to procedure name if value of identifier is 1. Otherwise, control is passed to the next sentence.
0064	2	**PICTURE CLAUSE INVALID FOR GROUP ITEM** data-name. **PICTURE CLAUSE DELETED.** The data entry was determined to be a group item from level number structure and a PICTURE clause conflicts with a group entry. The compiler deletes the PICTURE clause on the group item.
0065	2	**IMS ENVIRONMENT PROHIBITS USE OF LANGUAGE ELEMENT** xxx. **ELEMENT DELETED.** The specifying element is not allowed under IMS processing mode. The specified element is selected.
0066	2	**PROCEDURE DIVISION USING REQUIRED IN IMS ENVIRONMENT. NO CORRECTIVE ACTION TAKEN.** Procedure division USING must be present in the IMS/90 environment. No action is taken by the compiler.
0067	2	**ALL PROCEDURES PHRASE SPECIFIED MORE THAN ONCE. EXCESS CLAUSE DELETED.** The ALL PROCEDURES clause can appear only once in a program. No action is taken by the compiler.

Message Number	Severity Code	Message/Explanation/Action
0068	0	**LITERAL literal-string TRUNCATED AND MOVED TO element.** The literal contains more character positions than the receiver or contains more digits than the receiver when it is aligned on the decimal point. The literal is truncated and moved.
0070	1	**BLOCK OR RECORD SIZE FOR FILE file-name EXCEEDS 32,767. SIZE SET TO 32,767.** The block size (buffer size for MIRAM files) or record size is larger than the OS/3 limit of 32,767 bytes. OS/3 block length headers, variable record length headers, and physical I/O BCW data length fields use an unsigned half word to contain the length value (15 bits or a maximum length of 32,767). The actual permitted length is often less than 32,767 due to disk track capacities, multiplexer channel limits, etc. The size is set to 32,767.
0073	0	**SIZE OF LEVEL 01 REDEFINING AREA data-name UNEQUAL TO SIZE OF LEVEL 01 REDEFINED AREA.** The REDEFINES clause is valid. The size of each area conforms to the exact description of each entry. This message is for warning only.
0074	1	**USAGE OF data-name CONFLICTS WITH USAGE OF GROUP. NO CORRECTIVE ACTION TAKEN.** A data entry usage conflicts with the usage of one or more of the group entries that this data entry is subordinate to, or usage conflicts with a value on a group level. The compiler assumes group entry's usage as proper usage.
0075	2	**OCCURS CLAUSE IN data-name ENTRY INVALID. 4 DIMENSION TABLE DESCRIBED. CLAUSE DELETED.** A data entry with an OCCURS clause was encountered, which would cause more than three levels of subscripting. The compiler deletes the OCCURS clause on the data entry.
0076	2	**FILE file-name HAS NO DATA RECORD. NO CORRECTIVE ACTION TAKEN.** A level 01 data record was not encountered for this file. No action is taken by the compiler.

Message Number	Severity Code	Message/Explanation/Action
0077	2	**PRINTER CONTROL CHARACTER SPECIFICATION INVALID FOR file-name. SPECIFICATION IGNORED.** Printer control character may only be specified on file with sequential organization. Compiler ignores control character specification.
0078	3	**ADDITIONAL MEMORY REQUIRED FOR PROCESSING LABEL RECORD DESCRIPTIONS. OBJECT MODULE NOT PRODUCED.** There is not enough main storage available for holding all the label name definitions for this file. Compiler assumes that label name definitions that will not fit do not exist. Main storage is required to hold the SELECTS and label name definitions. To allow processing of more label names, allocate more main storage, shorten the size of the SELECT entries, or define fewer label names.
0079	1	**BLOCK LENGTH FOR FILE file-name NOT A MULTIPLE OF RECORD LENGTH. BLOCK SHORTENED TO HOLD AN INTEGRAL NUMBER OF RECORDS.** The block length is not evenly divisible by the record length for fixed mode tape, card reader, card punch, fixed mode SAM, or ISAM. The block is shortened to contain an integral number of records.
0080	1	**FILE-NAME file-name NOT FOUND IN SELECT CLAUSES. A SELECT ENTRY ASSIGNED TO DISK ASSUMED.** A file that does not have a SELECT entry (matched by file name) was encountered. Compiler assumes a SELECT entry defined with file name and assigned to disk device.
0081	1	**INVALID MODE SPECIFIED FOR FILE file-name. MODE F ASSUMED.** The mode specified is not permissible for this file. Mode F is assumed.
0083	1	**BLOCK CONTAINS MORE THAN 1 RECORD NOT PERMITTED FOR FILE file-name. BLOCK CONTAINS 1 RECORD ASSUMED.** A file assigned to U mode tape or printer with a BLOCK CONTAINS clause specifying more than one record was encountered. Compiler assumes block contains one record.
0084	1	**LABEL RECORDS OMITTED REQUIRED FOR FILE file-name. OMITTED ASSUMED.** A file assigned to a unit record device with other than LABEL RECORDS OMITTED was encountered. Compiler assumes labels to be omitted.

Message Number	Severity Code	Message/Explanation/Action
0085	1	**BLOCK SIZE SPECIFIED FOR FILE** file-name **INSUFFICIENT FOR MINIMUM I/O BUFFER. MINIMUM SIZE ASSUMED.** The BLOCK CONTAINS CHARACTERS clause specifies a buffer size insufficient for the minimum data management buffer. The minimum allowable buffer size is used.

Message Number	Severity Code	Message/Explanation/Action
0087	2	**RECORD DESCRIPTION ENTRY FOR LABEL RECORD data-name NOT FOUND. NO CORRECTIVE ACTION TAKEN.** A label name (from LABEL RECORDS ARE clause) with no 01 label description was encountered. The compiler assumes that the label name does not exist.
0089	1	**LABEL RECORDS STANDARD REQUIRED FOR FILE file-name. STANDARD ASSUMED.** A file assigned to mass storage device must specify LABEL RECORDS STANDARD clause. LABEL RECORDS STANDARD clause assumed.
0091	2	**SYNTAX REQUIRES element, char-string INVALID. STATEMENT IGNORED.** The character string encountered does not conform to the language element required by the COBOL syntax. Processing continues at the end of the statement.
0092	3	**MEMORY INSUFFICIENT FOR element PROCESSING. OVERFLOW DETECTED ON char-string. OBJECT MODULE NOT PRODUCED.** Processing of the language element could not continue due to insufficient storage. All previous processing of the language element is deleted and current processing of the character string is completed.
0093	1	**LITERAL 0 INVALID FOR SIGN CONDITION TEST. FIGURATIVE CONSTANT ZERO ASSUMED.** Literal 0 may not be specified in a sign condition test. Figurative constant ZERO substituted.
0094	1	**CHARACTER IN CHARACTER POSITION integer INVALID FOR type PICTURE pic-string. PICTURE SET TO S9.** An illegal PICTURE character, a PICTURE character inconsistent with the PICTURE type, or a violation of the PICTURE precedence rules has been detected. In order not to delete the data descriptor, the compiler sets the picture to S9.
0095	1	**type PICTURE pic-string INCOMPLETE. PICTURE SET TO S9.** As stated, the picture is incomplete and cannot be processed, e.g., SPPPP. In order not to delete the data descriptor, the compiler sets the picture to S9.

Message Number	Severity Code	Message/Explanation/Action
0097	1	**SIZE LIMIT OF integer BYTES EXCEEDED BY PICTURE pic-string. PICTURE SET TO S9.** The PICTURE clause specifies more storage than the maximum allowed for the picture type. In order not to delete the data descriptor, the compiler sets its picture to S9.
0098	1	**PICTURE pic-string EXCEEDS 18 DIGIT POSITIONS. PICTURE SET TO S9.** The number of digit positions in the picture exceeds 18. In order not to delete the data descriptor, the compiler sets the picture to S9.
0099	1	**VALUE OF INTEGER IN PARENTHESIS EQUALS ZERO OR GREATER THAN 4092 IN PICTURE pic-string. INTEGER SET TO 1.** A value contained within parentheses is either 0 or greater than 4092. The value within the parentheses is set to 1 and processing of the picture continues.
0100	1	**INTEGER DOES NOT FOLLOW LEFT PARENTHESIS IN PICTURE pic-string. PICTURE SET TO S9.** A left parenthesis within the PICTURE clause is not followed by a numeric integer. In order not to delete the data descriptor, the compiler sets the picture to S9.
0101	1	**RIGHT PARENTHESIS MISSING FROM PICTURE pic-string. PICTURE SET TO S9.** A right parenthesis does not follow a numeric integer preceded by a left parenthesis. In order not to delete the data descriptor, the compiler sets the picture to S9.
0102	1	**BOTH LEADING AND TRAILING SIGN INSERTION SPECIFIED IN PICTURE pic-string. PICTURE SET TO S9.** Two insertion sign characters have been encountered in the numeric edited picture. In order not to delete the data descriptor, the compiler sets the picture to S9.
0105	1	**INITIAL VALUE EXCEEDS SIZE OF DATA ITEM. EXCESS CHARACTERS TRUNCATED.** The value specified for the data item contains a greater number of characters than the data item, or is a numeric value that, when the decimal point is aligned, is larger than the maximum value the data item can contain. The excess characters are truncated.

Message Number	Severity Code	Message/Explanation/Action
0106	2	**INVALID POSITIONING OF KEY data-name IN HIERARCHY. KEY CLAUSE IGNORED.** There must not be any items with an OCCURS clause between the table item and its keys. The named key is processed as a regular data item, the key information is ignored.
0107	3	**ADDITIONAL MEMORY REQUIRED TO PROCESS HIERARCHY CONTAINING data-name. OBJECT MODULE NOT PRODUCED.** There is not enough main storage available to contain all entries subordinate to the 01 data entry. There are too many entries for the 01 hierarchy for main storage allocated. The compiler will not process the data entries that are not contained in main storage. To compensate, shorten the hierarchy, shorten names in data entries, or assign more main storage to compiler.
0108	3	**data-name EXCEEDS REDEFINES NESTING LIMIT. REDEFINES CLAUSE IGNORED.** There are too many levels of redefinition. This data entry exceeds the limit of redefinition. The compiler assumes this entry does not have REDEFINES clause.
0109	1	**data-name-1 HAS IMPROPER REDEFINES OBJECT data-name-2. OBJECT OF REDEFINES ASSUMED TO BE THE LAST DEFINED AREA.** The redefined area is a redefining area; i.e., the object of the REDEFINES clause has or is subordinate to a REDEFINES clause. The compiler assumes the redefinition of the last defined area with the same level as the subject of the REDEFINES clause.
0111	2	**DATA DESCRIPTION OF data-name NOT FOUND IN HIERARCHY. QUALIFIER IGNORED.** The definition of the entry is not in the current hierarchy. The compiler assumes the qualifier name in error does not exist.
0112	1	**RENAMES-OCCURS CONFLICT BETWEEN data-name-1 AND data-name-2. LAST ELEMENTARY ITEM ASSUMED AS OBJECT OF RENAMES.** The object of the RENAMES clause on data-name-1 has or is subordinate to an OCCURS clause. The compiler assumes the last elementary item in the hierarchy is the object of the RENAMES clause.

Message Number	Severity Code	Message/Explanation/Action
0113	1	**SIZE OF REDEFINING AREA data-name UNEQUAL TO SIZE OF REDEFINED AREA. LARGER SIZE ASSUMED FOR THE AREA.** The calculated length of the redefined area is not the same as the length of the redefining area. The larger size is used for the area.
0114	1	**SIZE OF ELEMENTARY ITEM data-name EXCEEDS LIMIT. SIZE SET TO LIMIT.** An elementary item with a length larger than the maximum was encountered. The compiler assumes the length to be 4092 for the elementary item.
0115	1	**SIZE OF GROUP ITEM data-name EXCEEDS LIMIT. SIZE OF GROUP ITEM SET TO 524,280. ENTIRE AREA SPECIFIED IS, HOWEVER, ALLOCATED.** The calculated length of a group item exceeds maximum limit. The length of group item is set to 524,280. The entire area specified, however, is allocated.
0116	1	**SIZE OF ITEM data-name CONTAINING AN OCCURS CLAUSE EXCEEDS LIMIT. ENTIRE AREA SPECIFIED IS ALLOCATED BUT SUBSCRIPTED REFERENCES TO THE TABLE MAY NOT GIVE CORRECT RESULTS.** The length of a table element (i.e., the size of the item containing an OCCURS clause) exceeds the maximum limit of 32,767 bytes. The compiler allocates the entire area for the table element; however, subscripted references to the table item may not work.
0117	1	**INVALID LEVEL NUMBER STRUCTURE ENCOUNTERED AT data-name. NO CORRECTIVE ACTION TAKEN.** A level number equal to the level of the data entry should have appeared in the hierarchy directly subordinate to the 01. The compiler assumes there was a level number on a data entry directly subordinate to the 01, e.g., a record with level number structure of 01...05...02 is processed as if it had level number structure of 01...02...05...02.
0118	1	**FIRST OBJECT OF LEVEL 66 ENTRY data-name ENDS AFTER SECOND OBJECT. SECOND OBJECT IGNORED.** The first object of a RENAMES clause does not precede the area of the second object of the RENAMES clause. The compiler assumes the second object does not exist.

Message Number	Severity Code	Message/Explanation/Action
0119	1	**SECOND OBJECT OF LEVEL 66 ENTRY data-name STARTS BEFORE FIRST OBJECT. ORDER OF OBJECTS ASSUMED REVERSED.** The second object of a RENAMES clause does not precede the first object of the RENAMES clause. The compiler assumes the objects are reversed. (The first is the second and the second is the first.)
0120	1	**USAGE INDEX CLAUSE INVALID FOR CONDITIONAL VARIABLE data-name. CLAUSE IGNORED.** A condition name entry is defined for a data with a USAGE INDEX clause. The compiler assumes alphanumeric usage for the conditional variable.
0121	1	**SIZE OF RECORD record-name UNEQUAL TO THAT OF OTHER RECORDS IN FILE SPECIFIED WITH MODE F. LARGER SIZE ASSUMED TO BE THE RECORD SIZE OF FILE.** A file defined with fixed-length record format does not have data records of the same length. The compiler assumes the largest data record length for calculation of record length for the file.
0122	1	**SIZE OF LABEL RECORD data-name NOT 80 CHARACTERS. 80 ASSUMED.** A standard user label record description has a length of other than 80 characters. A length of 80 characters is assumed.
0123	2	**SYNC CLAUSE SPECIFIED FOR data-name REQUIRES DIFFERENT SYNCHRONIZATION THAN OBJECT OF REDEFINES. SYNC CLAUSE IGNORED.** Self-explanatory The SYNC clause is ignored.
0124	1	**BLOCK CONTAINS INSUFFICIENT CHARACTERS TO HOLD ONE RECORD FOR FILE file-name. BLOCK SET TO CONTAIN ONE RECORD.** The value in the BLOCK CONTAINS integer CHARACTERS clause is less than that needed to contain the largest record plus control bytes. The compiler assumes BLOCK CONTAINS 1 RECORD.
0127	1	**RECORD CONTAINS SPECIFICATION NOT EQUAL TO SIZE OF LARGEST RECORD. LARGEST RECORD SIZE USED.** The RECORD CONTAINS clause does not specify the length of the largest data record. The compiler assumes that the length of the largest data record is specified in the RECORD CONTAINS clause.

Message Number	Severity Code	Message/Explanation/Action
0129	2	**REDEFINES CLAUSE NOT PERMITTED FOR LEVEL 01 ENTRY IN FILE OR COMMUNICATION SECTION. CLAUSE IGNORED.** A level 01 entry with a REDEFINES clause was encountered in the file or communication section. The compiler assumes the REDEFINES clause does not exist.
0130	2	**SUBJECT OF REDEFINES, data-name, NOT IN SAME SECTION AS OBJECT OF REDEFINES. REDEFINES CLAUSE IGNORED.** The subject of a REDEFINES clause is not in same section as entry with REDEFINES. The compiler assumes the REDEFINES clause does not exist.
0131	2	**OBJECT OF REDEFINES, data-name, WITHIN RANGE OF OCCURS. REDEFINES CLAUSE IGNORED.** The object of a REDEFINES clause has or is subordinate to an OCCURS clause. The compiler assumes the REDEFINES clause does not exist.
0132	2	**REDEFINES OBJECT, data-name, AND SUBJECT DATA NAME DO NOT HAVE IDENTICAL LEVEL NUMBERS. REDEFINES CLAUSE IGNORED.** The object and subjects of the REDEFINES clause do not have the same level numbers. The compiler assumes the REDEFINES clause does not exist.
0133	3	**INDEX NAME data-name EXCEEDS LIMIT. PREVIOUS INDEX NAMES ASSIGNMENT INVALIDATED.** The current compiler limit of index names is 255. This entry is the 256 specified index name. The compiler starts index name storage assignment over and reassigns the storage to the index names being processed.
0134	1	**NO LENGTH INDICATED IN ELEMENTARY ITEM data-name. LENGTH OF 1 ASSUMED.** An elementary item, determined from level number structure, with no length specified or assumed, was encountered. The compiler assumes a length of 1, signed was specified.
0136	1	**OBJECT OF RENAMES data-name HAS ILLEGAL LEVEL NUMBER. LAST ELEMENTARY ITEM ASSUMED AS OBJECT.** The object of the RENAMES clause has illegal level number. The compiler assumes the last elementary item as specified object of the RENAMES clause.

Message Number	Severity Code	Message/Explanation/Action
0137	2	**OBJECT OF REDEFINES IN data-name ENTRY HAS ILLEGAL LEVEL NUMBER. REDEFINES CLAUSE IGNORED.** The object of the REDEFINES clause is not a legal level for redefinition. The compiler assumes the REDEFINES clause does not exist.
0138	1	**USE DEBUGGING SECTIONS NOT GROUPED TOGETHER AT BEGINNING OF DECLARATIVES. PROCESSED AS IF IN CORRECT SEQUENCE.** All debugging sections must appear together immediately following the DECLARATIVES header. The debugging sections are processed as though they had appeared at the beginning of the DECLARATIVES.
0139	1	**SEGMENT NUMBER INCORRECT OR OUT OF SEQUENCE. SEGMENT NUMBER 0 ASSUMED.** The value of segment number does not fall within range of 0 to 99. The segment number is set to 0.
0140	2	**NO EXIT PROGRAM STATEMENT IN PROCEDURE DIVISION SPECIFIED WITH USING PHRASE. NO CORRECTIVE ACTION TAKEN.** No return mechanism to the calling program is provided. No corrective action is taken.
0142	2	**NO PROCEDURE DIVISION USING PHRASE OR EXIT PROGRAM STATEMENT ASSOCIATED WITH LINKAGE SECTION. NO CORRECTIVE ACTION TAKEN.** No mechanism provided for passing of arguments or exit to calling program. No corrective action is taken.
0143	2	**UNPAIRED ELSE ENCOUNTERED IN IF STATEMENT. IF STATEMENT TERMINATED AT THIS POINT.** ELSE encountered in IF statement with no preceding IF verb to match it with. The conditional statement is terminated at this point.
0144	0	**STOP RUN STATEMENT NOT ENCOUNTERED. NO CORRECTIVE ACTION TAKEN.** The STOP RUN statement was not encountered. No corrective action is taken.

Message Number	Severity Code	Message/Explanation/Action
0145	2	**EXIT STATEMENT NOT THE ONLY STATEMENT IN PARAGRAPH. NO CORRECTIVE ACTION TAKEN.** The EXIT statement must be the only statement in a paragraph. No corrective action is taken.
0146	1	**THE BEFORE OPTION OF THE USE STATEMENT IS NOT APPLICABLE. THE AFTER OPTION IS ASSUMED.** The BEFORE option is not applicable to the operating system. The AFTER option is assumed.
0147	1	**LENGTH OF PROGRAM NAME IN CALL STATEMENT EXCEEDS LIMIT. EXCESS CHARACTERS TRUNCATED.** Program name exceeds 8 or 80 characters in length. The program name in CALL statement truncated to 8 or 80 characters.
0148	2	**REFERENCE TO name CANNOT BE RESOLVED. STATEMENT DELETED.** A definition of the listed name has not been encountered. The statement containing the reference is deleted.
0149	2	**QUALIFIED REFERENCE TO name CANNOT BE RESOLVED. STATEMENT DELETED.** A definition of the listed name has not been encountered under the specified qualifiers. The statement containing the reference is deleted.
0150	1	**REFERENCE TO PROCEDURE name AMBIGUOUS. DEFINITION AT LINE number USED.** A definition of the listed paragraph name has not been encountered within the section from which the reference is made, while multiple definitions exist outside the section of reference. The reference is resolved by the paragraph name at the listed line number.
0151	2	**REFERENCE TO name OF name CANNOT BE RESOLVED DUE TO IMPROPER QUALIFIER AT LINE number. STATEMENT DELETED.** The qualifier of a procedure reference is not a section name; or is found in the data division; or the qualifier of a data reference is found in the procedure division. The statement containing the reference is deleted.

