GC26-3857-3
File No. S370-24

Systems

IBM VS COBOL for OS/VS

**Program Numbers 5740-CB1 (Compiler and Library)
5740-LM1 (Library Only)**

Release 2.4

Fourth Edition (August 1983)

This is a major revision of, and makes obsolete, GC26-3857-2.

This edition applies to Release 2.4 of OS/VS COBOL, Program Product 5740-CB1 (Compiler and Library) and 5740-LM1 (Library), and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the preface ("About This Book"). Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent republication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest <u>IBM System/370 and 4300 Processors Bibliography</u>, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

ABOUT THIS BOOK

This manual describes IBM OS/VS COBOL; it gives the rules for writing COBOL source programs that are to be compiled by the OS/VS COBOL compiler. It is meant to be used as a reference manual in writing OS/VS COBOL programs, and in conjunction with the IBM OS/VS COBOL Compiler and Library Programmer's Guide, SC28-6483 and the IBM OS/VS COBOL Language Reference Summary, GX26-3720.

COBOL (COmmon Business Oriented Language) is a programming language similar to English that is used for commercial data processing. It is developed by the Conference On DAta SYstems Languages (CODASYL). The standard of the language in the USA, approved by the American National Standards Institute (ANSI), is American National Standard COBOL, X3.23-1974.

The OS/VS COBOL Compiler and Library, Release 2, are designed according to the specifications of the following industry standards, as understood and interpreted by IBM as of April 1976:

- The highest level of American National Standard COBOL, X3.23-1974 (excepting the Report Writer module). American National Standard COBOL, X3.23-1974, is compatible with and identical to International Organization for Standardization/Draft INternational Standard (ISO/DIS) 1989-COBOL. (In this manual a reference to the *1974 Standard* or to *1974 Standard COBOL* is a reference to these two standards.)

  Portions of this manual are copied from *American National Standard COBOL*, X3.23-1974. This material is reproduced with permission from *American National Standard Programming Language COBOL*, X3.23-1974, copyright 1974 by the American National Standards Institute, copies of which may be purchased from the American National Standards Institute at 1430 Broadway, New York, New York, 10018.

- The highest level of American National Standard COBOL, X3.23-1968 (including the Report Writer module). American National Standard (ANS) COBOL, X3.23-1968, is identical to ISO 1989-1972. (In this manual a reference to the *1968 Standard* or to *1968 Standard COBOL* is a reference to these two standards.)

IBM EXTENSIONS

A significant number of IBM extensions are also included. The most important are:

Report Writer
PASSWORD Clause
TRANSFORM Statement
ENTRY Statement

Three types of extensions are included:

- Those that represent processing capabilities not included in the 1974 standard.

- Those that represent language from American National Standard COBOL, X3.23-1968, that is not included in the 1974 standard.

- Those that ease the 1974 standard rules for greater ease in programming.

For the convenience of users who wish to refer to the 1974 standard, all extensions in this manual are flagged.

MANUAL ORGANIZATION

The manual is organized for reference purposes as follows.

The General Description section describes OS/VS COBOL language in general terms, and also describes non-language features available with the compiler.

The main sections of the book give the specific rules for writing OS/VS COBOL source programs. There is a separate part for Language Considerations, for each of the COBOL Divisions, and for Special Features. For easier reference, System Dependencies are grouped together in a separate chapter.

Appendixes provide supplemental information:

- Appendix A describes briefly the language elements available through IBM OS Full American National Standard COBOL that are continued in OS/VS COBOL for compatibility purposes.
- Appendix B gives ASCII file processing considerations.
- Appendix C describes System/370 unit record processing.
- Appendix D describes intermediate results.
- Appendix E contains several direct access storage file processing