Message Number	Severity Code	Message/Explanation/Action
0152	1	**REFERENCE TO name AMBIGUOUS DUE TO DEFINITION AT LINE number. DEFINITION AT LINE number USED.** Duplicate definition of the listed unqualified name has been encountered in the same division. The definition at listed line number is used.
0153	1	**IMPROPER DEFINITION OF name AT LINE number IMPLIED BY MANNER OF REFERENCE. STATEMENT DELETED IF REFERENCE NOT RESOLVED.** A duplicate definition of the listed unqualified name has been found in another division. If the reference cannot be resolved within the COBOL division corresponding to the reference type, the statement is deleted.
0154	1	**name NOT UNIQUE. DUPLICATE DEFINITION FOUND AT LINE number. STATEMENT DELETED IF REFERENCE NOT RESOLVED.** A duplicate definition has been found for the qualifier. If the reference cannot be resolved within the COBOL division corresponding to the reference type, the statement is deleted.
0155	0	**IMPERATIVE STATEMENT NOT TERMINATED BY PERIOD AT END OF PARAGRAPH. PERIOD ASSUMED.** Last sentence of a paragraph was not terminated by a period. A period is required for correct COBOL syntax, even though it does not affect execution of imperative statements. A period is assumed after the last statement in the paragraph.
0156	2	**BOTH CD INITIAL AND PROCEDURE DIVISION USING SPECIFIED. USING PHRASE IGNORED.** If the USING phrase is specified in the PROCEDURE DIVISION header, the INITIAL clause must not be present in any CD entry. The USING phrase is deleted.
0157	0	**verb STATEMENT OPERAND name IMPROPERLY INDEXED. NO ACTION** TAKEN. An index name used to address a table element is not associated with the table but is associated with another table that has the same element size. No action. Precautionary warning.

Message Number	Severity Code	Message/Explanation/Action
0158	0	**verb STATEMENT REFERENCES WORKING-STORAGE ITEM data-name WHICH, IN IMS ENVIRONMENT, SHOULD NOT BE MODIFIED. NO ACTION TAKEN.** Due to the shared nature of programs operating under IMS mode, errors could occur if working-storage items are modified at object time. No action. Precautionary warning.
0159	2	**verb STATEMENT CONTAINS INVALID OPERAND data-name. STATEMENT** DELETED. The specified data item does not satisfy the requirements for the designated verb; for example, an alphabetic operand in an ADD statement. The statement containing the listed operand is deleted.
0160	2	**verb STATEMENT OPERAND data-name IMPROPERLY SUBSCRIPTED. STATEMENT DELETED.** The data item contains too many, too few, or an improper type of subscript. The statement containing the subscript error is deleted.
0161	2	**verb STATEMENT CONTAINS INCONSISTENT OPERAND data-name. STATEMENT DELETED.** The combination of operands in the statement conflict in their usage; for example, moving a numeric item to an alphabetic operand. The statement containing the inconsistent operand is deleted.
0162	1	**verb STATEMENT CONTAINS SIGNED LITERAL literal. SIGN DELETED.** A signed literal has been encountered. The sign of the literal is deleted.
0163	2	**COMPOSITE OF OPERANDS IN verb statement EXCEEDS 18 DIGITS. STATEMENT DELETED.** The superimposition of all operands to the left of the word GIVING exceeds 18 digits. The statement containing the composite error is deleted.
0164	2	**GO TO PRECEDES IMPERATIVE STATEMENTS. IMPERATIVE STATEMENTS** DELETED. A GO TO statement is followed by other imperative statements. The statements between the GO TO and the ELSE, IF, or period are deleted.

Message Number	Severity Code	Message/Explanation/Action
0165	2	**verb STATEMENT OPERAND data-name NOT DEFINED IN LINKAGE SECTION. STATEMENT DELETED.** The referenced data-name has not been defined in the linkage section. The statement containing the listed operand is deleted.
0166	1	**CONDITIONAL STATEMENT NOT TERMINATED BY PERIOD AT END OF PARAGRAPH. PERIOD ASSUMED.** Last sentence of a paragraph containing a conditional statement was not terminated by a period. A period is required to indicate where conditional statement ends. This does affect execution of the program. A period is assumed after the last statement in the paragraph.
0167	3	**ADDITIONAL MEMORY REQUIRED TO PROCESS STATEMENT CONTAINING data-name. OBJECT MODULE NOT PRODUCED.** This statement exceeds the main storage area available to process statements with multiple operands. The statement is deleted. Additional main storage should be assigned to the compiler or the statement must be rewritten as multiple statements.

Message Number	Severity Code	Message/Explanation/Action
0168	2	**verb STATEMENT EXCEEDS LIMIT OF INTERMEDIATE RESULT AREAS. STATEMENT DELETED.** The maximum number of temporary arithmetic data areas has been exceeded. Reduce the complexity of the expression or reduce the number of expressions in the statement.
0169	2	**verb STATEMENT OPERAND name NOT A RECORD OR FILE NAME. STATEMENT DELETED.** The input/output statement does not reference a record name or file name. The statement in error is deleted.
0170	2	**SENTENCE PRODUCES EXCESSIVE OBJECT CODE. NO CORRECTIVE ACTION TAKEN.** The maximum size of the object code produced for one sentence may not exceed 2048 bytes. Incorrect branching may occur during object program execution. Reduce the sentence size by rewriting it as several sentences or paragraphs.
0171	2	**NEXT SENTENCE PHRASE NOT FOLLOWED BY A PERIOD OR WORD ELSE OR WHEN. PHRASE DELETED.** NEXT SENTENCE must be followed by ELSE, period, or WHEN. The NEXT SENTENCE phrase is ignored.
0172	0	**PERFORM STATEMENT IN DECLARATIVES REFERENCES A NON-DECLARATIVE PROCEDURE. NO ACTION TAKEN.** A PERFORM within the declarative section referenced a procedure outside of the declarative section. No action. Precautionary warning.
0173	2	**verb STATEMENT OPERAND name REFERS TO FILE RECORD AREA. STATEMENT DELETED.** Both operands in the statement refer to the same storage area. The statement is deleted.
0174	2	**verb STATEMENT RECORD-NAME name NOT DEFINED IN FILE SECTION. STATEMENT DELETED.** The listed operand is not defined in the file section. The statement is deleted.

Message Number	Severity Code	Message/Explanation/Action
0175	0	**COMPARISON FOR EQUALITY INVOLVING A FLOATING-POINT OPERAND MAY NOT YIELD EXPECTED RESULTS. PRECAUTIONARY WARNING.** A floating-point operand appears in an equal- or not-equal-relation condition. Because of the limited precision in a floating-point number, the values of the two operands may not compare as equal or not equal. No action. Precautionary warning.
0176	2	**DESCRIPTIONS OF OPERANDS IN DIVIDE STATEMENT COULD PRODUCE ONLY ZERO RESULT. STATEMENT DELETED.** The description of the operands in a DIVIDE statement is structured so that only zeros could result for the quotient in the specified receiver. The DIVIDE statement is deleted.
0177	2	**verb STATEMENT CONFLICTS WITH SEGMENTATION RULES. STATEMENT DELETED.** A branching verb is invalidly specified according to the rules of segmentation, or an ALTER statement refers to a paragraph that does not begin with a GO TO. The statement in error is deleted.
0178	2	**verb STATEMENT INCOMPLETE OR CONTAINS INVALID OPERAND OR OPTION. STATEMENT DELETED.** An operand conflicts with a specified option or with another operand, or an option that must be specified for a given statement was not encountered. For example, a WRITE statement to a mass storage device must contain an INVALID KEY clause. The statement is deleted.
0179	2	**INTERNAL LABEL TABLE OVERFLOW. PROCESSING INCOMPLETE.** Either a sentence requires more than 256 internal labels or more than 24 internal labels are active. Requirements for internal labels may be lowered by reducing the number of statements in a sentence.
0180	2	**CLASS OF LITERAL CONFLICTS WITH CLASS OF data-name. MOVE OPERATION DELETED.** A nonnumeric literal containing numeric characters is being moved to an alphabetic item, or a nonnumeric literal containing nonnumeric characters is being moved to a numeric item. The MOVE operation is deleted.

Message Number	Severity Code	Message/Explanation/Action
0181	0	**element-1 TRUNCATED AND MOVED TO element-2.** The item being moved contains a greater number of character positions than the receiver, or, when decimal-point aligned, contains a greater number of digit positions than the receiver. The data name or intermediate result is truncated and moved.
0182	2	**COMPLETE TRUNCATION OF SIGNIFICANT DIGITS OF element-1. ZEROS MOVED TO element-2.** Decimal point alignment is such that no portion of the item being moved can be contained in the receiving operand. Zeros moved to the receiving field or intermediate result.
0183	0	**REDUNDANT ROUND ON data-name. ROUNDING IGNORED.** The numeric description of the arithmetic result is such that no excess digit positions are available for rounding into the listed operand. The round operation is deleted.
0184	0	**REDUNDANT SIZE ERROR ON data-name.** The numeric description of the arithmetic result is such that its value could never exceed the largest value that can be contained in the listed operand. No action taken.
0185	1	**literal IN DISPLAY OR STOP STATEMENT NOT AN UNSIGNED INTEGER. NON-INTEGER TRUNCATED OR SIGN REMOVED.** A literal operand in a DISPLAY or STOP statement must be an unsigned integer. The literal is made unsigned or truncated to an integer.
0186	1	**VALUE OF INTEGER TIMES IN PERFORM STATEMENT EXCEEDS LIMIT. VALUE SET TO MAXIMUM LIMIT.** The value of integer TIMES IN PERFORM statement exceeds maximum limit of 32,767. The value is set to 32,767.
0187	1	**VALUE OF INTEGER IN WRITE ADVANCING STATEMENT EXCEEDS LIMIT. VALUE SET TO 1.** The integer specified in the WRITE ADVANCING statement exceeds limit of 255. The value of integer is set to 1.

Message Number	Severity Code	Message/Explanation/Action
0188	2	**FILE ON LINE number HAS NO ASSOCIATED verb STATEMENT WITHIN PROGRAM. NO CORRECTIVE ACTION TAKEN.** An OPEN or CLOSE statement has not been specified for the file or the OPEN statement is inconsistent with the activity associated with the file. Results during execution are unpredictable.
0189	2	**verb STATEMENT NOT PERMITTED IN USE LABEL PROCEDURE. STATEMENT DELETED.** The CALL, CANCEL, and all I/O statements except ACCEPT (from software devices) and DISPLAY are not allowed within a USE LABEL PROCEDURE. The statement is deleted.
0190	2	**ADDITIONAL MEMORY REQUIRED TO PRODUCE OBJECT PROGRAM LISTING. LISTING NOT PRODUCED.** The main storage assigned for the compiler is insufficient to generate the object program listing. The object module, however, is produced. To obtain the object program listing, a recompilation with more main storage assignment is required.
0191	3	**ADDITIONAL MEMORY REQUIRED TO PRODUCE OBJECT PROGRAM. OBJECT MODULE NOT PRODUCED.** The main storage assigned to the compiler is insufficient to generate the object module. A recompilation with more main storage assignment is necessary.
0192	0	**TRUNCATION MAY OCCUR WHEN FLOATING-POINT ITEM element-1 IS MOVED TO element-2. MOVE PERFORMED.** A floating-point item is being moved to a receiver which is not floating-point, and there may be a loss of significance. The move is performed.
0193	2	**ON STATEMENT CONTAINS INVALID INTEGER integer. STATEMENT DELETED.** The literal in an ON statement is negative, noninteger, or its value is greater than the allowable maximum (2147483648). The ON statement is deleted.
0194	2	**ON STATEMENT CONTAINS INCONSISTENT INTEGER SPECIFICATIONS. STATEMENT DELETED.** The values of the literals in the statement are inconsistent. The value of the literal in the ON phrase plus the value of the literal in the AND EVERY phrase, if present, is greater than the value of the literal in the UNTIL phrase. The ON statement is deleted.

Message Number	Severity Code	Message/Explanation/Action
0195	2	**verb STATEMENT REFERENCES WORKING-STORAGE ITEM data-name WHICH, IN A RE-ENTRANT IMS ENVIRONMENT, MUST NOT BE MODIFIED. STATEMENT DELETED.** Due to the shared nature of programs operating under IMS reentrant mode, working-storage items cannot be modified at object time. The statement is deleted.
0196	1	**CURRENCY SIGN SYMBOL symbol INVALID. "$" USED.** A character that may not be used as a currency sign was specified in the currency sign clause of SPECIAL-NAMES section. The symbol $ is used for the currency sign.
0197	1	**MORE THAN ONE CHARACTER SPECIFIED FOR element LITERAL. FIRST CHARACTER USED.** A currency sign literal or the second literal of a SOURCE-ALPHABET clause has more than one character. The first character of the string is used.
0198	2	**INCORRECT DIVISION OR SECTION HEADER IN CONTROL DIVISION. PROCESSING CONTINUES WITH NEXT VALID CLAUSE.** The CONTROL DIVISION or ALPHABET SECTION statement is specified incorrectly. Processing resumes at the SOURCE-ALPHABET or MESSAGES statement.
0199	2	**INVALID LITERAL literal IN SOURCE ALPHABET CLAUSE. LITERAL IGNORED.** The second literal of the SOURCE-ALPHABET clause is less than the first literal, or the literal is greater than 255. The literal is ignored.
0200	2	**MULTIPLE ERRORS FOUND IN element CLAUSE. PROCESSING CONTINUES WITH NEXT VALID CLAUSE.** Another error has been found during an error recovery processing of the first error. All words are ignored until a valid clause is found.
0201	1	**EXTRANEOUS PERIOD SPECIFIED IN element CLAUSE. PERIOD IGNORED.** A period precedes an expected element. The period is ignored.

Message Number	Severity Code	Message/Explanation/Action
0202	2	**A PERIOD TERMINATES AN INCOMPLETE element CLAUSE. CLAUSE IGNORED.** A period is found instead of the expected element, and the expected element is not found immediately following the period. All words are ignored until a valid clause is found.
0203	2	**element CLAUSE SPECIFIED OUT OF SEQUENCE. CLAUSE IGNORED.** The clause is not in the proper paragraph or section or out of sequence within the paragraph, or has already been specified. The clause is ignored.
0204	1	**CRITICAL RESERVED WORD MISSING WHERE xxx APPEARS IN element CLAUSE. CLAUSE IGNORED.** The syntax analysis on this clause cannot be continued because of a missing reserved word critical to the syntax of a clause. The clause is ignored.
0205	1	**RESERVED WORD MISSING WHERE xxx APPEARS IN element CLAUSE. MISSING WORD ASSUMED PRESENT.** The missing reserved word is required in the format but not critical to syntax analysis. Processing continues as if the reserved word were specified.
0206	1	**TERMINATING PERIOD MISSING FOR element CLAUSE. PERIOD ASSUMED.** The clause is not terminated by a period. A terminating period is assumed.
0207	1	**element CLAUSE SHOULD NOT BEGIN IN AREA B. AREA A ASSUMED.** The first word of this clause starts in area B. The clause is assumed to have started in area A.
0208	1	**element CLAUSE SHOULD NOT BEGIN IN AREA A. AREA B ASSUMED.** The first word of this clause starts in area A. The clause is assumed to begin in area B.
0209	2	**element CLAUSE MISSING OR OUT OF SEQUENCE. OUT OF SEQUENCE CLAUSE IGNORED.** The processing of a paragraph, section, or division is completed but a mandatory clause was not encountered. If the missing clause is encountered later, it will be ignored.

Message Number	Severity Code	Message/Explanation/Action
0210	1	**RESERVED WORD EXPECTED WHERE xxx APPEARS IN element CLAUSE. RESERVED WORD ASSUMED MISSPELLED.** A user-defined word was encountered where a noncritical reserved word was expected. The reserved word is assumed to be misspelled.
0211	2	**LITERAL EXPECTED WHERE xxx APPEARS IN element CLAUSE. CLAUSE IGNORED.** The missing literal is critical to syntax analysis for the clause. All words are ignored until a valid clause is encountered.
0212	2	**USER-DEFINED WORD EXPECTED WHERE xxx APPEARS IN element CLAUSE. CLAUSE IGNORED.** The expected user-defined word was not found. The clause is ignored.
0213	1	**RESERVED WORD xxx USED AS USER-DEFINED WORD IN element CLAUSE. NO CORRECTIVE ACTION TAKEN.** A reserved word was found where a user-defined word was expected. No corrective action taken.
0215	1	**element CLAUSE SPECIFIED MORE THAN ONCE. CLAUSE IGNORED.** The clause has already been specified in the program. The clause is ignored.
0216	1	**INVALID DEVICE TYPE FOR SPECIFIED ORGANIZATION. DISK ASSUMED.** The device type specified for the file conflicts with the file organization. The disk device is assumed.
0217	1	**DUPLICATE VALUE OF CLAUSES SPECIFIED IN FD ENTRY. DUPLICATE CLAUSE IGNORED.** The file-id or password appears more than once in the VALUE OF clause. The duplicate entry is ignored.
0218	1	**TERMINATING PERIOD FOR A SEQUENCE OF CLAUSES MISSING. PERIOD ASSUMED.** The expected terminating period was not found. If a COPY statement is the prior clause, a period is missing before the word COPY. A terminating period is assumed.

Message Number	Severity Code	Message/Explanation/Action
0220	1	**name MONITORED BY MORE THAN ONE USE FOR DEBUGGING PROCEDURE. FIRST MONITORING PROCEDURE USED.** The indicated name is monitored by more than one USE FOR DEBUGGING statement. The first monitoring procedure encountered is used.
0221	2	**USE FOR DEBUGGING PROCEDURE-NAME name MONITORED AT LINE number. STATEMENT IGNORED.** Procedure names defined within debugging sections must not appear in a USE FOR DEBUGGING statement. The statement is ignored.
0222	2	**ALL PROCEDURES PHRASE SPECIFIED IN USE FOR DEBUGGING AND PROCEDURE-NAME name ALSO MONITORED AT LINE number. STATEMENT FOR PROCEDURE-NAME IGNORED.** When the ALL PROCEDURES phrase is specified, individual procedure names must not appear in USE FOR DEBUGGING statements. The statement referencing the procedure name is ignored.
0223	2	**FIRST ENTRY IN xxx section NOT type ENTRY. A DUMMY type ENTRY GENERATED.** The first entry in the file section or communication section was not an FD or a CD entry. The compiler generates a dummy level indicator entry.
0224	2	**FIRST ENTRY SUBORDINATE TO type entry NOT A type entry. A DUMMY ENTRY GENERATED.** The first entry subordinate to an FD entry was not a level 01 entry. The compiler generates a dummy level 01 entry.
0225	1	**INTEGER-2 NOT GREATER THAN INTEGER-1 IN OCCURS CLAUSE. INTEGER-2 IGNORED.** Integer-2 must be greater than integer-1 in the OCCURS clause. The greater value is used as maximum number of occurrences.
0226	1	**FOOTING INTEGER GREATER THAN LINAGE INTEGER. SAME VALUE ASSUMED.** The value of integer for FOOTING area must be less than or equal to the value of integer for the LINAGE clause. The FOOTING integer is assumed to have the same value as LINAGE.

Message Number	Severity Code	Message/Explanation/Action
0227	2	**element CLAUSE NOT PERMITTED IN IMS/90 ENVIRONMENT. CLAUSE IGNORED.** The clause is not allowed when IMSCOD=YES is specified. The clause is ignored.
0228	2	**data-name NOT SUBORDINATE TO ITEM WITH OCCURS DEPENDING ON CLAUSE. RESULTS ARE UNPREDICTABLE.** A data entry that has an OCCURS DEPENDING ON clause may only be followed by data entries subordinate to it. No corrective action is taken.
0229	2	**MULTIPLE OCCURS DEPENDING ON CLAUSES NOT ALLOWED IN HIERARCHY. RESULTS ARE UNPREDICTABLE.** Only one OCCURS DEPENDING ON clause is allowed in a hierarchy. No corrective action is taken.
0230	2	**LEVEL 77 NOT ALLOWED IN FILE OR COMMUNICATION SECTION. LEVEL 01 ASSUMED.** Self-explanatory The level number is changed to 01.
0231	1	**INTEGER-2 NOT GREATER THAN INTEGER-1 IN RECORD CONTAINS CLAUSE. INTEGER-2 IGNORED.** Integer-2 must be greater than integer-1 in the RECORD CONTAINS clause. The greater value is used.
0232	1	**xxx NOT VALID CHANNEL NUMBER IN SYSCHAN CLAUSE. SYSCHAN-1 ASSUMED.** The specified channel number is not valid for the associated operating system. Channel 1 is assumed.
0233	1	**STANDARD-0 OR STANDARD-1 NOT VALID WHEN VARIABLE RECORD FORMAT FOR THE FILE INDICATED. CODE-SET CLAUSE IGNORED.** Only fixed record format is permitted for a tape file specified with STANDARD-0 or STANDARD-1. The CODE-SET clause is ignored.

Message Number	Severity Code	Message/Explanation/Action
0234	0	**INTO PHRASE USED IN verb STATEMENT FOR name FILE WITH VARIABLE RECORD FORMAT. INTO OPERATION PERFORMED.** The INTO phrase is used with a file containing logical records of variable sizes. The INTO phrase is performed.
0235	0	**COMPARISON OF ALPHANUMERIC LITERAL TO NUMERIC EMBEDDED SIGN ITEM data-name MAY NOT PRODUCE EXPECTED RESULTS.** ANSI COBOL rules require that the byte containing the embedded (overpunch) sign participate in the comparison but that the sign itself not participate. The required manipulation of the overpunch sign byte makes it unlikely that the comparison will work as intended. The comparison is performed.
0236	1	**INITIAL CLAUSE APPEARS IN MORE THAN ONCE CD ENTRY. CLAUSE IGNORED.** Only one input CD entry may contain the INITIAL clause. The clause is ignored.
0237	1	**SIZE OF INPUT CD RECORD data-name NOT EQUAL TO 87 CHARACTERS. RECORD SIZE SET TO 87.** The size of an input CD area must be exactly 87 characters. The size is set to 87 characters.
0238	1	**element NOT SUPPORTED. ELEMENT CHANGED TO xxx.** The indicated logical device name is not supported in this system. Logical device name is changed as indicated in the message.
0239	2	**NUMBER OF INDEX-NAMES SPECIFIED IN DESTINATION TABLE EXCEEDS LIMIT. EXCESS IGNORED.** The number of index names specified for one destination table may not exceed 10. The excess index names are ignored.
0240	1	**CODE-SET CLAUSE SPECIFIED FOR NON-TAPE FILE. CLAUSE IGNORED.** The CODE-SET clause may be specified only for tape files. The native character code-set is assumed.

Message Number	Severity Code	Message/Explanation/Action
0241	0	**MORE THAN ONE CLASS-NAME CLAUSE WITHOUT VALUE PHRASE SPECIFIED. DEFAULT VALUE USED.** Only one CLASS-NAME clause may be specified without the VALUE phrase. The character set specified in the SOURCE-ALPHABET clause is used.
0242	1	**FORMAT ERROR ON BASIS CARD CONTAINING char-string. FIRST 8 CHARACTERS OF SPECIFIED NAME USED. IF NAME MISSING "SOURCE" USED AS NAME.** The BASIS card has format error. The first eight characters of the specified name are used. If the name is not specified, a default name "SOURCE" is used.

Message Number	Severity Code	Message/Explanation/Action
0243	1	**KEY SPECIFIED IN verb STATEMENT EXCEEDS LIMIT. KEY TRUNCATED TO 10 CHARACTERS.** Password for ENABLE or DISABLE statement may not exceed 10 characters. The KEY is truncated to 10 characters.
0244	2	**LINAGE-COUNTER MODIFIED BY verb STATEMENT. STATEMENT IGNORED.** LINAGE-COUNTER may only be referenced but not modified. The statement is ignored.
0245	1	**INTEGER SPECIFIED IN APPLY CYLINDER-OVERFLOW AREA CLAUSE EQUAL TO OR GREATER THAN 100. TWENTY PERCENT IS USED.** A percent of 100 or greater was specified for cylinder overflow area. 20 percent of each cylinder is used for overflow area.
0246	1	**INTEGER IN APPLY CYLINDER-INDEX CLAUSE EXCEEDS THE LIMIT OF 32,767. THE LIMIT IS USED.** The value of integer in the APPLY CYLINDER-INDEX clause may not exceed the limit of 32,767. Value is assumed to be 32,767.
0247	1	**ALPHABET-NAME REFERENCED IN CODE-SET CLAUSE DEFINED AS LITERAL. CODE-SET CLAUSE IGNORED.** The alphabet-name referenced in a CODE-SET clause specifies the character code set of the file and must be defined by one of the reserved word phrases in the SPECIAL-NAMES paragraph. The native character code set is assumed.
0248	2	**INVALID PSEUDO-TEXT IN COPY STATEMENT. PSEUDO-TEXT IGNORED.** The pseudo-text preceding the reserved word BY is null or contains a debugging line or comments only. The pseudo-text is not processed.
0249	2	**DATA-NAME REFERENCED IN KEY PHRASE OF READ OR START STATEMENT NOT SPECIFIED IN SELECT CLAUSE. STATEMENT DELETED.** The data-name referenced in the KEY phrase of a random READ or in a START statement is not specified in the SELECT clause for that file. The READ or START function is deleted. Other parts of the statement, such as INTO and FROM moves, and error checking may be present. Execution of the statement yields unpredictable results.

Message Number	Severity Code	Message/Explanation/Action
0250	0	**LANGUAGE ELEMENT xxx yyy EXCEEDS SPECIFIED FIPS LEVEL. ELEMENT BELONGS TO LEVEL nnn. NO CORRECTIVE ACTION TAKEN.** The language element used in the program exceeds the specified FIPS processing level. The language element is accepted.
0251	0	**ALTERNATE KEY SHARES LEFTMOST POSITION WITH OTHER KEYS IN SAME FILE.** Although ANSI COBOL rules prohibit keys having the same leftmost position, OS/3 COBOL permits it for compatibility with other OS/3 processors. The ALTERNATE KEY clause is accepted.
0252	2	**verb STATEMENT NOT ALLOWED IN DECLARATIVES. STATEMENT DELETED.** Input/output verbs and sort or merge statements are not allowed in DECLARATIVES. The statement is ignored.
0253	1	**SIZE OF INDEX-AREA IN APPLY CLAUSE NOT A MULTIPLE OF 256 OR EXCEEDS LIMIT OF 32,512. A SIZE OF nnnn CHARACTERS IS USED.** Integer-6 in APPLY INDEX-AREA clause is not a multiple of 256 or exceeds the limit. The value indicated in the message text is used.
0254	0	**EXTRANEOUS PERIOD SPECIFIED PRIOR TO element CLAUSE. PERIOD** IGNORED. A period should not appear before the clause. The period is ignored.
12**	3	**COMPILER ERROR code.** A compiler error occurred when processing the source program line indicated by the line number. Contact your local Sperry Univac representative. Correcting any source program error on or before the indicated line may avoid the compiler error.

Appendix D. Federal Information Processing Standard Flagging Facility

D.1. FIPS PUB 21-1 COBOL LEVELS

The *Federal Information Processing Standard Publication 21—1* (*FIPS PUB 21—1*) identifies the Federal Standard COBOL by four levels: low, low-intermediate, high-intermediate, and high. The Federal Standard COBOL is a subset of *American National Standard COBOL, X3.23—1974*. Table D-1 identifies the COBOL modules that comprise each of the four federal levels.

Table D—1. Federal Standard COBOL Levels

Module	Level			
	Low	Low-Intermediate	High-Intermediate	High
Nucleus	1	1	2	2
Table handling	1	1	2	2
Sequential I-O	1	1	2	2
Relative I-O	-	1	2	2
Indexed I-O	-	-	-	2
Sort-merge	-	-	1	2
Report writer*	-	-	-	-
Segmentation	-	1	1	2
Library	-	1	1	2
Debug	-	1	2	2
Inter-program communications	-	1	2	2
Communication	-	-	2	2

*The report writer is not required for any Federal Standard level.

D.2. FLAGGING OPTIONS

The compiler provides five options for monitoring a source program at compile time. Four options comprise the four levels of Federal Standard COBOL (Table D-1); the fifth option is concerned with OS/3 extensions to the language:

Option	Function
5	Allows all language elements supported by the compiler, both standard and extended, without flags
4	Flags all language elements beyond the high level of the Federal Standard COBOL, indicating those that are extensions
3	Flags all language elements beyond the high-intermediate Federal level, indicating those that are extensions and those that belong to the high Federal Standard COBOL level
2	Flags all language elements beyond low-intermediate level, indicating those that are extensions and those that belong to the high-intermediate or high level of the Federal Standard COBOL
1	Flags all language elements beyond the low Federal level, indicating those that are extensions and those that belong to the higher levels

NOTE:

The language elements pertaining to the file processing facilities that exceed the user-specified FIPS level are flagged on the ORGANIZATION clause in the file control entry, but not on references to the file.

The user may specify any of the five levels as a SYSGEN option or a compile-time parameter option. If none of the FIPS options is specified, compiler default option 5 is used.

The flagged language elements, if syntactically correct, are retained for compilation.

Appendix E. Object Program Processing Considerations

E.1. INTERMEDIATE RESULTS IN ARITHMETIC OPERATIONS

The compiler reduces arithmetic statements to a series of one or more simple arithmetic operations, each producing an intermediate result. The intermediate result of an arithmetic operation may be used as an operand in subsequent arithmetic operations or may be used as the final result of a statement.

Example:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

where:

Operand 1	Operator	Operand 2	Intermediate Result (ir)
F	**	G	ir1
B	*	C	ir2
D	/	E	ir3
A	+	ir2	ir4
ir4	-	ir3	ir5
ir5	+	ir1	Y

The compiler provides a description for each intermediate result that is appropriate for use in the operation or series of operations for which it is intended. The description can be expressed as a numeric PICTURE; however, an intermediate result may contain as many as 30 digits. If the number of digits in an intermediate result is computed to exceed 30, the result is truncated to use the 30 least significant digits.

A description of an intermediate result requires only a digit size and a point location. In terms of the PICTURE clause, the digit size is equivalent to the number of 9's in the PICTURE character-string and the point location is the number of digit places that the assumed decimal point is displaced from the least significant digit.

Example:

PICTURE	Digit Size	Point Location
99V9	3	1
PP999	3	5
99PP	2	-2

E.1.1. Floating-Point Operands

If at least one floating-point (COMP-1 or COMP-2), floating-point display, or floating-point literal operand is used, the range of intermediate results is $\pm5.4*10^{-79}$ to $\pm7.2*10^{75}$.

NOTE:

The following paragraphs apply only to non-floating-point operands.

E.1.2. ADD and SUBTRACT Statements

The description of the intermediate result area is determined by forming the composite of operands (6.6.2) and appending one additional digit in the most significant position to contain overflow when 10 or fewer operands immediately follow the verb, or two digits for more than 10 operands.

E.1.3. MULTIPLY Statement

The description of the intermediate result has a digit size equal to the sum of the digit sizes of the two operands being multiplied and has a point location equal to the sum of the point locations of the two operands.

E.1.4. DIVIDE Statement

The following abbreviations are used in the discussion of the DIVIDE statement:

DD	Dividend
DR	Divisor
Q	Quotient
pl	Point location
ds	Digit size

The description of the quotient intermediate result has a point location equal to the point location of the composite of receiving operands. If the ROUNDED phrase is specified for any operand, the point location for that operand is increased by 1 to form the composite. The digit size for the description of the quotient intermediate result is computed as follows:

quotient digit size = DDpl–DRpl+Qpl+DDds

The point location and digit size for the description of the remainder intermediate result is computed as follows:

A = 0
B = DDpl–Qpl–DRpl

If B < 0 then
    A = absolute value of B, and
    B = 0

remainder point location = DDpl+A
remainder digit size = DRds+B

NOTE:

The value of A represents the number of zero digits that must be padded on the low-order end of the dividend to produce the desired quotient described by the PICTURE character-string.

The value of B represents the number of zero digits that must be padded on the low-order end of the divisor to produce the desired quotient described by the PICTURE character-string.

E.2. EXPRESSIONS

For arithmetic expressions the following abbreviations are used:

L Length in mappable digits

pl Point location, which is the number of places that the decimal point is displaced from the position it would occupy if the mappable digits were considered an integer. For example, for the PICTURE 99V9, pl = 1, because the decimal point has been displaced one position; for the PICTURE PP999, pl = 5. A negative value in pl indicates trailing P's in the associated PICTURE, e.g., for the PICTURE 99PP, pl = -2.

OP1 First operand

OP2 Second operand

ir Intermediate result

comp Composite of operands

mag Magnitude = L - pl

The maximum value that a variable can assume is $10^{mag}-10^{-pl}-1$

When expressions are evaluated, a composite is formed of all operands except those immediately to the right of the exponentiation operator. The receiving data item, when present, is considered in determining the composite. The following rules apply:

Operator	**Description**
+ –	$pl_{ir} = \max (pl_{OP1}, pl_{OP2})$ $L_{ir} = \max (mag_{OP1}, mag_{OP2}) + pl_{ir} + 1$
*	$pl_{ir} = pl_{OP1} + pl_{OP2}$ $L_{ir} = mag_{OP1} + mag_{OP2} + pl_{ir}$
/	$pl_{ir} = pl_{comp}$ $L_{ir} = pl_{OP2} - pl_{OP1} + L_{OP1} + pl_{ir}$
**	Intermediate result is floating point (E.1.1).

NOTE:

When an expression appears in a COMPUTE statement and the ROUNDED option is specified, one digit is added in the least significant position of the receiver description before the composite is formed.

When application of the preceding rules produces an intermediate result length that is greater than 30, the description must be readjusted. In these cases, $L_{ir} = 30$.

Appendix F. Non-English Language Feature

F.1. FUNCTION

The non-English language feature supported by this compiler involves three aspects of a COBOL program: user-defined words in source programs, compiler listing headings and diagnostics, and object program class test.

- User-defined words in source programs

  Data-names, procedure names, and other user-defined words (except system-related words) may be composed of characters specified via the source program.

  The set of COBOL characters that may be used to form user-defined words may be extended by a user program to include additional characters from the computer character set.

- Compiler listing headings and diagnostics

  A user program may specify that an alternate set of the compiler listing headings and diagnostics be used during compilation.

- Object program class test

  A mnemonic-name may be associated with a user-generated set of computer characters. This name may then be used in a class test to determine if the contents of a data item consist entirely of the specified characters.

F.2. LANGUAGE CONCEPTS

F.2.1. Control Division

The control division identifies the character set from which user-defined words in the source program may be formed. The control division also names the load module containing the alternate text of compiler listing headings and diagnostics.

F.2.2. CLASS-NAME Clause

The CLASS-NAME clause in the SPECIAL-NAMES paragraph provides a means of specifying a mnemonic-name for class test purposes and relating it to a specified set of characters.

F.2.3. Extended Class Condition

The extended class condition determines if the operand consists entirely of the characters specified in the associated CLASS-NAME clause.

F.3. COMPOSITE LANGUAGE FORMAT

The following is the composite language format of the non-English language feature.

The leftmost margin 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.

Format:

```
Margin    Margin
A         B
[CONTROL  DIVISION.
 ALPHABET SECTION.
                    [SOURCE-ALPHABET clause]
                    [MESSAGES clause].]
          .
          .
          .
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
          .
          .
          .
[SPECIAL NAMES.
          .
          .
          .
                    [CLASS-NAME clause]...]
          .
          .
          .
  PROCEDURE DIVISION.
          .
          .
          .
                identifier IS [NOT] CLASS-NAME mnemonic-name
```

F.4. CONTROL DIVISION

Function:

The control division identifies the character set from which user-defined words may be formed in the source program. It also names the load module containing the alternate text of compiler listing headings and diagnostics.

Format:

```
Margin   Margin
A        B
_____________________________________________________________
[CONTROL DIVISION.
 ALPHABET SECTION.
          [SOURCE-ALPHABET CHARACTERS ARE
          [
              literal-1 [{THROUGH} literal-2]
                        [{THRU   }          ]

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

          [MESSAGES ARE alternate-text-module-name].]
```

Rules:

1. The control division follows the reference format rules of a COBOL source program. However, the control division, if present, must be the first division of a COBOL source program followed by the identification division, the environment division, the data division, and the procedure division.

   Within the control division, the format defines the order of presentation in the source program.

2. The control division must begin with the reserved words CONTROL DIVISION followed by a period and a space.

3. The alphabet section must begin with the reserved words ALPHABET SECTION followed by a period and a space.

4. The SOURCE-ALPHABET clause defines the additional characters to extend the standard COBOL character set. These extended characters are used in the user-defined words in the source program.

   Example:

   If the SOURCE-ALPHABET clause is defined as:

   ```
   SOURCE-ALPHABET CHARACTERS ARE 124, 125.
   ```

   Then, the characters # and @ are used to form a data-name or procedure-name, such as:

   ```
   PARA#1Ø
   #@
   AMOUNT
   ```

5. The literals in the SOURCE-ALPHABET clause may be numeric or nonnumeric literals.

   - A numeric literal specifies the decimal ordinal position (1 through 256) within the native computer character set (EBCDIC) of valid source alphabets from which user-defined words may be formed. This ordinal position is always one greater than the binary value of the character. For example, hexadecimal 00 is the first character (ordinal position), and hexadecimal 01 is the second character (ordinal position 2).

   - A nonnumeric literal represents non-English alphabet characters. The bit configuration of each character in the native computer character set specified in the nonnumeric literal is included in the set of bit configurations for valid source alphabets from which user-defined words may be formed.

   - When the THROUGH option is used, the nonnumeric literal must consist of only one character. All bit configurations within the range specified are included in the set of valid source alphabets. The range of bit configurations must indicate an ascending sequence of binary values.

   - The literals may not specify those characters in the standard COBOL character set that denote special meanings in the syntax of the COBOL language.

6. The alternate-text-module-name is a user-defined system-related word. The first five characters are to be used as a unique name of the load module of the non-English listing headings and diagnostic message text.

7. The control division is required only if the non-English language feature is invoked.

8. The COPY statement may appear in the control division just as in any other division.

9. The following user-defined system-related words must conform to the standard COBOL rules for formation of user-defined words:

   alternate-text-module-name
   library-name
   lfdname
   program-name
   text-name
   literal-1 in the CALL and CANCEL statements
   contents of identifier-1 in the CALL and CANCEL statements

F.5. ENVIRONMENT DIVISION

F.5.1. CLASS-NAME Clause

Function:

The CLASS-NAME clause provides a means of specifying a mnemonic-name for class test purposes and relating it to a specified set of characters.

Format:

```
B
12
┌CLASS-NAME IS mnemonic-name
└
      ┌VALUE IS literal-1 ┌{THROUGH} literal-2┐
      └                   └{THRU   }          ┘
      ┌literal-3 ┌{THROUGH} literal-4┐┐...┐┐...
      └          └{THRU   }          ┘┘   ┘┘
```

Rules:

1. The CLASS-NAME clause has no relationship with a CODE-SET clause, the alphabet-name clause relating to a PROGRAM COLLATING SEQUENCE clause, or a COLLATING SEQUENCE phrase of a SORT or MERGE statement.

2. The CLASS-NAME clause defines the complete set of characters used as data. Data fields containing these characters are tested by the extended class condition.

   Example 1:

   The following clause defines 10 characters as the complete set of characters used as data:

   ```
   CLASS-NAME IS ABC
        VALUE IS ="ØØ" THRU ="Ø9".
   ```

   Example 2:

   The following clause defines the first 10 characters of the computer character set and the alphabetic characters A through Z as the complete set of characters used as data:

   ```
   CLASS-NAME IS XYZ
        VALUE IS
          1 THRU 1Ø
          ="C1" THRU ="C9"
          "J" THRU "R"
          "S" THRU "Z".
   ```

3. If the VALUE phrase is omitted, the SOURCE-ALPHABET clause must be specified, and it is assumed that the same set of characters specified in the SOURCE-ALPHABET clause is to be associated with the mnemonic-name.

   Example:

   ```
   .
   .
   .
   SOURCE-ALPHABET CHARACTERS ARE 124, 125.
   .
   .
   .
   CLASS-NAME IS DEF.
   ```

In this example, the VALUE phrase is omitted because the character set used in the CLASS-NAME test is the same character set defined in the SOURCE-ALPHABET clause. Therefore, the entire COBOL character set plus the characters # and @ are tested in the extended class condition.

4. The standard COBOL class condition NUMERIC and ALPHABETIC maintain their standard COBOL definitions and cannot be redefined by the CLASS-NAME clause.

5. More than one CLASS-NAME clause is permitted if the VALUE phrase is specified. Only one CLASS-NAME clause without the VALUE phrase can be used.

F.6. PROCEDURE DIVISION

F.6.1. Extended Class Condition

Function:

The extended class condition determines if the operand consists entirely of the characters specified in the associated CLASS-NAME clause.

Format:

```
identifier IS [NOT] CLASS-NAME mnemonic-name
```

Rules:

1. The mnemonic-name must also be specified in the CLASS-NAME clause in the SPECIAL-NAMES paragraph of the environment division.

2. The extended class condition has no bearing upon the standard COBOL NUMERIC test and ALPHABETIC test. The standard COBOL class condition maintains its definitions of NUMERIC and ALPHABETIC.

F.7. NON-ENGLISH TEXT UTILITY PROGRAM

The non-English text utility program accepts as input the alternate text for the compiler listing headings and diagnostics. It produces a source file containing the alternate text, which is assembled and linked into a load module. The load module is then loaded as the text to be used during compilation.

Appendix G. IMS Action Programs

G.1. GENERAL

COBOL programs to be executed under control of the SPERRY Information Management System (IMS) should be compiled using the IMSCOD=YES parameter for serially-reusable and shared-code action programs, or the IMSCOD=REN parameter for reentrant action programs. (See Appendix A.)

G.2. ACTION PROGRAMS

A COBOL program running under control of IMS is called an action program. Specify the IMSCOD=REN compiler parameter to generate reentrant action programs. To generate shared-code action programs, specify the IMSCOD=YES compiler parameter. To generate serially-reusable action programs, specify the IMSCOD=YES compiler parameter. The following rules and restrictions of COBOL action programs must be observed, when the IMSCOD=YES or IMSCOD=REN parameter is specified.

Rules:

1. The following COBOL verbs, clauses, and sections are illegal in the IMS environment and the compiler deletes them from the program.

   ACCEPT MESSAGE COUNT
   ALTER
   CALL identifier
   CANCEL
   CLOSE
   COMMUNICATION SECTION
   DECLARATIVES
   DELETE
   DISABLE
   ENABLE
   EXHIBIT
   FILE SECTION
   INPUT-OUTPUT SECTION
   MERGE
   OPEN
   READ
   RECEIVE
   RELEASE
   RETURN
   REWRITE
   SEGMENT-LIMIT
   SEND
   SORT
   START
   STOP
   SYSCHAN-n
   SYSCONSOLE
   SYSFORMAT
   SYSIN
   SYSIPT
   SYSLOG
   SYSLST
   SYSOPT
   SYSOUT
   SYSSCOPE
   SYSTEM-SHUTDOWN
   SYSTERMINAL
   SYSWORK
   TRACE
   WRITE

2. The PROCEDURE DIVISION header must contain a USING clause.

3. The IMS action program may invoke IMS functions via the CALL statement with the IMS function name expressed as a nonnumeric literal. For a list of IMS function names, refer to the Action programming in COBOL and BAL user guide, UP-9207 (current version).

   In a shared-code action program (IMSCOD=YES), IMS functions may not be invoked from linked-in subroutines; function calls must be coded in the main program.

   In reentrant action programs (IMSCOD=REN), the IMS functions RETURN or ACTIVATE, and the TIP/30 functions TIPDXC, TIPJUMP, TIPRTN, and TIPXCTL, should not be called from linked-in subroutines. COBOL generates special object code for these function names to deallocate the object program reentrancy control area in the IMS work area. Calls to these functions from linked-in subroutines deallocates the object program reentrancy control area for that linked-in subroutine. Reentrancy control areas allocated before this problem occurs are not affected. Control areas could be allocated after these inactive control areas (see G.4). Because of this, you may run out of space in the work area, which destroys program data areas and, possibly, abnormally terminates the action program.

4. A segment number greater than or equal to 50 is diagnosed and changed to 0.

5. In the SPECIAL-NAMES paragraph, only two implementor-names, SYSCOM and SYSSWCH[-n], may be defined.

6. The following verbs should not have working-storage items as receiving operands:

   ACCEPT
   ADD
   COMPUTE
   DIVIDE
   INSPECT
   MOVE
   MULTIPLY
   PERFORM (varying)
   SEARCH (varying)
   SET
   STRING
   SUBTRACT
   TRANSFORM
   UNSTRING

   If IMSCOD=YES and the compiler detects a statement in a shared-code or serially-reusable action program that modifies working-storage, the compiler:

   - generates the object code for the statement; and

   - issues a precautionary diagnostic message.

   If IMSCOD=REN and the compiler detects a statement in a reentrant action program that modifies working-storage, the compiler deletes the statement and issues a serious diagnostic message.

   The contents of working-storage items in a COBOL action program should not be modified because a COBOL action program is potentially sharable or reentrant. If the contents were modified, the modified contents could have been modified again by a concurrent execution of the same action program. Therefore, the contents of the modified working-storage items at the time immediately before the interrupt could be different than those at the time immediately following the interrupt.

   NOTE:

   Index-names are not working-storage items and may be modified in IMS action programs.

7. In addition to invoking IMS functions with CALL statements, a COBOL action program may call subroutines. If the action program is shared or serially-reusable (compiled with IMSCOD=YES), compile all COBOL subroutines with the IMSCOD=YES parameter. If the subroutines are not written in COBOL, they should follow the conventions of an IMS environment (see Information management systems concepts and facilities, UP-9205 (current version)). If the action program is reentrant (compiled with IMSCOD=REN), compile all subroutines as reentrant, regardless of whether they are written in COBOL.

   All subroutines associated with a COBOL action program must be statically bound with the action program. (The CALLST=YES is assumed by the compiler when IMSCOD=YES or IMSCOD=REN is specified.) In this case, all CALL statements must specify subroutine names with nonnumeric literals. Do not use these subroutine names: ACTIVATE, RETURN, TIPDXC, TIPJUMP, TIPRTN, or TIPXCTL, because the compiler generates special object code for these names. This code deallocates the object program reentrancy control area in the IMS work area for the calling program (see G.4).

   The values of index-names in a reentrant COBOL subroutine are not saved between successive executions of the subroutine. Index names are part of the object program reentrancy control area, and are allocated in the IMS work area on each entry to the subroutine and deallocate it on each exit from the subroutine.

8. For a COBOL shared-code object program to be reentrant at CALL interrupts (IMSCOD=YES), the volatile work area used by the program must be saved and restored by the IMS system. The size of the work area, which varies between programs, is displayed in decimal on the compilation summary listing. The message reads:

   ```
   SHARED CODE VOLATILE DATA AREA=nnn BYTES
   ```

   This size is used in computing the SHRDSIZE parameter in the IMS configurator. Refer to the IMS system support functions user guide/programmer reference, UP-8364 (current version).

9. When you specify IMSCOD=REN, the COBOL compiler generates a reentrant object module. This is achieved by placing object program reentrancy control variables in the high order portion of the IMS work area. The compiler reports the additional work area required for object program reentrancy control in the compilation summary listing with the message:

   ```
   RE-ENTRANCY CONTROL = xxxxxx WORKAREA BYTES
   (NOT INCLUDING PROGRAM DEFINED DATA AREAS)
   ```

   Configure IMS with this work area size plus the size of the program-defined data areas on the WORKSIZE parameter. When the action program calls subroutines, WORKSIZE must be large enough to meet the maximum program data area requirements and the size of all the concurrently active object program reentrancy control areas. If you don't specify a large enough work area, program-defined data areas are destroyed, and the action program may be abnormally terminated.

   For more information, see IMS system support functions user guide, UP–8364 (current version). IMS action programming in COBOL and basic assembly language (BAL), UP–9207 (current version), and G.4 of this manual.

10. Normally, execution time errors result in a CE error message and program termination. In an action program, execution time errors result in a program check interrupt, a snapshot dump with the address of the CE message in register 1, and termination of the action program. If there is insufficient work area available for the COBOL reentrancy control variables (IMSCOD=REN), the action program may terminate with a program check, with the program status word address in the dump pointing to the error message.

11. The compiler does not diagnose the ACCEPT and DISPLAY statements that reference DATE, DAY, TIME, SYSCOM, or SYSSWCH. These statements (ACCEPT and DISPLAY) cannot be used in an IMS action program, especially in a multithread environment. IMS provides date and time information as part of the action program's linkage section data.

G.3. COMPILER PARAMETER SPECIFICATIONS AND IMS CONFIGURATION SPECIFICATIONS

Table G–1 indicates which settings of the IMS configuration parameter are allowed for action programs compiled with the IMSCOD parameter settings. Compile serially-reusable action programs and subroutines with the IMSCOD=YES parameter. Shared-code action programs are not recommended. They are only supported to be compatible with earlier releases, and offer no advantages over reentrant action programs. Compile reentrant action programs with the IMSCOD=REN parameter.

Table G–1. IMS Configuration

	IMS Action Program			IMS Action Subroutine	
IMSCOD Parameter Setting	**Serially Reusable**	**Shared Code**	**Reentrant**	**Serially Reusable**	**Reentrant**
IMSCOD=NO	Allowed[1]	Not allowed	Not allowed	Allowed[1]	Not allowed
IMSCOD=YES	Recommended	Allowed	Not allowed	Recommended	Not allowed
IMSCOD=REN	Allowed[2]	Allowed[2,3]	Recommended	Allowed[2]	Recommended

NOTES:

1. *Because the compiler performs no validation against non-IMS system interfaces, adhere to the IMS environment programming rules (see IMS facilities and concepts, UP-9205 (current version)).*

2. *The program allocates object program reentrancy control areas in the IMS work area, even though the program is not reentrant. Configure work area using rule 9 in G.2.*

3. *Configure a volatile data area even though the compiler does not report the volitile data area size.*

G.4. REENTRANT ACTION PROGRAM WORK AREA USAGE

COBOL reentrant action programs use the high-order portion of the IMS work area for object program reentrancy control variables. A marker at the end of each object program reentrancy control area contains a 6-byte character string, COBL74, and a 2-byte binary count of the number of bytes to the next marker. When an action program uses more than one COBOL reentrant object module, the areas are stacked from back to front, linked by these markers.

A new object program searches the work area from back to front, looking at these markers until it finds a marker that does not contain COBL74. The object program puts the character string COBL74 and the 2-byte count (of the number of bytes to the next marker) in this marker and moves the object program control variables into the area preceding the marker.

Because there is no way for the object program to determine where the program data area ends, the object program control variables may overwrite the program data area. If the marker search hits the beginning of the work area, the object program forces a program check of the action program.

When an object program encounters an EXIT PROGRAM statement, or a CALL statement to ACTIVATE, RETURN, TIPDXC, TIPJUMP, TIPRTN, or TIPXCTL function, the object program zeros out the marker, freeing the control area for future object programs. Figure G–1 shows the use of the IMS work area.

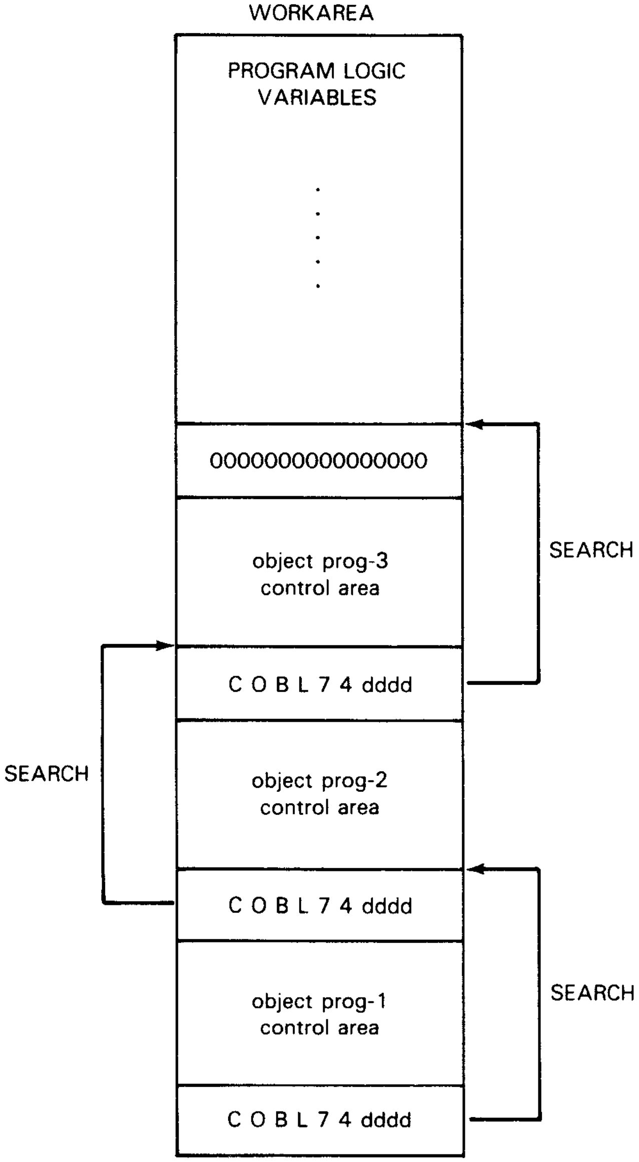


Figure G–1. IMS Work Area Usage

Appendix H. Job Control Stream Requirements

H.1. GENERAL

There are two ways to invoke the COBL74 compiler:

- provide the required job control statements in the job stream (see OS/3 job control user guide, UP-8065 (current version)); or

- use a single job control procedure call statement (jproc call) provided by Sperry Univac.

A jproc call generates all the job control statements needed to execute the COBL74 compiler. By specifying the proper options for the keyword parameters of the jproc call, a desired job control stream is generated. The jproc calls provide the ability to compile, link-edit, and immediately execute this load module (COBL74LG).

NOTE:

The COBL74 compiler requires three disk scratch files, a printer file named PRNTR, and X'E000' bytes of storage.

H.2. PROCEDURE CALL STATEMENT

Function:

This procedure call statement generates the necessary job control statements to execute the COBOL language processor (COBL74). Optionally, it can generate the job control statements that specify the following:

- Input source library

- Output object library

- Copy library

- PARAM control statements that specify the compiler options

Format:

```
// [source-module-name] ⎧COBL74  ⎫ ⎡PRNTR=⎧N                             ⎫⎤
                        ⎨COBL74L ⎬ ⎢      ⎨ ⎛⎧lun⎫[,vol-ser-no]⎞         ⎬⎥
                        ⎩COBL74LG⎭ ⎢      ⎪ ⎜⎨N  ⎬             ⎟         ⎪⎥
                                   ⎣      ⎩ ⎝⎩20 ⎭             ⎠         ⎭⎦

   ⎡,IN=⎧(vol-ser-no,label)⎫⎤ ⎡,LIN=⎧(vol-ser-no,label)⎫⎤
   ⎢    ⎪(RES)             ⎪⎥ ⎢     ⎪(RES,label)       ⎪⎥
   ⎢    ⎨(RES,label)       ⎬⎥ ⎢     ⎨(RUN,label)       ⎬⎥
   ⎢    ⎪(RUN,label)       ⎪⎥ ⎢     ⎪(*, label)        ⎪⎥
   ⎣    ⎩(*,label)         ⎭⎦ ⎣     ⎩(RES,$Y$SRC)      ⎭⎦

   ⎡,LINn=⎧(vol-ser-no,label)⎫⎤   ⎤ ⎡,OBJ=⎧(vol-ser-no,label)⎫⎤
   ⎢      ⎪(RES,label)       ⎪⎥   ⎥ ⎢     ⎪(RES,label)       ⎪⎥
   ⎢      ⎨(RUN,label)       ⎬⎥...⎥ ⎢     ⎨(RUN,label)       ⎬⎥
   ⎢      ⎩(*,label)         ⎭⎥   ⎥ ⎢     ⎪(*,label)         ⎪⎥
   ⎣                          ⎦   ⎦ ⎣     ⎩(RUN,$Y$RUN)      ⎭⎦

   ⎡,SCR1=⎧vol-ser-no⎫⎤ ⎡,SCR2=⎧vol-ser-no⎫⎤ ⎡,SCR3=⎧vol-ser-no⎫⎤
   ⎣      ⎩RES       ⎭⎦ ⎣      ⎩RES       ⎭⎦ ⎣      ⎩RUN       ⎭⎦

   ⎡,ALTLOD=⎧(vol-ser-no,label)⎫⎤[,option=specification]
   ⎢        ⎪(RES,label)       ⎪⎥
   ⎢        ⎪(RUN,label)       ⎪⎥
   ⎢        ⎨(*,label)         ⎬⎥
   ⎢        ⎪(RES,$Y$LOD)      ⎪⎥
   ⎣        ⎩(RES,$Y$RUN)      ⎭⎦
   [ERRFIL=(vol-ser-no,label,module-name)]
```

where:

source-module-name
Specifies the 1- to 6-character source module name; only needed when the IN parameter is used.

COBL74
Specifies compilation of an ANSI 1974 COBOL source program.

COBL74L
Specifies compilation of an ANSI 1974 COBOL source program and link-edit of the object modules.

COBL74LG
Specifies compilation of an ANSI 1974 COBOL source program, link-edit of the object modules, and execution of the load module.

NOTE:

Device assignment sets must be specified prior to the jproc.

PRNTR Keyword Parameter:

```
⎡PRNTR=⎧N                         ⎫⎤
⎢      ⎨⎛⎧lun⎫[,vol-ser-no]⎞      ⎬⎥
⎢      ⎪⎜⎨n  ⎬             ⎟      ⎪⎥
⎣      ⎩⎝⎩20 ⎭             ⎠      ⎭⎦
```

Specifies the logical unit number of the printer. N specifies that the device assignment set for the printer is to be manually inserted in the control stream.

IN Keyword Parameter:

This parameter specifies the input file definition and generates a PARAM IN control statement. The options are:

IN=(vol-ser-no,label)
Specifies the file identifier (label) and the volume serial number (vol-ser-no) where the source input is located.

IN=(RES)
Specifies that the source input is located on the SYSRES device in $Y$SRC.

IN=(RES,label)
This is used if the source input is located on the SYSRES device, but the file identifier (label) is of user-own specification, not $Y$SRC.

IN=(RUN,label)
Specifies that the source input is located on the job's $Y$RUN file, with the file identifier (label) of user-own specification.

IN=(*,label)
Specifies that the source input is located on a catalog file identified by the file identifier (label).

If omitted, the source input is in the form of embedded data cards (/$,source deck,/*).

LIN Keyword Parameter:

LIN=(vol-ser-no,label)
Defines the volume serial number (vol-ser-no) and the file identifier (label) where the copy modules are located. The LFD name is COPY$.

LIN=(RES,label)
Specifies that the copy modules are located on the job's SYSRES device in the file identified by the file identifier (label).

LIN=(RUN,label)
Specifies that the copy modules are located on the job's $Y$RUN file, with the file identifier (label) specified by the user.

LIN=(*,label)
Specifies that the copy modules are located on a catalog file identified by the file identifier (label).

If more than one copy library is present, the additional libraries are specified with the LINn parameter. The n value varies from a minimum of 1 to a maximum of 9. The compiler searches the libraries in the following order: LIN, LIN1, LIN2, etc, through LIN9.

NOTE:

Use the LINn format only with JPROC calls, not in PARAM statements (see A.2 for the PARAM statement format).

OBJ Keyword Parameter:

This parameter specifies the output file definition and generates a PARAM OBJ control statement. The options are:

OBJ=(vol-ser-no,label)

Specifies the file identifier (label) and the volume serial number (vol-ser-no) where the object module is located.

OBJ=(RES,label)

Specifies that the object module is located on the SYSRES device, with the file identifier specified by the label parameter.

OBJ=(RUN,label)

Specifies that the object module is located on the job's $Y$RUN file, with a file identifier (label) of user-own specification.

OBJ=(*,label)

Specifies that the object module is located on a catalog file identified by the file identifier (label).

If omitted, the object module is located on the job's $Y$RUN file.

NOTE:

The OBJ keyword parameter must not be used with COBL74L or COBL74LG.

SCR1 Keyword Parameter:

SCR1=$\left\{\begin{array}{l}\text{vol-ser-no}\\ \textbf{RES}\end{array}\right\}$

Specifies the volume serial number of the work file with an identifier of $SCR1.

SCR2 Keyword Parameter:

SCR2=$\left\{\begin{array}{l}\text{vol-ser-no}\\ \textbf{RES}\end{array}\right\}$

Specifies the volume serial number of the work file with an identifier of $SCR2.

SCR3 Keyword Parameter:

SCR3=$\left\{\begin{array}{l}\text{vol-ser-no}\\ \textbf{RUN}\end{array}\right\}$

Specifies the volume serial number of the work file with an identifier of $SCR3.

ALTLOD Keyword Parameter:

ALTLOD=(vol-ser-no,label)

Specifies the location of the compiler to be used, if other than $Y$LOD.

ALTLOD=(RES,label)

Specifies that the alternate load library is located on the job's SYSRES device in the file identified by the file identifier (label).

ALTLOD=(RUN,label)

Specifies that the alternate load library is located on the job's $Y$RUN file, with the file identifier (label) specified by the user.

ALTLOD=(*,label)

Specifies that the alternate load library is located on a catalog file identified by the file identifier (label).

If omitted, the compiler is loaded from $Y$RUN for execution and $Y$LOD for compilation and linking.

Option Keyword Parameter:

`option=specification`

Specifies the various compiler options parameters, such as LIST=YES, etc. (See Appendix A.) It cannot specify the IN, LIN, and OBJ parameters because these are job parameters that require job control language automatically generated by a JPROC.

ERRFIL Keyword Parameter:

`ERRFIL=(vol-ser-no,label,module-name)]`

Defines an error file for compile-time diagnostics and generates a PARAM ERRFIL control statement. The module name, volume serial number, and file identifier (label) must be specified.

NOTE:

The ERRFIL keyword parameter must not be specified unless the IN keyword parameter is also specified.

Example 1a:

The following illustrates the use of the COBL74 procedure call statement in its basic form:

```
    1        10     16                                                        72
   ------------------------------------------------------------------------------
1. | // JOB COBOL1A
2. | // COBL74
3. | /$
4. |   .
5. |   . source deck
6. |   .
7. | /*
```

Line	Explanation
1	Indicates that the name of the job is COBOL1A
2	Indicates the name of the procedure being called (COBL74). There are no keyword parameters specifying special options for this compilation.
3	Indicates start of data
4–6	Represents the source deck to be compiled
7	Indicates end of data

As coded, this example can be the first step in a job to be followed by the link-edit jproc call, or it can be an entire job in itself by specifying a /& (end-of-job) statement and a // FIN (terminate card reader operations) statement on lines 8 and 9, respectively. The latter case could be used to test-compile a new program or an updated version of an existing program.

Example 1b:

The basic form given in example 1a generates the following control stream:

```
      1          10     16                                                   72
    ----------------------------------------------------------------------------
 1. | // JOB COBOL1A
 2. | // DVC 2Ø/   // LFD PRNTR
 3. | // DVC RES
 4. | // EXT ST,C,3,CYL,1
 5. | // LBL $SCR1  // LFD $SCR1
 6. | // DVC RES
 7. | // EXT ST,C,3,CYL,1
 8. | // LBL $SCR2  // LFD $SCR2
 9. | // DVC RUN
1Ø. | // EXT ST,C,3,CYL,1
11. | // LBL $SCR3  // LFD $SCR3
12. | // EXEC COBL74
13. | /$
14. |      .
15. |      . source deck
16. |      .
17. | /*
```

Line	Explanation
1	Indicates that the name of the job is COBOL1A
2	Indicates the default logical unit number and LFD name of the printer
3–5	Indicates that the first work file needed for compiling is, by default, on the SYSRES device, has both a file identifier and LFD name of $SCR1, and uses the sequential access technique; that allocation is contiguous, with three cylinders allocated for the secondary increment and one cylinder of initial allocation.
6–8	Identifies the second work file needed for compiling. The only difference between this work file and the first work file is that file identifier and LFD name are $SCR2 rather than $SCR1.
9–11	Indicates that the third work file needed for compiling is, by default, on the device containing the job's YRUN file. Both the file identifier and the LFD name are $SCR3, and the file extent specification is the same as the first and second work files.
12	Loads the COBL74 compiler from YLOD
13	Indicates start of data
14–16	Represents the source deck to be compiled
17	Indicates end of data

As with example 1a, this example can be the first step in a job, or it can be the entire job in itself by specifying the /& statement and the // FIN statement on lines 18 and 19, respectively.

Example 2a:

The following example illustrates the use of a COBL74 procedure call statement that defines many of the keyword parameters:

```
    1         10     16                                                    72
   ----------------------------------------------------------------------------
1. | // JOB COBOL2A
2. | //PROGNM COBL74 PRNTR=21,IN=(RES,U$SRC),                              X
3. | //1             OBJ=(DSC2,U$OBJ),                                     X
4. | //2             LIN=(DSC1,COPYLIB1),                                  X
5. | //3             SCR1=DSC4,SCR2=DSC1,                                  X
6. | //4             LSTREF=YES,OBJLST=NO,AXREF=YES,                       X
7. | //5             PROVER=YES
8. | /&
9. | // FIN
```

Line Explanation

1 Indicates that the name of the job is COBOL2A

2 Indicates the name of the procedure being called (COBL74). The source module name is PROGNM. The logical unit number of the printer is 21, and the input file is on the SYSRES device, with a file identifier of U$SRC.

3 Indicates that the output file volume serial number is DSC2, with a file identifier of U$OBJ

4 Indicates that the copy module volume serial number is DSC1, with a file identifier of COPYLIB1

5 Indicates that the second work file needed for compiling is on the device with a volume serial number of DSC4, and the third work file is on the device with a volume serial number of DSC1. By default, the device for the first work file is the SYSRES device.

6-7 Indicates the compiler options for this compilation:

LSTREF=YES
Specifies the generation of a source program listing including the definition line numbers of operands.

OBJLST=NO
Specifies not to generate an object code listing.

AXREF=YES
Specifies the generation of an alphabetically ordered cross-reference listing of procedure and data names.

PROVER=YES
Specifies the generation of an address map list of procedure names and verb statements.

8 End of job

9 Terminates card reader operations

As written, this example is a 1-step job that compiles your source program. It produces a nonexecutable object module. Before your program could be executed, a job step would have to be inserted in the control stream that would link-edit the object module to produce an executable load module.

Example 2b:

Based on the keyword parameters specified in example 2a, the following control stream is generated:

```
     1        10     16                                                       72
    ------------------------------------------------------------------------------
1.  |// JOB COBOL2A
2.  |// DVC 21  // LFD PRNTR
3.  |// DVC RES
4.  |// LBL U$SRC    // LFD INCPUT
5.  |// DVC 5Ø    // VOL DSC2
6.  |// LBL U$OBJ    // LFD OUTCPUT
7.  |// DVC 51    // VOL DSC1
8.  |// LBL COPYLIB1  // LFD COPY$
9.  |// DVC RES
1Ø. |// EXT ST,C,3,CYL,1
11. |// LBL $SCR1    // LFD $SCR1
12. |// DVC 52    // VOL DSC4
13. |// EXT ST,C,3,CYL,1
14. |// LBL $SCR2    // LFD $SCR2
15. |// DVC 51    // VOL DSC1
16. |// EXT ST,C,3,CYL,1
17. |// LBL $SCR3    // LFD $SCR3
18. |// EXEC COBL74
19. |// PARAM IN=PROGNM/INCPUT
2Ø. |// PARAM OBJ=OUTCPUT
21. |// PARAM LSTREF=YES
22. |// PARAM OBJLST=NO
23. |// PARAM AXREF=YES
24. |// PARAM PROVER=YES
25. |/&
26. |// FIN
```

Line   Explanation

1   Indicates that the name of the job is COBOL2A

2   Indicates that the printer is to be assigned to the logical unit number 21, with an LFD name of PRNTR. This was obtained from line 2 in example 2a.

3   Indicates that the input file is on the device containing the SYSRES volume. This was obtained from the IN parameter on line 2 in example 2a.

4   Indicates that the input file has a file identifier of U$SRC, with an LFD name of INCPUT. This was obtained from the IN parameter on line 2 in example 2a.

5   Indicates that the output file volume serial number is DSC2. This was obtained from the OBJ parameter on line 3 in example 2a. It is assigned to the device with a logical unit number of 50, which was the first available number in the range of 50-54.

6   Indicates that the output file has a file identifier of U$OBJ, with an LFD name of OUTCPUT. This was obtained from the OBJ parameter on line 3 in example 2a.

7   Indicates that the copy library has a volume serial number of DSC1. It is assigned to the device with a logical unit number of 51, which was the next available number in the range of 50-54. Logical unit number 50 was already assigned to the device with a volume serial number of DSC2 (line 5), so the next available logical unit number is used. This was obtained from the LIN parameter on line 4 in example 2a.

Line Explanation

8 Indicates that the copy library is labeled COPYLIB1, with an LFD name of COPY$. This was obtained from the LIN parameter on line 4 in example 2a.

9–11 Indicates that the first work file needed for compiling is, by default, on the SYSRES device, has both a file identifier and LFD name of $SCR1, and uses the sequential access technique; and that allocation is contiguous, with three cylinders allocated for the secondary increment and one cylinder of initial allocation.

12–14 Indicates that the second work file needed for compiling has a volume serial number of DSC4. This volume serial number has not been previously used in this job, so the next available logical unit number (52) is assigned to this device. This work file has both a file identifier and LFD name of $SCR2, and has the same file extent specification as the first work file. This was obtained from the SCR2 parameter on line 5 in example 2a.

15–17 Indicates that the third work file needed for compiling has a volume serial number of DSC1. Since this volume serial number was already used, this work file uses the same device logical unit number of 51. This work file has both a file identifier and LFD name of $SCR3, and has the same file extent specification as the first and second work files. This was obtained from the SCR3 parameter on line 5 in example 2a.

18 Loads the COBL74 compiler from YLOD

19–24 PARAM control statements identify the processing options for the ANSI 1974 COBOL compiler (COBL74). These are generated in the following manner:

Line 19 - The module name PROGNM is generated from the label field in line 2 of example 2a. The filename INCPUT is generated automatically when the IN parameter is specified.

Line 20 - The filename OUTCPUT is generated automatically when the OBJ parameter is used.

Lines 21-24 - Generated by the compiler options specifications of lines 6 and 7 in example 2a.

25 End of job

26 Terminates card reader operations

Example 3a:

The following example illustrates the use of the COBL74L procedure call statement:

```
     1          10     16                                                  72
 1.  // JOB MASTER
 2.  // DVC 50
 3.  // VOL DSC1
 4.  // LBL U$LOD
 5.  // LFD LNKLIB
 6.  // COBL74 MXREF=YES,PROVER=YES
 7.  /$
     .
 8.  .  COBOL source program
     .
 9.  /*
10.  /$
11.     LINKOP OUT=LNKLIB
12.     LOADM ABC123
13.  /*
14.  /&
15.  // FIN
```

Line	Explanation
1	Indicates that the name of the job is MASTER
2-5	Defines a file U$LOD on volume DSC1 to be used to hold the linked object module
6	Indicates the name of the procedure being called (COBL74L) and indicates the compiler options for this compilation
7	Indicates start of data
8	Indicates the COBOL source program
9	Indicates end of data
10	Indicates start of data
11	Indicates that the linkage editor is to write the load module to the file with the lfdname LNKLIB
12	Indicates that the name of the load module is ABC123
13	Indicates end of data
14	Indicates end of job
15	Terminates card reader operations

Example 3b:

Based on the keyword parameters specified explicitly and implicitly in example 3a, the following control stream is generated:

```
     1         10    16                                                  72
    ------------------------------------------------------------------------
1.  // JOB MASTER
2.  // DVC 50  // VBL DSC1  // LBL U$LOD  // LFD LNKLIB
3.  // DVC 20  // LFD PRNTR
4.  // DVC RES  // EXT ST,C,3,CYL,1  // LBL $SCR1  // LFD $SCR1
5.  // DVC RES  // EXT ST,C,3,CYL,1  // LBL $SCR2  // LFD $SCR2
6.  // DVC RES  // EXT ST,C,3,CYL,1  // LBL $SCR3  // LFD $SCR3
7.  // EXEC COBL74
8.  // PARAM MXREF=YES,PROVER=YES
9.  /$
    .
10. .  COBOL source program
    ,
11. /*
12. // DVC RES  // EXT ST,C,3,CYL,1  // LBL $SCR1 // LFD $SCR1
13. // EXEC LNKEDT
14. /$
       LINKOP OUT=LNKLIB
15.
16.    LOADM ABC123
17. /*
18. &
19. // FIN
```

Line	Explanation
1	Indicates that the name of the job is MASTER
2	Defines a file U$LOD on volume DSC1 to be used to hold the linked object module
3	Indicates that the printer is to be assigned to logical unit number 20 with an lfdname of PRNTR
4–6	Defines the three work files necessary for compiler execution
7	Loads and executes the COBL74 compiler
8	Indicates parameter options
9	Indicates start of data
10	Indicates the COBOL source program
11	Indicates end of data
12	Defines the work file necessary for LNKEDT execution
13	Loads and executes the linkage editor

14 Indicates start of data

15 Indicates that the linkage editor is to write the load module to the file with the lfdname LNKLIB

16 Indicates that the name of the load module is ABC123

17 Indicates end of data

18 Indicates end of job

19 Terminate card reader operations

Example 4a:

The following example illustrates the use of the COBL74LG procedure call statement. The input file and the output listings for the compiler are defined.

```
   1        10    16                                                      72
1. // JOB   MASTER
2. // DVC...// LFD
3. //MASTER COBL74LG  IN=(ABC123,PAYMAST)       X
4. //1            LSTREF=YES,AXREF=YES,PROVER=YES
5. /&
6. // FIN
7.    data
8. /*
```

Line Explanation

1 Indicates that the name of the job is MASTER

2 Indicates the device assignment sets needed to execute the resulting load module

3 Indicates that the name of the source module is MASTER and that the name of the procedure being called is COBL74LG. Therefore, this example compiles, link-edits, and executes the source program MASTER. The input file (source language) is on the device with a volume serial number of ABC123 and has a file identifier of PAYMAST.

4 Indicates the compiler listing options

5 End of job

6 Terminates card reader operations

7 Indicates the data for the program

8 Indicates end of data

Example 4b:

Based on the keyword parameters specified explicitly and implicitly in example 4a, the following control stream is generated:

```
     1          10     16                                                      72
    ---------------------------------------------------------------------------------
 1. // JOB    MASTER
 2. // DVC...// LFD
 3. // OPTION  LINK,GO
 4. // DVC     2Ø     // LFD PRNTR
 5. // DVC    5Ø     // VOL ABC123
 6. // LBL    PAYMAST  // LFD INCPUT
 7. // DVC    RES
 8. // EXT    ST,C,3,CYL,1
 9. // LBL    $SCR1 // LFD $SCR1
10. // DVC    RES
11. // EXT    ST,C,3,CYL,1
12. // LBL    $SCR2 // LFD $SCR2
13. // DVC    RUN
14. // EXT    ST,C,3,CYL,1
15. // LBL    $SCR3 // LFD $SCR3
16. // EXEC   COBL74
17. // PARAM IN=MASTER/INCPUT
18. // PARAM LSTREF=YES
19. // PARAM AXREF=YES
20. // PARAM PROVER=YES
21. /&
22. // FIN
23.    data
24. /*
```

Line   Explanation

1   Indicates that the name of the job is MASTER

2   Indicates the device assignment sets needed to execute the resulting load module

3   Indicates that the source program is to be link-edited and then executed after it has been compiled. This was obtained from COBL74LG specified on line 2 in example 4a.

4   Indicates, by default, that the printer is to be assigned to the logical unit number 20, with an LFD name of PRNTR

5   Indicates that the input file (source language) is on the device with the logical unit number of 50 and has a volume serial number of ABC123. This was obtained from the IN parameter on line 2 in example 4a.

6   Indicates that the input file (source language) has a file identifier of PAYMAST, with an LFD name of INCPUT. This was obtained from the IN parameter on line 2 in example 4a.

7-9 Indicates, by default, that the first work file needed for compiling is on the SYSRES device, has both a file identifier and LFD name of $SCR1, uses sequential access technique; that allocation is contiguous, with three cylinders allocated for the secondary increment and one cylinder of initial allocation.

10-12 Indicates, by default, that the second work file needed for compiling is on the SYSRES device. This work file has both a file identifier and LFD name of $SCR2. It has the same file extent specification as the first work file.

13-15 Indicates, by default, that the third work file needed for compiling is on the SYSRUN device. This work file has both a file identifier and LFD name of $SCR3. It has the same file extent specification as the first and second work file.

16 Loads the COBL74 compiler from $Y$LOD

17-20 PARAM control statements that identify the processing options for the COBL74 compiler. These are generated as follows:

Line 16 - The module name MASTER is generated from the label field on line 2 of example 4a. The filename INCPUT is generated automatically when the IN parameter is specified.

Lines 17-19 - Generated by the compiler options specifications of line 3 in example 4a.

21 End of job

22 Terminates card reader operations

23 Indicates the data for the program

24 Indicates end of data

Any output from the compiler is temporarily stored on the $Y$RUN device.

Implicit in the // OPTION LINK,GO statement on line 2 of example 4b is the creation of a load module named LNKLOD by the linkage editor and the execution of that load module. This is performed after the source program has been compiled.

Example 5:

The following example shows a typical compilation from a workstation:

```
     1         10      16                                                      72
   ------------------------------------------------------------------------------
1. | LOGON SYSPUBS,6944,DOIT
2. | /EDT
   | . )
   | . } COBOL source program
   | . )
   | @WRITE
   | @HALT
3. | RV JC$BUILD
   | .
   | .  job control stream
4. | RV COMPIL
5. | LOGOFF
```

Line	Explanation
1	Identifies the user-id SYSPUBS, the account-number 6944, and the password DOIT
2	Calls the system editor. The COBOL source program immediately follows /EDT command. After the last COBOL source statement, the @WRITE command is issued to save the source program in a library. The @HALT command ends the EDT session.
3	Calls the system build command. This command writes the job control stream to the system job control stream library file ($Y$JCS). This job control stream defines the system resources the source program requires.
4	Calls the control stream from $Y$JCS. This step compiles the source program.
5	Ends the workstation session

Example 6:

The following example also shows compilation from a workstation:

```
      1          10     16
1.  | LOGON      user-id
2.  | /EDT
    |     .)
    |     .}COBOL source program
    |     .)
    |     @WRITE INPUT1
    |     // JOB COMPIL
    |     //COBOL1 COBL74,IN=INPUT1
    |     /&
    |     @WRITE $Y$JCS
    |     @HALT
3.  | RV COMPIL
4.  | LOGOFF
```

Line	Explanation
1	Connects the workstation terminal to the system and also identifies the user
2	Calls the system editor (EDT)
3	Compiles the program
4	Disconnects the workstation terminal from the system

H.3. COMPILER STATUS INDICATORS

The compiler sets the following status indicators in the user program switch indicator (UPSI) byte. These indicators may be used in conjunction with the // SKIP job control card:

- Switch-0 (X'80') is set to 1 if the compiler does not create a complete object module. This condition might be caused by an "insufficient memory available" diagnostic or the SPROUT option.

- Switch-1 (X'40') is set to 1 if the compiler issues any diagnostic message with severity code 2 or 3.
- Switch-2 (X'20') is set to 1 if the compiler issues any diagnostic messages with the severity code 1.

H.4. DATA DEFINITION (DD) JOB CONTROL STATEMENT KEYWORD PARAMETERS

The DD job control statement is used to change data management keywords at execution time. Instead of changing your COBOL source code, you can override data management keyword specifications when your COBOL object program is executing. The DD statement keyword parameters that may be specified are:

ACCESS=$\left\{\begin{array}{l}\text{EXC}\\ \text{EXCR}\\ \text{SRDO}\\ \text{SRD}\\ \text{SUPD}\\ \text{SADD}\end{array}\right\}$

FILABL= $\left\{\begin{array}{l}\text{NO}\\ \text{NSTD}\\ \text{STD}\end{array}\right\}$

LACE=n

RCB= $\left\{\begin{array}{l}\text{YES}\\ \text{NO}\end{array}\right\}$

RECV= $\left\{\begin{array}{l}\text{YES}\\ \text{FCE}\end{array}\right\}$

SIZE= $\left\{\begin{array}{l}\text{AUTO}\\ \text{nn}\end{array}\right\}$

TPMARK= NO

UOS=n

VMNT= $\left\{\begin{array}{l}\text{ONE}\\ \text{NO}\end{array}\right\}$

VSEC= $\left\{\begin{array}{l}\text{YES}\\ \text{nn}\end{array}\right\}$

When specifying these keyword parameters, extreme care must be used so that the effect of changing one parameter does not cause a conflict with another parameter. To avoid conflicts, the user should carefully examine the file usage specified in COBOL programs and the default parameters set by the compiler-generated data management specifications.

The DD statement applies to basic data management users and consolidated data management users. For keyword parameter information, see the basic data management user guide, UP-8068 (current version), or the consolidated data management macroinstructions user guide, UP-8826 (current version). A complete description of the DD job control statement is explained in the job control user guide, UP-8065 (current version).

Appendix I. Reserved Words

The following reserved words are part of the OS/3 COBOL language structure and cannot be used as user-defined words or system-names.

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

BEFORE
BEGINNING
BLANK
BLOCK
BLOCK-COUNT
BOTTOM
BY

CALL
CANCEL
CD
CF

CH
CHANGED
CHARACTER
CHARACTERS
CLASS-NAME
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONNECT-FREE
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY

CYLINDER-INDEX
CYLINDER-OVERFLOW

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
ENDING
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ESI
EVERY
EXCEPTION
EXHIBIT
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILE-ID
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
FUNCTION-KEYS

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEX-AREA
INDEXED
INDICATE
INDICES
INITIAL
INITIATE

INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
ISAM

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
MESSAGES
MODE
MODULES
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY

NAMED
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC

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

PAGE
PAGE-COUNTER
PASSWORD
PERCENT
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PRINTER
QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
READY
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

SAM
SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-FILE-SIZE
SORT-MERGE
SORT-MODE-SIZE
SOURCE
SOURCE-ALPHABET
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
SPECIFIC

STANDARD
STANDARD-0
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED
SYSCHAN-n (n=1 thru 15)
SYSCOM
SYSCONSOLE
SYSFORMAT
SYSIN
SYSIPT
SYSLOG
SYSLST
SYSOPT
SYSOUT
SYSSCOPE
SYSSWCH
SYSSWCH-n (n=0 thru 31)
SYSTEM
SYSTEM-SHUTDOWN
SYSTERMINAL
SYSWORK

TABLE
TALLYING
TAPE
TAPES
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME

TIMES
TO
TOP
TRACE
TRAILING
TRANSFORM
TYPE

UNIT
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING
VERIFY

WHEN
WHEN-COMPILED
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

*DEBUG
+
-
*
/
**
>
<
=
==

Appendix J. Character Sets

The Extended Binary Coded Decimal Interchange Code (EBCDIC) is the standard character set for OS/3. The EBCDIC character set is given in Table J–1.

This table also provides the means for converting the American Standard Code for Information Interchange (ASCII) 8-bit code (ASCII-8) to EBCDIC.

Table J—1. Correspondence between EBCDIC, ASCII-8, and Punched Card Codes (Part 1 of 3)

EBCDIC		Character Name	Symbol	Card Punches	ASCII-8 (Hex.)
Dec.	Hex.				
0	00	NUL		12-0-9-8-1	00
1	01	SOH		12-9-1	01
2	02	STX		12-9-2	02
3	03	ETX		12-9-3	03
4	04			12-9-4	04
5	05			12-9-5	09
6	06			12-9-6	86
7	07	DEL (Delete)		12-9-7	7F
8	08			12-9-8	97
9	09			12-9-8-1	8D
10	0A			12-9-8-2	8E
11	0B	VT		12-9-8-3	0B
12	0C	FF		12-9-8-4	0C
13	0D	CR		12-9-8-5	0D
14	0E	SO		12-9-8-6	0E
15	0F	SI		12-9-8-7	0F
16	10	DLE		12-11-9-8-1	10
17	11	DC1		11-9-1	11
18	12	DC2		11-9-2	12
19	13	DC3		11-9-3	13
20	14			11-9-4	9D
21	15			11-9-5	85
22	16	BS		11-9-6	08
23	17			11-9-7	87
24	18	CAN		11-9-8	18
25	19	EM		11-9-8-1	19
26	1A			11-9-8-2	92
27	1B			11-9-8-3	8F
28	1C	FS		11-9-8-4	1C
29	1D	GS		11-9-8-5	1D
30	1E	RS		11-9-8-6	1E
31	1F	US		11-9-8-7	1F
32	20			11-0-9-8-1	80
33	21			0-9-1	81
34	22			0-9-2	82
35	23			0-9-3	83
36	24			0-9-4	84
37	25	LF		0-9-5	0A
38	26	ETB		0-9-6	17
39	27	ESC		0-9-7	1B
40	28			0-9-8	88
41	29			0-9-8-1	89
42	2A			0-9-8-2	8A
43	2B			0-9-8-3	8B
44	2C			0-9-8-4	8C
45	2D	ENQ		0-9-8-5	05
46	2E	ACK		0-9-8-6	06
47	2F	BEL		0-9-8-7	07
48	30			12-11-0-9-8-1	90
49	31			9-1	91
50	32	SYN		9-2	16
51	33			9-3	93
52	34			9-4	94
53	35			9-5	95
54	36			9-6	96
55	37	EOT		9-7	04
56	38			9-8	98
57	39			9-8-1	99
58	3A			9-8-2	9A
59	3B			9-8-3	9B
60	3C	DC4		9-8-4	14
61	3D	NAK		9-8-5	15
62	3E			9-8-6	9E
63	3F	SUB		9-8-7	1A
64	40	SP (Space)			20
65	41			12-0-9-1	A0
66	42			12-0-9-2	A1
67	43			12-0-9-3	A2
68	44			12-0-9-4	A3
69	45			12-0-9-5	A4
70	46			12-0-9-6	A5
71	47			12-0-9-7	A6
72	48			12-0-9-8	A7
73	49			12-8-1	A8
74	4A	Opening bracket	[	12-8-2	5B
75	4B	Period, decimal	.	12-8-3	2E
76	4C	Less than	<	12-8-4	3C
77	4D	Opening parenthesis	(	12-8-5	28
78	4E	Plus sign	+	12-8-6	2B
79	4F	Exclamation point	!	12-8-7	21
80	50	Ampersand	&	12	26
81	51			12-11-9-1	A9
82	52			12-11-9-2	AA
83	53			12-11-9-3	AB
84	54			12-11-9-4	AC
85	55			12-11-9-5	AD
86	56			12-11-9-6	AE
87	57			12-11-9-7	AF
88	58			12-11-9-8	B0
89	59			11-8-1	B1
90	5A	Closing bracket	]	11-8-2	5D
91	5B	Dollar sign	$	11-8-3	24
92	5C	Asterisk	*	11-8-4	2A
93	5D	Closing Parenthesis	)	11-8-5	29
94	5E	Semicolon	;	11-8-6	3B
95	5F	Circumflex	^	11-8-7	5E
96	60	Minus sign, hyphen	—	11	2D
97	61	Slash, virgule, solidus	/	0-1	2F
98	62			11-0-9-2	B2
99	63			11-0-9-3	B3
100	64			11-0-9-4	B4
101	65			11-0-9-5	B5
102	66			11-0-9-6	B6
103	67			11-0-9-7	B7

Table J—1. Correspondence between EBCDIC, ASCII-8, and Punched Card Codes (Part 2 of 3)

EBCDIC		Character Name	Symbol	Card Punches	ASCII-8 (Hex.)
Dec.	Hex.				
104	68			11-0-9-8	B8
105	69			0-8-1	B9
106	6A	Vertical line	|	12-11	7C
107	6B	Comma, cedilla	,	0-8-3	2C
108	6C	Percent sign	%	0-8-4	25
109	6D	Underline	_	0-8-5	5F
110	6E	Greater than	>	0-8-6	3E
111	6F	Question mark	?	0-8-7	3F
112	70			12-11-0	BA
113	71			12-11-0-9-1	BB
114	72			12-11-0-9-2	BC
115	73			12-11-0-9-3	BD
116	74			12-11-0-9-4	BE
117	75			12-11-0-9-5	BF
118	76			12-11-0-9-6	C0
119	77			12-11-0-9-7	C1
120	78			12-11-0-9-8	C2
121	79	Grave accent	`	8-1	60
122	7A	Colon	:	8-2	3A
123	7B	Number sign, pound sign	#	8-3	23
124	7C	Commerical at symbol	@	8-4	40
125	7D	Apostrophe, acute accent	'	8-5	27
126	7E	Equal sign	=	8-6	3D
127	7F	Quotation mark, dieresis	"	8-7	22
128	80			12-0-8-1	C3
129	81	a	a	12-0-1	61
130	82	b	b	12-0-2	62
131	83	c	c	12-0-3	63
132	84	d	d	12-0-4	64
133	85	e	e	12-0-5	65
134	86	f	f	12-0-6	66
135	87	g	g	12-0-7	67
136	88	h	h	12-0-8	68
137	89	i	i	12-0-9	69
138	8A			12-0-8-2	C4
139	8B			12-0-8-3	C5
140	8C			12-0-8-4	C6
141	8D			12-0-8-5	C7
142	8E			12-0-8-6	C8
143	8F			12-0-8-7	C9
144	90			12-11-8-1	CA
145	91	j	j	12-11-1	6A
146	92	k	k	12-11-2	6B
147	93	l	l	12-11-3	6C
148	94	m	m	12-11-4	6D
149	95	n	n	12-11-5	6E
150	96	o	o	12-11-6	6F
151	97	p	p	12-11-7	70
152	98	q	q	12-11-8	71
153	99	r	r	12-11-9	72
154	9A			12-11-8-2	CB
155	9B			12-11-8-3	CC
156	9C			12-11-8-4	CD
157	9D			12-11-8-5	CE
158	9E			12-11-8-6	CF
159	9F			12-11-8-7	D0
160	A0			11-0-8-1	D1
161	A1	Overline, tilde	~	11-0-1	7E
162	A2	s	s	11-0-2	73
163	A3	t	t	11-0-3	74
164	A4	u	u	11-0-4	75
165	A5	v	v	11-0-5	76
166	A6	w	w	11-0-6	77
167	A7	x	x	11-0-7	78
168	A8	y	y	11-0-8	79
169	A9	z	z	11-0-9	7A
170	AA			11-0-8-2	D2
171	AB			11-0-8-3	D3
172	AC			11-0-8-4	D4
173	AD			11-0-8-5	D5
174	AE			11-0-8-6	D6
175	AF			11-0-8-7	D7
176	B0			12-11-0-8-1	D8
177	B1			12-11-0-1	D9
178	B2			12-11-0-2	DA
179	B3			12-11-0-3	DB
180	B4			12-11-0-4	DC
181	B5			12-11-0-5	DD
182	B6			12-11-0-6	DE
183	B7			12-11-0-7	DF
184	B8			12-11-0-8	E0
185	B9			12-11-0-9	E1
186	BA			12-11-0-8-2	E2
187	BB			12-11-0-8-3	E3
188	BC			12-11-0-8-4	E4
189	BD			12-11-0-8-5	E5
190	BE			12-11-0-8-6	E6
191	BF			12-11-0-8-7	E7
192	C0	Opening brace	{	12-0	7B
193	C1	A	A	12-1	41
194	C2	B	B	12-2	42
195	C3	C	C	12-3	43
196	C4	D	D	12-4	44
197	C5	E	E	12-5	45
198	C6	F	F	12-6	46
199	C7	G	G	12-7	47
200	C8	H	H	12-8	48
201	C9	I	I	12-9	49
202	CA			12-0-9-8-2	E8
203	CB			12-0-9-8-3	E9
204	CC			12-0-9-8-4	EA
205	CD			12-0-9-8-5	EB
206	CE			12-0-9-8-6	EC
207	CF			12-0-9-8-7	ED

Table J—1. Correspondence between EBCDIC, ASCII-8, and Punched Card Codes (Part 3 of 3)

EBCDIC		Character Name	Symbol	Card Punches	ASCII-8 (Hex.)
Dec.	Hex.				
208	D0	Closing brace	}	11-0	7D
209	D1	J	J	11-1	4A
210	D2	K	K	11-2	4B
211	D3	L	L	11-3	4C
212	D4	M	M	11-4	4D
213	D5	N	N	11-5	4E
214	D6	O	O	11-6	4F
215	D7	P	P	11-7	50
216	D8	Q	Q	11-8	51
217	D9	R	R	11-9	52
218	DA			12-11-9-8-2	EE
219	DB			12-11-9-8-3	EF
220	DC			12-11-9-8-4	F0
221	DD			12-11-9-8-5	F1
222	DE			12-11-9-8-6	F2
223	DF			12-11-9-8-7	F3
224	E0	Reverse slash	\	0-8-2	5C
225	E1			11-0-9-1	9F
226	E2	S	S	0-2	53
227	E3	T	T	0-3	54
228	E4	U	U	0-4	55
229	E5	V	V	0-5	56
230	E6	W	W	0-6	57
231	E7	X	X	0-7	58
232	E8	Y	Y	0-8	59
233	E9	Z	Z	0-9	5A
234	EA			11-0-9-8-2	F4
235	EB			11-0-9-8-3	F5
236	EC			11-0-9-8-4	F6
237	ED			11-0-9-8-5	F7
238	EE			11-0-9-8-6	F8
239	EF			11-0-9-8-7	F9
240	F0	0	0	0	30
241	F1	1	1	1	31
242	F2	2	2	2	32
243	F3	3	3	3	33
244	F4	4	4	4	34
245	F5	5	5	5	35
246	F6	6	6	6	36
247	F7	7	7	7	37
248	F8	8	8	8	38
249	F9	9	9	9	39
250	FA			12-11-0-9-8-2	FA
251	FB			12-11-0-9-8-3	FB
252	FC			12-11-0-9-8-4	FC
253	FD			12-11-0-9-8-5	FD
254	FE			12-11-0-9-8-6	FE
255	FF	EO (Eight ones)		12-11-0-9-8-7	FF

Appendix K. PICTURE Clause

K.1. GENERAL

This appendix is intended as a tutorial guide to using the PICTURE clause. It is not intended to replace the standard text that covers more detailed rules governing the PICTURE clause and its relation to other clauses in other divisions of a COBOL program.

K.2. USE OF THE PICTURE CLAUSE AND ITS SYMBOLS

The PICTURE clause describes the general characteristics and editing requirements of an elementary data item. It must be specified for every elementary item but is not allowed with an index data item, an internal floating-point data item, or a group item.

The format of the PICTURE clause is:

```
    PICTURE IS character-string.
```

A character-string in a PICTURE clause consists of certain allowable combinations of PICTURE symbols. The allowable combinations determine the category of the elementary item. The length of the character-string can be from 1 to 30 characters.

The standard for COBOL defines three types of PICTURE symbols: data type, sign and assumed decimal point, and editing symbols:

1. Data Type Symbols

   The data type symbols are: A, 9, and X. A defines alphabetic data; 9 defines numeric data; X defines alphanumeric data.

2. Sign and Assumed Decimal Point Symbols

   These symbols are: S, V, and P. S describes the presence of an operational sign; V indicates the location of the assumed decimal point; and P specifies the location of the assumed decimal point when the point is not within or adjacent to the digits that appear in the data item.

3. Editing Symbols

There are two types of editing symbols:

- Insertion editing symbols
- Zero suppression and replacement editing symbols

The insertion editing symbols are: B O / , . + – $ (or alternate currency symbol), CR, and DB. Among these symbols, the CS, +, and – serve both as fixed insertion symbols and floating insertion symbols. The rstg are fixed insertion symbols.

The zero suppression and replacement editing symbols are: Z and *.

The following symbols can appear only once in one PICTURE character string:

S V . CR DB

The following symbols can appear more than once in one PICTURE character-string:

A B P X Z 9 Ø / , + - * $

NOTE:

The PICTURE symbols used to describe an external floating point data item are: +, –, 9, V, and E. Floating point data items are the OS/3 COBOL extensions to the standard COBOL. For more on the use and meaning of external floating point PICTURE character strings, see 5.3.3.4.

K.3. DESCRIPTIONS AND EXAMPLES OF PICTURE CLAUSE SYMBOLS

The following subparagraphs describe each of the standard COBOL PICTURE symbols. Examples are provided with each description. It is assumed in the following examples that the DECIMAL-POINT IS COMMA and CURRENCY SIGN IS literal clauses are not specified.

Symbol	Explanation
A	Each A in the character string represents a character position that contains a letter or a space. The symbol defines an alphabetic, alphanumeric, or alphanumeric edited data item and is counted in the size of the data item.

Examples:

```
PICTURE IS A(1Ø).
```

The data item is 10 bytes long and alphabetic.

```
PICTURE IS AAA9999.
```

The data item is seven bytes long and alphanumeric (because of the mixture of letters and numbers). It is equivalent to the PICTURE string X(7).

When you use the character A with any other PICTURE character except B, the data item becomes alphanumeric or alphanumeric edited. In these cases, PICTURE character A is the same as the PICTURE character X.

PICTURE IS XXØ99BA.

The data item is seven bytes long and alphanumeric edited. The largest elementary data item that can be moved to this item without truncation is five bytes (because the 0 and B are editing characters). This picture string is equivalent to the string XXOXXBX.

Symbol Explanation

9 Each 9 in the character string represents a character position that contains a numeral. The symbol defines a numeric or numeric edited data item and is counted in the size of the item. It can also describe an alphanumeric or an alphanumeric edited item.

Examples:

PICTURE IS 999V99.

The data item is five bytes long (unless a USAGE clause is specified) and numeric. The V indicates the position of the decimal point.

PICTURE IS $999.99.

The data item is seven bytes long and numeric edited. The largest numeric item that can be moved to this field without truncation has a PICTURE string of 999V99. Typical values for this data item are: $123.45, and $001.50.

PICTURE IS X99AA.

The data item is five bytes long and alphanumeric. It is equivalent to the PICTURE string X(5).

PICTURE IS AAAB999.

The data item is seven bytes long and alphanumeric edited. It is equivalent to X(3)BX(3).

Symbol Explanation

X Each X in the character string represents a character position that contains any character from X'00' through X'FF', defines an alphanumeric or alphanumeric edited item, and is counted in the size of the item.

Examples:

PICTURE IS XXX.

The data item is three bytes long and alphanumeric.

PICTURE IS XXBBXXX.

The data item is seven bytes long and alphanumeric edited. The largest elementary data item that can be moved to this item without truncation is five bytes long (because of the two B editing characters). A typical value for this item is AB△△CDE.

Symbol Explanation

S

The symbol S in a character string indicates an operational sign in a numeric data item. It is written as the leftmost character in the PICTURE string. The symbol is not counted in the size of the item unless an associated SIGN clause specifies the SEPARATE CHARACTER option.

Examples:

`PICTURE IS S999.`

The data item is three bytes long and numeric. In the absence of a SIGN clause, the sign is a trailing overpunch sign (i.e., an X'C' or X'D' in high order four bits of the last bytes). The value +123 appears on this data item as X'FIF2C3'; the value −123 would appear as X'FIF2D3'.

`PICTURE IS S9(8)V99 SIGN IS LEADING SEPARATE CHARACTER.`

This data item is 11 bytes long and numeric. The value +123.45 appears in this data item as the character string +0000012345.

Symbol Explanation

V

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

Examples:

`PICTURE IS 9(6)V9(2).`

The data item is eight bytes long and numeric. The position of the V indicates that the least significant two digits in this field are to the right of the decimal point. The value 1234 appears as the character string 00123400; the value 789.5 appears as 00078950.

`PICTURE IS V9(5).`

The data item is five bytes long and numeric. The leading digit in the field is immediately to the right of the decimal point. The value .25 appears as the character string 25000.

Symbol Explanation

P

Each P indicates an assumed decimal scaling position. The symbol is used to specify the location of an assumed decimal point where the point is not within the number that appears in the data item. The symbol P is not counted in the size of the item. However, it is counted in determining the maximum number of digit positions allowed for a numeric or numeric edited data item. The symbol P can appear only to the left or the right of 9's as a continuous string of Ps within a PICTURE description.

Examples:

`PICTURE IS PP999.`

The data item is three bytes long, numeric, and the leading digit is two decimal positions to the right of the decimal point (i.e., the maximum value in this data item is .00999). The value .00125 appears in this item as 125. If the value .0123 were moved to this item, it would be truncated and appear as the character string 230 (having the value .0023).

PICTURE IS $$,$$$,$$$PPP.

The data item is 10 bytes long, numeric edited, and the least significant digit is three decimal places to the left of the decimal point (i.e., the smallest nonzero value that can be held in this item is 1000). Such a PICTURE string might be useful for financial reports represented in thousands of dollars. The value 12345000 moved to this item results in the character string △△△$12,345.

Symbol	Explanation
B	Each B represents a character position where a space character is to appear. The symbol may appear in the character string to describe an alphanumeric edited data item. Each B is counted in the size of the data item.

Examples:

PICTURE IS AAAB999.

The data item is seven bytes long and alphanumeric edited. It is equivalent to XXXBXXX.

Sending Field PICTURE	Sending Field Value	Edited Result
X(6)	ABC123	ABC△123
X(5)	ABC12	ABC△12△
X(7)	ABC1234	ABC△123

PICTURE IS 99B99B99.

The data item is eight bytes long, numeric edited, and holds six numeric digits.

Sending Field PICTURE	Sending Field Value	Edited Result
9(6)	123456	12△34△56
9(7)	1234567	23△45△67
9(5)	12345	Ø1△23△45
S9(6)	+123456	12△34△56
S9(6)	-123456	12△34△56

Symbol	Explanation
Ø	Each 0 in the character string represents a character position where the number 0 is inserted. The 0 is counted in the size of the item. The symbol may appear in the PICTURE character string to describe an alphanumeric edited or numeric data item.

Examples:

PICTURE IS XØAAØXX.

The data item is seven bytes long and alphanumeric edited.

Sending Field PICTURE	Sending Field Value	Edited Result
X(5)	ABCDE	AØBCØDE

PICTURE IS $99,999.ØØ.

The data item is 10 bytes long and numeric edited. The largest numeric field that can be moved to the item without truncation has the numeric PICTURE 9(5).

Sending Field PICTURE	Sending Field Value	Edited Result
9(5)	1234	$Ø1,234.ØØ
9(6)V99	123456.78	$23,456.ØØ
S9(3)V99	-1.5	$ØØ,ØØ1.ØØ

Symbol Explanation

/ Each stroke (/) in the character string represents a character position where the stroke character will appear. The / is counted in the size of the item. The symbol may appear in the PICTURE character string to describe an alphanumeric edited or numeric edited data item.

Examples:

PICTURE IS XX/XX/XX.

The data item is eight bytes long and alphanumeric edited.

Sending Field PICTURE	Sending Field Value	Edited Result
X(6)	ABCDEF	AB/CD/EF
X(3)	ABC	AB/CΔ/ΔΔ
X(7)	ABCDEFG	AB/CD/EF

PICTURE IS 99/99/99.

The data item is eight bytes long, numeric edited, and holds six numeric digits.

Sending Field PICTURE	Sending Field Value	Edited Result
9(6)	83Ø228	83/Ø2/28
9(5)	12345	Ø1/23/45
S9(7)V9	-1234567.8	23/45/67

Symbol Explanation

, Each comma (,) represents a character position where the comma character will appear. The symbol is counted in the size of the data item. The symbol is used only to describe a numeric edited data item and must not be the last character in the string.

Examples:

PICTURE IS $9,999,999.

The data item is 10 bytes long, numeric edited, and holds 7 numeric digits.

Sending Field PICTURE	Sending Field Value	Edited Result
9(7)	1234567	$1,234,567
S9(4)	-1234	$0,001,234
9(8)V99	12345678.90	$2,345,678

Symbol Explanation

. The period (.) represents a character position where the period will appear. It also represents the decimal point for alignment purposes. The symbol is counted in the size of the item and must not be the last character in the string. When the DECIMAL-POINT IS COMMA clause is specified in the SPECIAL-NAMES paragraph, the functions of the period and the comma are exchanged.

Examples:

PICTURE IS 999.99.

The data item is six bytes long, numeric edited, and holds five decimal digits.

Sending Field PICTURE	Sending Field Value	Edited Result
9(3)V99	123.45	123.45
9(5)	12345	345.00

Symbol Explanation

+ -
CR DB These symbols are used as editing sign control symbols. They represent the character position where the editing sign conrol ssymbol is placed. The symbols are mutually exclusive in any one character string. The symbol + or – must be specified either as the leftmost or rightmost character position and is counted in the size of the item. The symbol CR or DB must be specified as the rightmost character position in the character string. Each symbol represents the character position in determining the size of the data item. The symbol + indicates that a + or – character is used to represent the sign in the field. The symbols –, CR, and DB represent a negative value and space characters represent a positive or zero value.

Examples:

PICTURE IS +$99.99.

Sending Field PICTURE	Sending Field Value	Edited Result
99V99	12.34	+$12.34
S999	-123	-$23.00
9V9	1.5	+$01.50

`PICTURE IS 999.99-.`

Sending Field PICTURE	Sending Field Value	Edited Result
S9(3)V9(2)	-123.45	123.45-
S9(3)V9(2)	+123.45	123.45Δ

`PICTURE IS $9(4).9(2)CR.`

Sending Field PICTURE	Sending Field Value	Edited Result
S9(4)V9(2)	-123.45	$Ø123.45CR
S9(4)V9(2)	+123.45	$Ø123.45ΔΔ

`PICTURE IS $999.99DB.`

Sending Field PICTURE	Sending Field Value	Edited Result
S99V99	-12.34	$ØØ12.34DB
S99V99	+12.34	$ØØ12.34ΔΔ

Symbol	Explanation
$	The $ (or alternate currency symbol) represents a character position where a currency symbol will appear. The currency symbol is represented by either the dollar 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. Unless it is preceded by a + or a – symbol, the currency sign must be in the leftmost character position in the character string.

Examples:

`PICTURE IS $999.99.`

Sending Field PICTURE	Sending Field Value	Edited Result
9(3)V99	123.45	$123.45

`PICTURE IS -$9(5).99.`

Sending Field PICTURE	Sending Field Value	Edited Result
9(5)	123	Δ$ØØ123.ØØ
S9(6)	-123456	-$23456.ØØ
S9(2)V9(2)	+12.34	Δ$ØØØ12.34

Symbol Explanation

Floating
Insertion
Editing
+ - $

The + – and $ symbols are used either as fixed insertion characters or as floating insertion characters. Floating insertion editing is specified by using a string of at least two allowable floating insertion symbols in a PICTURE character string. The leftmost symbol of the floating insertion string represents the leftmost limit at which this character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this character can appear. The second leftmost floating insertion symbol represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data can replace all characters at or to the right of this limit.

Examples:

PICTURE IS ++.99.

Sending Field PICTURE	Sending Field Value	Edited Result
S99V999	+12.345	+2.34
S9V99	-1.23	-1.23
S9V99	+Ø.12	Δ+.12
S9V99	+Ø.Ø1	Δ+.Ø1

PICTURE IS -----.99.

Sending Field PICTURE	Sending Field Value	Edited Result
S9(4)V99	-12.34	ΔΔ-12.34
S9(4)V99	-Ø.Ø1	ΔΔΔΔ-.Ø1
S9(5)	-12345	-2345.ØØ
S99V99	+1.23	ΔΔΔΔ1.23

PICTURE IS $$$$.99.

Sending Field PICTURE	Sending Field Value	Edited Result
9(4)V99	1234.56	$234.56
9(4)V99	1.23	ΔΔ$1.23
S9(2)	-3	ΔΔ$3.ØØ

Any fixed insertion symbols (B,0,/,,) within or to the right of the floating insertion string are considered part of the floating insertion string.

Examples:

```
PICTURE IS ++,+++,++9.ØØ.
```

Sending Field PICTURE	Sending Field Value	Edited Result
S9(4)	−1234	ΔΔΔΔ-1,234.ØØ
S9(4)	0	ΔΔΔΔΔΔΔΔ+Ø.ØØ
S9(4)V99	+1234.56	ΔΔΔΔ+1,234.ØØ
S9(7)	+1234567	+1,234,567.ØØ

```
PICTURE IS $(2),$(3),$$9.99.
```

Sending Field PICTURE	Sending Field Value	Edited Result
9(7)V99	1234567.89	$1,234,567.89
9(7)V99	67.89	ΔΔΔΔΔΔΔ$67.89
9(7)V99	Ø.89	ΔΔΔΔΔΔΔΔ$Ø.89
9(7)V99	Ø	ΔΔΔΔΔΔΔΔ$Ø.ØØ

A second method of floating insertion editing is to represent all numeric character positions by the floating insertion symbol. When editing is performed, the result depends on the value of the data. If the value is zero, the entire data item contains spaces. If the value is nonzero, the result is the same as if the floating insertion character were used only to the left of the decimal point.

Examples:

```
PICTURE IS ----.
```

Sending Field PICTURE	Sending Field Value	Edited Result
S9(3)	+12	ΔΔ12
S9(3)	-1	ΔΔ-1
S9(3)	Ø	ΔΔΔΔ
S9(3)V99	-Ø.5	ΔΔΔΔ
S9(3)V99	-1.5	ΔΔ-1

PICTURE IS ++,+++.++.

Sending Field PICTURE	Sending Field Value	Edited Result
S9(4)V99	+123	ΔΔ+123.ØØ
S9(4)V99	-Ø.Ø1	ΔΔΔΔΔ-.Ø1
S9(4)V99	Ø	ΔΔΔΔΔΔΔΔΔ
S9(5)V99	+1234	+1,234.ØØ
S9(5)V99	+1ØØØØ	ΔΔΔΔΔΔΔΔΔ

PICTURE IS $$,$$$.$$.

Sending Field PICTURE	Sending Field Value	Edited Result
S9(4)V99	+123.45	ΔΔ$123.45
S9(4)V99	-123.45	ΔΔ$123.45
S9(4)V99	+Ø.Ø1	ΔΔΔΔΔ$.Ø1
S9(4)V99	Ø	ΔΔΔΔΔΔΔΔΔ

Symbol Explanation

Z Each Z in a character string represents the leftmost leading numeric character positions that are replaced by a space character when the content of that character position is zero. Each Z is counted in the size of the item.

Examples:

PICTURE IS Z99.99+.

Sending Field PICTURE	Sending Field Value	Edited Result
S999V99	+123.45	123.45+
S999V99	-1.2	ΔØ1.2Ø-
S999V99	Ø	ΔØØ.ØØ+

PICTURE IS $ZZ999.99.

Sending Field PICTURE	Sending Field Value	Edited Result
9(5)V99	12345.67	$12345.67
9(5)V99	1.23	$ΔΔØØ1.23
9(6)	1234	$Δ1234.ØØ
9(6)	123456	$23456.ØØ

Symbol	Explanation
*	Each asterisk (*) represents a character position where an asterisk is placed when the content of that position is zero. Each asterisk is counted in the size of the item.

Examples:

`PICTURE IS -*99.99.`

Sending Field PICTURE	Sending Field Value	Edited Result
S9(3)V99	+123.45	Δ123.45
S9(3)V99	+12.34	Δ*12.34
S9(3)V99	-1.23	-*Ø1.23

`PICTURE IS $**,***.99.`

Sending Field PICTURE	Sending Field Value	Edited Result
9(5)V99	12345.67	$12,345.67
9(5)V99	345.67	$***345.67
9(5)V99	Ø.67	$******.67
9(5)V99	Ø	$******.ØØ

Symbol	Explanation
Zero Suppression and Replacement Editing	
Z and *	The symbols Z and * are used for zero suppression. These symbols are mutually exclusive in a PICTURE character string. Zero suppression and replacement editing is specified by using a string of one or more of the allowable symbols to represent leading numeric character positions. These positions are replaced with spaces (Z) or asterisks (*) when the associated character position in the data contains a zero.

In a PICTURE character string, there are two ways to represent zero suppression and replacement editing.

One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. When editing is performed, any leading zeros in the data appearing in the same character position as a suppression symbol are replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

The other way is to represent all numeric character positions as suppression symbols in the PICTURE character string. When editing is performed and the value of the data is nonzero, 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 contains spaces. If the value is zero and the suppression symbol is *, the entire data item except the actual decimal point contains asterisks.

Examples:

```
PICTURE IS $**,***.**.
```

Sending Field PICTURE	Sending Field Value	Edited Result
S9(5)V99	12345.67	$12,345.67
S9(5)V99	1.23	$*****1.23
S9(5)V99	Ø.Ø1	$******.Ø1
S9(5)V99	Ø	*******.**

```
PICTURE IS ZZ,ZZZ.ZZ.
```

Sending Field PICTURE	Sending Field Value	Edited Result
S9(5)V99	12345.67	12,345.67
S9(5)V99	1.67	ΔΔΔΔΔ1.67
S9(5)V99	Ø.Ø1	ΔΔΔΔΔΔ.Ø1
S9(5)V99	Ø	ΔΔΔΔΔΔΔΔΔ

```
PICTURE IS ZZ,ZZZ.ZZ+.
```

Sending Field PICTURE	Sending Field Value	Edited Result
S9(5)V99	+12345.67	12,345.67+
S9(5)V99	-1.67	ΔΔΔΔΔ1.67-
S9(5)V99	+Ø.Ø1	ΔΔΔΔΔΔ.Ø1+
S9(5)V99	Ø	ΔΔΔΔΔΔΔΔΔΔ

Any fixed insertion symbol (B,0,/,,) within or to the immediate right of the string of floating zero suppression symbols is considered part of the string.

Examples:

```
PICTURE IS $**,***.**B-.
```

Sending Field PICTURE	Sending Field Value	Edited Result
S9(5)V99	-12345.67	$12,345.67Δ-
S9(5)V99	+1.67	$*****1.67ΔΔ
S9(5)V99	-Ø.Ø1	$******.Ø1Δ-
S9(5)V99	Ø	*******.****

```
PICTURE IS ZZZ,999.
```

Sending Field PICTURE	Sending Field Value	Edited Result
9(6)	123456	123,456
9(6)	1234	ΔΔ1,234
9(6)	123	ΔΔΔΔ123
9(6)	1	ΔΔΔΔØØ1

The Z * + – and $ symbols are mutually exclusive as floating replacement symbols in one PICTURE character string (for example, the PICTURE string $$$***.99 is illegal). The asterisk as zero suppression symbol and the BLANK WHEN ZERO clause must not be specified for the same entry.

Glossary

This glossary contains an alphabetically arranged collection of definitions of terms, abbreviations, acronyms, and symbols used in this document. The terms are defined in accordance with their meanings in COBOL and may not have the same meaning for other languages.

Most of the definitions are brief and do not include detailed descriptions, because they are items in support of the text.

A

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 set of letters A through Z, and the space.

alphanumeric character

Any character in the EBCDIC 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 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.

arithmetic operator

A single character or a fixed 2-character combination that belongs to the following set:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

ascending key

A key whose values determine the ordering of data (starting with the lowest value of key to the highest value of key) according to the rules for comparing data items.

assumed decimal point

A decimal point position that 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 during the execution of:

1. a READ statement for a sequentially accessed file;

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

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

B

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 records that are either continued within the block or that overlap the block. The term is synonymous with physical record.

C

called program

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

calling program

A program that 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

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 that 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

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

CMCS

See *COBOL message control system.*

COBOL character set

The complete set consisting of the following 51 characters:

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

COBOL message control system

A component of the message control system that interfaces the COBOL communication object program with the integrated communications access method.

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 compiler translates a COBOL source program 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 also *negated simple condition, combined condition,* and *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 also *simple condition* and *complex condition*.

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 that has a condition-name assigned to one or more of its values.

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; and

3. form conditions (logical connectives). See *logical operator*.

contiguous items
Items that are described by consecutive entries in the data division and have a definite hierarchic relationship to each other.

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 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.

D

data clause
A clause appearing in a data description entry in the data division that describes 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 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 formats, represents a word that cannot be subscripted, indexed, or qualified unless specifically permitted by the rules for that format.

debugging line
Any line with D in the indicator area.

debugging section
A section that contains a USE FOR DEBUGGING statement.

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

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.

delimiter
A character or a sequence of contiguous characters identifying the end of a string of characters and separating 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 whose values arrange data in order 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
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 according to specific rules. There are four divisions in a COBOL program: identification, environment, data, and procedure.

division header

A combination of words followed by a period and a space indicating the 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 nonsequential manner (see *random access*) and obtained from a file in a sequential manner (see *sequential access*), during the scope of the same OPEN statement.

E

editing character

A single character or a fixed 2-character combination belonging to the following set:

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

An 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 for that file, with the EXTEND phrase specified, and before the execution of a CLOSE statement for that file.

F

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 either of the following data division entries:

File description (FD)
Sort file description (SD)

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, 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 file description entry within the file section of the data division.

file organization

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

file section

The section of the data division that contains file description and sort file description entries together with their associated record descriptions.

format

A specific arrangement of a set of data.

G

group item

A named contiguous set of elementary or group items.

H

hexadecimal literal

A string of hexadecimal digits bounded by quotation marks and immediately preceded by an equal sign. The string may include any hexadecimal digits up to a maximum of 30 digits.

high-order end

The leftmost character of a string of characters.

I

ICAM

See *integrated communications access method.*

I-O-CONTROL

The name of an environment division paragraph in which object program requirements are specified 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.

I-O mode

The state of a file after execution of an OPEN statement for that file, with the I-O phrase specified, 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 indexes 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 SPERRY UNIVAC systems.

index

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

INDEX-AREA

The location in main storage in which index blocks are processed by MIRAM files during keyed operations on indexed files (ORGANIZATION IS INDEXED).

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 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.

indexed sequential access method

An access method that uses record keys from an index file to access files randomly or sequentially.

input file

A file opened in the input mode.

input mode

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

input/output file

A file 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 that provides information required for transmission and handling of data during execution of the object program.

input procedure

A set of statements 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 or zero unless explicitly allowed by the rules of that format.

integrated communications access method

A component of the message control system that is the communications subsystem of the OS/3 operating system.

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.

ISAM

See *indexed sequential access method.*

ISAM file

A file with ISAM organization.

ISAM organization

The file structure supporting existing indexed sequential files created by the indexed sequential access method (ISAM) data management.

K

key

A data item that identifies the location of a record, or a set of data items that 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 required when the format in which the word appears is used in a source program.

L

level indicator

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

level-number

A user-defined word that indicates the position of a data item in the hierarchical structure of a logical record or that indicates special properties of a data description entry. A level-number is expressed as a 1- or 2-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 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, AND, OR, or both 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.

M

mass storage

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

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

A communication control system that supports the processing of messages.

message count

The count of the number of complete messages existing in the designated queue of messages.

message indicators

Conceptual indications that notify the MCS that a specific condition exists (end of group, end of message, or end of segment). The message indicators are EGI (end-of-group indicator), EMI (end-of-message indicator), and ESI (end-of-segment indicator).

Within the hierarchy of EGI, EMI, and ESI, and 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.

mnemonic-name

A user-defined word that is associated in the environment division with a specified implementor-name.

N

native character set

The character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

native collating sequence

The 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 that logically follows the current record of a file.

noncontiguous items

Elementary data items, in the working-storage and linkage sections, that 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 EBCDIC 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 set of digits 0 through 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

There are two types of numeric literals: fixed-point and floating-point.

A fixed-point literal is a string of characters chosen from the following set:

0-9
+ (plus)
- (minus)
. (decimal)

Fixed-point literals must be formed according to the following rules:

1. The literal may contain 1 to 18 digits.

2. The literal may contain only one sign character. If a sign is used, it must be the leftmost character of the literal. An unsigned literal is assumed to be positive.

3. The literal may contain only one decimal point. The decimal point may appear anywhere in the literal except as the rightmost character. A decimal point designates an assumed decimal point location. (The assumed decimal point in any numeric literal or data item is where the compiler and the object program assume the decimal point to be, though no memory position is reserved for a separate decimal point character.) A literal with no decimal point is an integer.

A floating-point literal is a data item whose potential range of value is too great for fixed-point representation.

A floating-point literal must have the following format:

```
        {±} mantissa E {±} exponent
```

where:

±
: The two plus or minus signs are optional.

mantissa
: Consists of 1 to 16 digits with a required decimal point; the decimal point may appear in any position.

exponent
: Consists of the symbol E, followed by an optional sign, followed by one or two digits. (A zero exponent may be written as 0 or 00.)

The literal must contain no spaces. The exponent must appear immediately to the right of the mantissa.

The signs are the only optional characters in the format. An unsigned mantissa or exponent is assumed to be positive.

The value of the literal is the product of the mantissa and 10 raised to the power given by the exponent.

Example:

$+1.5E - 2 = 1.5 \times 10^{-2}$

The magnitude of the number represented by a floating-point literal must not exceed $.72 \times 10^{76}$.

O

OBJECT-COMPUTER

The name of an environment division paragraph that describes the complete environment in which the object program is executed.

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 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 for that file, with the OUTPUT or EXTEND phrase specified, 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.

P

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 or spaced.

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:

- Identification Division
  PROGRAM-ID.
  AUTHOR.
  INSTALLATION.
  DATE-WRITTEN.
  DATE-COMPILED.
  SECURITY.

- 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

An ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or clause.

physical record

See *block*.

prime record key

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

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 used to name a paragraph or section in the procedure division. It consists of a paragraph-name (that 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 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:

,	comma
;	semicolon
.	period
″ or ′	quotation mark
(	left parenthesis
)	right parenthesis
	space
=	equal sign

Q

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 used in a reference with another data name at a lower level in the same hierarchy.
2. A section-name used in a reference with a paragraph-name specified in that section.
3. A library-name used in a reference 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.

R

random access

An access mode in which the program-specified value of a key data item identifies the logical record 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 asociated with a particular record.

record key

A key whose contents identify a record within an ISAM file.

record-name

A user-defined word that names a record described in a record description entry in the data division.

reentrant program

A computer program that can be entered repeatedly and can be used simultaneously by more than one program, as long as its external program parameters and instructions are not modified during its execution.

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:

>	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 meanings 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.

reserved word

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

run unit

A set of one or more object programs that function at object time as a unit to provide problem solutions.

S

SAM

See *sequential access method.*

SAM file

A file with SAM organization.

SAM organization

The file structure supported by the sequential access method (SAM) data management.

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 divisions.

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:

- Environment Division
  CONFIGURATION SECTION.
  INPUT-OUTPUT SECTION.

- Data Division
  FILE SECTION.
  WORKING-STORAGE SECTION.
  LINKAGE SECTION.
  COMMUNICATION 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 that names a section in the procedure division.

segment-number

A user-defined word that classifies sections in the procedure division for purposes of segmentation. Segment-numbers may contain only the characters 0 through 9. A segment-number may be expressed either as a 1- or 2-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 method

An access method 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.

serially-reusable program

A program that lets only one user at a time access the action program. The program is not available to other users until the current user is finished with it.

shared-code program

A program that lets two or more users access an action program concurrently. Shared code programs are only partially shareable and must be COBOL action programs.

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)

slack byte

An unused character position provided by the compiler for synchronization purposes.

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, 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 that describes the computer environment in which the source program is compiled.

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:

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

special-character word

A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES

The name of the environment division paragraph in which implementor-names are related to user-specified mnemonic-names.

special registers

Compiler-generated storage areas primarily used to store information produced for use of specific COBOL features.

standard data format

The specification plan used to describe characteristics of data in a COBOL data division where characteristics or properties of data are expressed according to the appearance of the data on printed pages.

statement

A syntactically valid combination of words and symbols written in the procedure division and beginning with a verb.

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.*

subqueue

A logical hierarchical division of a queue.

subscript

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

subscripted data-name

An identifier composed of a data-name followed by one or more subscripts enclosed in parentheses.

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 used to communicate with the operating environment.

T

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 that 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 the values true and false.

U

unary operator

A plus (+) or a minus (-) sign that precedes a variable or a left parenthesis in an arithmetic expression and 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.

V

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.

W

word

A character-string of not more than 30 characters that 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 and is composed of either noncontiguous items or working-storage records, or both.

77-level-description-entry

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

Index

Term	Reference	Page
A		
ACCEPT statement	6.6.1	6—20
Access methods		
indexed	8.2.1.3	8—2
ISAM	8.2.1.4	8—2
relative	8.2.1.2	8—1
sequential	8.2.1.1	8—1
Action programs, IMS		
description	G.1	G—1
illegal COBOL verbs, clauses and sections	G.2	G—1
ADD statement		
description	6.6.2	6—25
intermediate arithmetic results	E.1.2	E—2
Alignment		
elementary item, boundaries	Table 5—10	5—41
standard rules	2.5	2—12
Alphabet-name	2.3.1.1	2—5
ALTER statement		
description	6.6.3	6—26
segmentation effects	10.6.1	10—3
SORT statement rules	6.6.33	6—90
Arithmetic expressions		
evaluation rules	6.3.2	6—7
operators	6.3.1	6—6
permissible symbol combinations	Table 6—1	6—8
Arithmetic statements		
description	6.5.4	6—19
multiple results	6.5.6	6—20

Term	Reference	Page
Ascending sequence		
INSPECT statement	6.6.18	6—47
MERGE statement	9.5.4	9—5
SORT statement, rules	6.6.33	6—90
B		
BLANK WHEN ZERO clause	5.3.3.10	5—45
BLOCK CONTAINS clause		
block size calculations, tape, card, print	Table 5—2	5—8
block size calculations, mass storage	Table 5—3	5—9
format and rules	5.3.1.1	5—6
C		
CALL statement		
action programs	G.2	G—1
description	6.6.4	6—27
CANCEL statement	6.6.5	6—28
CD clause	5.6	5—53
Cd-name		
description	2.3.1.1	2—5
input	5.6.1	5—53
output	5.6.2	5—57

Term	Reference	Page
Character sets	Appendix J	
	2.1	2—1
Class condition		
extended	F.2.3	F—2
	F.6.1	F—6
simple	6.4.1.2	6—12
CLASS-NAME clause	F.2.2	F—2
	F.5.1	F—4
CLOSE statement		
description	6.6.6	6—29
statement options	Table 6—5	6—30
COBOL		
character set	2.1	2—1
extensions	1.2.2	1—3
language structure, modular	1.2	1—1
literals	2.3.2	2—7
object program, invoking	14.4.1	14—3
PICTURE character-string	2.3.3	2—11
reserved words	Appendix I	
separators	2.2	2—4
source coding format rules	2.7	2—14
words	2.3.1	2—5
CODE-SET clause	5.3.1.7	5—17
Collating sequence, SORT statement		
MERGE statement	9.5.4	9—5
SORT statement	6.6.33	6—90
Combined and negated combined conditions		
format and rules	6.4.2.2	6—15
permissible combinations	Table 6—4	6—15
Communication		
COBOL object program, invoking	14.4.1	14—3
COBOL object program, scheduled initiation	14.4.1.1	14—3
devices, relationship to COBOL programs	14.4	14—2
environment	Fig. 14—1	14—3
message control system	14.2	14—1
messages and message segments	14.5	14—4
queues	14.6	14—5
Communication, inter-program		
CALL statement	13.3.2	13—3
CANCEL statement	13.3.3	13—4
control transfer	13.1.1	13—1
data items, accessing	13.1.2	13—2
EXIT PROGRAM statement	13.3.4	13—4
linkage records	13.2.2	13—3
noncontiguous linkage storage	13.2.1	13—2
values, initial	13.2.3	13—3

Term	Reference	Page
Communication section		
CD clause	5.6.2	5—57
DESTINATION COUNT clause	5.6.2	5—57
DESTINATION TABLE OCCURS clause	5.6.2	5—58
END-KEY clause	5.6.1	5—53
ERROR KEY clause	5.6.2	5—57
error key codes	Table 5—11	5—60
INDEXED BY clause	5.6.2	5—57
input communication description	5.6.1	5—53
MESSAGE COUNT clause	5.6.1	5—54
MESSAGE DATE clause	5.6.1	5—54
MESSAGE TIME clause	5.6.1	5—54
output communication description	5.6.2	5—57
STATUS KEY clause, input	5.6.1	5—54
STATUS KEY clause, output	5.6.2	5—57
SYMBOLIC DESTINATION clause	5.6.2	5—58
SYMBOLIC QUEUE clause	5.6.1	5—54
SYMBOLIC SUB-QUEUE clauses	5.6.1	5—54
TEXT LENGTH clause, input	5.6.1	5—54
TEXT LENGTH clause, output	5.6.2	5—57
Comparison		
index-names or index data items	6.4.1.1.3	6—11
nonnumeric operands	6.4.1.1.2	6—11
numeric operands	6.4.1.1.1	6—10
Compiler		
compilation summary listing	B.1	B—1
cross-reference listing	B.6	B—3
diagnostic listing	B.2	B—1
	C.2	C—1
diagnostic messages	Appendix C	
locator/map/cross-reference listing	B.5	B—2
object code listing	B.4	B—2
options	A.2	A—1
	Table A—1	A—2
options affecting compiler output	A.2	A—1
	A.3	A—4
options of PARAM statement	Table A—1	A—2
source listing	B.3	B—1
Complex conditions	6.4.2	6—13
COMPUTATIONAL, USAGE clause	5.3.3.5	5—33
COMPUTE statement	6.6.7	6—33
Condition-name	2.3.1.1	2—5
	2.6.5	2—14
Condition-name condition	6.4.1.3	6—12

Term	Reference	Page
Conditional expressions		
abbreviated combined relation conditions	6.4.2.3	6—16
class condition	6.4.1.2	6—12
combined and negated combined conditions	6.4.2.2	6—15
complex conditions	6.4.2	6—13
condition-name condition	6.4.1.3	6—12
evaluation rules	6.4.3	6—17
negated simple conditions	6.4.2.1	6—14
permissible comparisons	Table 6—2	6—9
relation condition	6.4.1.1	6—9
sign condition	6.4.1.5	6—13
simple conditions	6.4.1	6—8
switch-status condition	6.4.1.4	6—13
Conditions		
AT END	8.2.4	8—5
evaluation rules	6.4.3	6—17
INVALID KEY	8.2.5	8—5
Configuration section, format	4.3	4—2
CONNECT-FREE phrase	4.3.3	4—5
	Table 4—2	4—9
Constants, figurative	2.3.2	2—7
Continuation, lines	2.7	2—16
Control statement considerations, linkage editor	10.5.4	10—4
COPY statement	6.6.8	6—33
CORRESPONDING phrase	6.5.3	6—18
Currency symbol	5.3.3.4	5—23
Current record pointer	8.2.2	8—3

Term	Reference	Page
D		
Data, classes	2.4	2—11
Data description		
entry clauses	Table 5—5	5—20
format and rules	5.3.3	5—18
DATA division		
communication section	5.6	5—53
file description, format and rules	5.3.1	5—4
file section	5.3	5—4
format	5.2.1	5—1
indexed file processing	8.5.2.2	8—12
inter-program communication	13.2	13—2
ISAM file processing	8.7.2	8—16
level-numbers	5.2.2.3	5—3
linkage section	5.5	5—51
relative file processing	8.4.2.2	8—9
SAM file processing	8.6.2	8—14
sequential file processing	8.3.2.2	8—6
sort programs	9.4	9—3
working storage section	5.4	5—49
Data items, class and category, elementary and group	Table 5—5	5—20
Data-name	2.3.1.1	2—5
DATA RECORDS clause	5.3.1.5	5—12
DATE	6.6.1	6—21
DAY	6.6.1	6—21
DEBUG statement	6.6.43	6—119
Debugging language		
compile-time switch	12.2.2	12—2
conditions and contents of DEBUG-item	Table 12—1	12—5
DEBUG-ITEM register	12.2.1	12—2
*DEBUG packet	12.4.3.4	12—4
EXHIBIT statement	12.4.3.2	12—4
extended debugging facility	12.4.3	12—3
object-time switch	12.2.3	12—2
ON statement	12.4.3.1	12—3
USE FOR DEBUGGING statement	12.4.1	12—3
WITH DEBUGGING MODE clause	12.3.1	12—2
Declaratives	6.1.1	6—1
DELETE statement	6.6.9	6—36
Descending sequence		
INSPECT statement	6.6.18	6—47
MERGE statement	9.5.4	9—5
SORT statement, rules	6.6.33	6—90

Term	Reference	Page
Description entry		
record, linkage section	5.5.2	5—53
record, working storage	5.4.2	5—51
77-level, linkage section	5.5.1	5—52
77-level, working storage	5.4.1	5—50
DESTINATION COUNT clause	5.6.2	5—57
DESTINATION TABLE OCCURS clause	5.6.2	5—57
DISABLE statement	6.6.10	6—36
DISPLAY statement	6.6.11	6—38
DIVIDE statement		
description	6.6.12	6—41
intermediate arithmetic results	E.1.4	E—2
Dynamic access mode		
description	8.2.1.3	8—2
indexed files, environment division	8.5.2.1	8—11
READ statement, rules	6.6.25	6—72
REWRITE statement, rules	6.6.29	6—79

E

Editing, sign control symbol results	Table 5—7	5—29
EGI	6.6.31	6—85
EMI	6.6.31	6—85
ENABLE statement	6.6.13	6—43
End-of-group indicator	6.6.31	6—85
End-of-message indicator	6.6.31	6—85
End-of-segment indicator	6.6.31	6—85

Term	Reference	Page
Environment division		
configuration section	4.3	4—2
debugging language	12.3	12—2
FILE-CONTROL paragraph	4.4.1	4—10
indexed file processing	8.5.2.1	8—11
input-output section	4.4	4—10
I-O-CONTROL paragraph	4.4.2	4—18
ISAM file processing	8.7.1	8—15
non-English language feature	Appendix F	
OBJECT-COMPUTER paragraph	4.3.2	4—3
relative file processing	8.4.2.1	8—8
sequential file processing	8.3.2.1	8—6
sort programs	9.3	9—2
SOURCE-COMPUTER paragraph	4.3.1	4—3
SPECIAL-NAMES paragraph	4.3.3	4—4
Error codes	Table 5—12	5—61
ERROR KEY clause	5.6.2	5—57
ESI	6.6.31	6—83
EXHIBIT statement	6.6.14	6—44
EXIT statement	6.6.15	6—44
Extensions		
Series 90 environment	1.2.2	1—3
VS/9 COBOL	1.2.2	1—3

F

Federal information processing standard flagging facility		
flagging options	D.2	D—2
	Table D—1	D—1
PUB 21-1 COBOL levels	D.1	D—1
Figurative constants		
description	2.3.2	2—7
TRANSFORM statement, rules	6.6.39	6—101

Term	Reference	Page
FILE-CONTROL paragraph		
description	4.4.1	4—15
Series 90 SAM files	4.4.1	4—17
Series 90 ISAM files	4.4.1	4—17
File-name	2.3.1.1	2—5
File organizations		
indexed	8.2.1.3	8—1
ISAM	8.2.1.5	8—2
relative	8.2.1.2	8—1
SAM	8.2.1.4	8—2
sequential	8.2.1.1	8—1
File section		
data description	5.3.3	5—18
file description	5.3.1	5—4
sort/merge file description	5.3.2	5—17
FILLER clause	5.3.3.2	5—21
FIPS	Appendix D	
Floating point		
items	5.3.3.4	5—23
numeric literals	2.3.2	2—7
operands	E.1.1	E—2
FUNCTION-KEYS phrase	4.3.3	4—5
	Table 4—3	4—9

G

Term	Reference	Page
GO TO statement		
description	6.6.16	6—44
SORT statement, rules	6.6.33	6—90

I

Term	Reference	Page
Identification division	Section 3	
Identifier	2.6.4	2—14
IF statement	6.6.17	6—46
IMS parameter and configuration specifications	G.3	G—4
Index-name		
comparison	6.4.1.1.3	6—11
SET statement use	6.6.32	6—88
user-defined word	2.3.1.1	2—5
INDEXED BY clause	5.6.2	5—57
Indexed file processing	8.5	8—11
Indexed I-O module	1.2	1—1
Indexing	2.6.3	2—13
Indexed organization		
Series 90, SAM	8.2.1.4	8—2
Series 90, ISAM	8.2.1.5	8—3
Input-output section	4.4	4—15
Insertion editing, types and characters	5.3.3.4	5—25
	Table 5—6	5—24
INSPECT statement	6.6.18	6—47
Intermediate arithmetic results, object program processing	E.1	E—1
I-O CONTROL paragraph		
environment division	4.4.2	4—23
sort program	9.3.2	9—3
I-O status	8.2.3	8—3
ISAM file processing	8.6	8—13

Term	Reference	Page
J		
Job control stream requirements	Appendix H	
JUSTIFIED clause	5.3.3.9	5—44
K		
Key		
INVALID KEY condition	8.2.5	8—5
INVALID KEY phrase, DELETE statement	6.6.9	6—36
READ statement, INVALID KEY phrase	6.6.25	6—72
RECORD KEY phrase	6.6.9	6—36
REWRITE clause, INVALID KEY phrase	6.6.29	6—79
START statement, KEY/INVALID KEY phrase	6.6.34	6—90
WITH KEY phrase, DISABLE statement	6.6.10	6—37
WITH KEY phrase, ENABLE statement	6.6.13	6—43
WRITE statement, INVALID KEY phrase	6.6.42	6—114
L		
LABEL RECORDS clause	5.3.1.5	5—12
Level characteristics		
indexed files	8.5.1	8—11
inter-program communication	13.1.3	13—2
relative files	8.4.1	8—8
sequential files	8.3.1	8—5
Level-indicators	5.2.2.1	5—2
Level-numbers		
regular	2.3.1.1	2—5
	5.2.2.2	5—2
	5.3.3.1	5—21
special	5.2.2.3	5—3
Library, COPY statement	11.2	11—1
Library-name	2.3.1.1	2—5
LINAGE clause	5.3.1.6	5—13
LINAGE-COUNTER special register	8.2.6	8—5
Linkage editor control statements	10.5.4	10—4

Term	Reference	Page
Linkage section		
record description entry	5.5.2	5—53
77-level description entry	5.5.1	5—52
Literals, rules	2.3.2	2—7
LOCK phrase		
CLOSE, use	6.6.6	6—29
OPEN, use	6.6.23	6—60
Logical devices		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
M		
MCS	See message control system.	
MERGE verb		
description	6.6.19	6—53
segmentation effects	10.6.3	10—5
sort programs	9.5.4	9—5
Message control system		
COBOL object program	14.3	14—2
description	14.2	14—1
generation	14.7	14—7
invocation	14.4.1.2	14—4
relationship to COBOL programs	14.4	14—2
MESSAGE COUNT clause		
ACCEPT statement	6.6.1	6—20
description	5.6.1	5—53
MESSAGE DATE clause	5.6.1	5—53
MESSAGE TIME clause	5.6.1	5—53
Messages		
compiler diagnostic	Appendix C	
complete and segmented, handling	14.5	14—4
Mnemonic-name	2.3.1.1	2—5
MOVE statement		
description	6.6.20	6—56
permissible moves	Table 6—6	6—58
MULTIPLY statement		
description	6.6.21	6—59
intermediate arithmetic results	E.1.3	E—2
Multivolume sequential files	8.3.4	8—8
Multivolume SAM files	8.6.4	8—15

N

Term	Reference	Page
Naming convention, object module	10.5.3	10—3
Non-English language feature		
control division	F.4	F—3
description	F.1	F—1
environment division	F.5	F—4
format	F.3	F—2
procedure division	F.6	F—6
text utility program	F.7	F—6
Nonnumeric literals, TRANSFORM statement, rules	6.6.39	6—101
Nonnumeric operands, comparison	6.4.1.1.2	6—11
Numeric operands, comparison	6.4.1.1.1	6—10

O

Term	Reference	Page
OBJECT-COMPUTER paragraph	4.3.2	4—3
Object module naming convention	10.5.3	10—3
Object program, COBOL		
description	14.3	14—2
execution considerations	13.4	13—4
expressions	E.2	E—3
intermediate results, arithmetic operations	E.1	E—1
invoking	14.4.1	14—3
processing	Appendix E	
scheduled initiation	14.4.1.1	14—3
OCCURS clause	5.3.3.7	5—38
ON statement	6.6.22	6—60
OPEN statement		
description	6.6.23	6—60
permissible I/O statements, OPEN mode	Table 6—7	6—62
Operands, overlapping	6.5.5	6—20

P

Term	Reference	Page
Page, logical		
format	Fig. 5—2	5—15
size	5.3.1.6	5—13
spacing, SEND statement	6.6.31	6—85
Page spacing		
SEND statement	6.6.31	6—85
WRITE statement	6.6.42	6—114
Paragraph-name	2.3.1.1	2—6
PARAM statement, options	Table A—1	A—2
PERFORM statement		
description	6.6.24	6—64
segmentation effects	10.6.2	10—4
SORT statement, rules	6.6.33	6—90
PICTURE clause		
character precedence chart	Table 5-9	5-32a
function and format	5.3.3.4	5-23
tutorial guide	Appendix K	
Pointer, current record	8.2.2	8—3
Printer-destined files		
description	8.3.3	8—8
READ statement, rules	6.6.25	6—72
Procedure division		
action programs	G.2	G—1
arithmetic expressions	6.3	6—6
COBOL verbs	6.6	6—20
conditional expressions	6.4	6—8
debugging language	12.4	12—3
format	6.1.3.2	6—3
indexed file processing	8.5.2.3	8—12
inter-program communication	13.3	13—3
ISAM file processing	8.7	8—15
non-English language feature	F.6	F—6
relative file processing	8.4.2.3	8—10
sequential file processing	8.3.2.3	8—7
sort programs	9.5	9—4
Program-name	2.3.1.1	2—5
Pseudo-text delimiters	2.2	2—4

Term	Reference	Page
Q		
Qualification	2.6.1	2—12
Queues, message		
description	14.6	14—5
enqueueing/dequeueing	14.6.2	14—5
hierarchy	14.6.3	14—5
logical disconnectives, enabling/disabling	14.6.1	14—5
R		
Range checking	7.2.2.3	7—4
READ statement	6.6.25	6—72
RECEIVE statement	6.6.26	6—76
RECORD CONTAINS clause	5.3.1.2	5—10
Record-name	2.3.1.1	2—5
REDEFINES clause	5.3.3.3	5—22
REENTRANT action program	G.2	G—1
Relation condition		
abbreviated combined	6.4.2.3	6—16
description	6.4.1.1	6—9
Relative file processing	8.4	8—8
RELEASE statement	6.6.27 9.5.1	6—77 9—4
RENAMES clause	5.3.3.12	5—48
Reserved words, COBOL	Appendix I	
RETURN statement	6.6.28 9.5.2	6—78 9—4
REWRITE statement	6.6.29	6—79
ROUNDED phrase		
ADD statement	6.6.2	6—24
description	6.5.1	6—17
DIVIDE statement	6.6.12	6—41
MULTIPLY statement	6.6.21	6—59
SUBTRACT statement	6.6.37	6—99

Term	Reference	Page
S		
SEARCH statement		
description	6.6.30	6—81
with two WHEN phrases	Fig. 6—5	6—83
Section-name	2.3.1.1	2—5
SEGMENT-LIMIT clause	10.5.2	10—3
Segment-number	2.3.1.1 6.1.3.2	2—5 6—3
Segmentation		
ALTER statement	10.6.1	10—4
classification	10.3	10—2
control	10.4	10—3
fixed portion	10.2.1	10—1
independent segments	10.2.2	10—2
MERGE statement	10.6.4	10—5
PERFORM statement	10.6.2	10—5
program structure	10.5	10—3
SORT statement	10.6.3	10—5
SEND statement	6.6.31	6—85
Separators, rules	2.2	2—4
Sequential file processing	8.3	8—5
SET statement	6.6.32 7.4.3	6—88 7—4
Simple conditions, negated	6.4.2.1	6—14
SIGN clause	5.3.3.6	5—36
Sign condition	6.4.1.5	6—13
SIZE ERROR phrase		
ADD statement	6.6.2	6—25
DIVIDE statement	6.6.12	6—41
general description	6.5.2	6—18
MULTIPLY statement	6.6.21	6—59
SUBTRACT statement	6.6.37	6—99
Sort/merge		
file description	5.3.2	5—17
module	1.2	1—1

Term	Reference	Page
SORT programs		
control entry	9.3.1	9—3
I-O CONTROL paragraph	9.3.2	9—3
MERGE statement	9.5.4	9—5
RELEASE statement	9.5.1	9—4
RETURN statement	9.5.2	9—4
SORT statement	9.5.3	9—4
special registers	9.2.2	9—2
SORT statement		
description	6.6.33	6—90
segmentation effects	10.6.3	10—4
SORT programs	9.5.3	9—4
SOURCE-COMPUTER paragraph	4.3.1	4—3
SPECIAL-NAMES paragraph		
action programs	G.2	G—1
environment division	4.3.3	4—4
Special registers, sort	9.2.2	9—2
START statement	6.6.34	6—94
Statements		
compiler-directing	6.2.3	6—6
conditional	6.2.2	6—5
continuation	2.7	2—16
imperative	6.2.1	6—4
Status key		
clause	5.6.1	5—53
	5.6.2	5—57
conditions	5.6.2	5—57
values and meanings	Table 8—1	8—4
workstations	Table 4—1	4—8
STOP statement	6.6.35	6—96
STRING statement	6.6.36	6—97
Subscripting	2.6.2	2—13
SUBTRACT statement		
description	6.6.37	6—99
intermediate arithmetic results	E.1.2	E—2
Switch-status condition	6.4.1.4	6—13
SYMBOLIC DESTINATION clause	5.6.2	5—57

Term	Reference	Page
SYMBOLIC QUEUE clause	5.6.1	5—53
SYMBOLIC SUB-QUEUE clause	5.6.1	5—53
SYNCHRONIZED clause	5.3.3.8	5—40
SYSCHAN-n	4.3.3	4—4
SYSCOM		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SYSCONSOLE		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SYSFORMAT		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SPECIAL NAMES paragraph	4.3.3	4—4
SYSIN		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SYSLOG	6.6.11	6—38
SYSOUT	6.6.11	6—38
SYSSWCH		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SYSSWCH-n		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SYSTEM-SHUTDOWN	4.3.3	4-6
SYSTERMINAL		
ACCEPT statement	6.6.1	6—20
DISPLAY statement	6.6.11	6—38
SPECIAL NAMES paragraph	4.3.3	4—4
SYSWORK		
ACCEPT statement	6.6.1	6—22
DISPLAY statement	6.6.11	6—38
SPECIAL NAMES paragraph	4.3.3	4—6

Term	Reference	Page
T		
Table handling		
comparisons, index-name or data item	7.4.2	7—5
data division	7.3	7—4
definition	7.2.1	7—1
indexing	7.2.2.2	7—3
procedure division	7.4	7—4
referencing items	7.2.2	7—2
SET statement, overlapping operands	7.4.3	7—5
statements	7.4.1	7—4
subscripting	7.2.2.1	7—3
Tallying and/or replacing		
INSPECT statement	6.6.18	6—47
UNSTRING statement	6.6.40	6—104
TEXT-LENGTH clause		
input	5.6.1	5—53
output	5.6.2	5—57
Text-name	2.3.1.1	2—6
TIME	6.6.1	6—21
TRACE statement	6.6.38	6—100
TRANSFORM statement		
description	6.6.39	6—101
permissible FROM/TO options	Table 6—9	6—102
Truncation	5.3.3.4	5—23
U		
UNSTRING statement	6.6.40	6—104
USAGE clause	5.3.3.5	5—33
USE statement	6.6.41	6—106

Term	Reference	Page
V		
VALUE clause	5.3.3.11	5—45
VALUE OF clause	5.3.1.4	5—12
VARYING phrase		
one condition, PERFORM statement	Fig. 6—2	6—67
SEARCH statement	6.6.30	6—81
three conditions, PERFORM statement	Fig. 6—4	6—70
two conditions, PERFORM statement	Fig. 6—3	6—69
Verbs, COBOL		
ACCEPT	6.6.1	6—20
ADD	6.6.2	6—25
ALTER	6.6.3	6—26
CALL	6.6.4	6—27
CANCEL	6.6.5	6—28
CLOSE	6.6.6	6—29
COMPUTE	6.6.7	6—33
COPY	6.6.8	6—33
DELETE	6.6.9	6—36
DISABLE	6.6.10	6—37
DISPLAY	6.6.11	6—38
DIVIDE	6.6.12	6—41
ENABLE	6.6.13	6—43
EXHIBIT	6.6.14	6—44
EXIT	6.6.15	6—44
GO TO	6.6.16	6—45
IF	6.6.17	6—46
INSPECT	6.6.18	6—47
MERGE	6.6.19	6—53
MOVE	6.6.20	6—56
MULTIPLY	6.6.21	6—59
ON	6.6.22	6—60
OPEN	6.6.23	6—60
PERFORM	6.6.24	6—64
READ	6.6.25	6—72
RECEIVE	6.6.26	6—76
RELEASE	6.6.27	6—77
RETURN	6.6.28	6—78
REWRITE	6.6.29	6—79
SEARCH	6.6.30	6—81
SEND	6.6.31	6—85
SET	6.6.32	6—88
SORT	6.6.33	6—90
START	6.6.34	6—94
STOP	6.6.35	6—96
STRING	6.6.36	6—97
SUBTRACT	6.6.37	6—99
TRACE	6.6.38	6—100
TRANSFORM	6.6.39	6—101
UNSTRING	6.6.40	6—103
USE	6.6.41	6—106
WRITE	6.6.42	6—114
DEBUG	6.6.43	6—119

Term	Reference	Page
W		
WHEN COMPILED special register	6.5.7	6—20
Words, COBOL		
reserved	2.3.1.3	2—6
system names	2.3.1.2	2—6
user-defined	2.3.1.1	2—5
Working storage section		
record description entry	5.4.2	5—51
77-level description entry	5.4.1	5—50
Workstation status key values	Table 4—1	4—8
WRITE statement	6.6.42	6—114

Term	Reference	Page
Z		
Zero suppression, editing	5.3.3.4	5—25

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD

CUT

SPERRY

USER COMMENT SHEET

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN.: SOFTWARE SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT

SPERRY

USER COMMENT SHEET

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

____

(Document Title)

_______________ _______________ _______________

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN.: SOFTWARE SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

SPERRY

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

____

(Document Title)

________________	________________	________________
(Document No.)	*(Revision No.)*	*(Update Level)*

Comments:

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN: Documentation Quality Control Group
C/O SYSTEM PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19422-9990

FOLD

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update Level)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

Unisys Corporation
E/MSG Product Information Development
PO Box 500 C1-NE6
Blue Bell, PA 19422-9990

FOLD

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

____
(Document Title)

_______________ _______________ _______________
(Document No.) *(Revision No.)* *(Update Level)*

Comments:

From:

____
(Name of User)

____
(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

Unisys Corporation
E/MSG Product Information Development
PO Box 500 — E5-114
Blue Bell, PA 19422-9990

FOLD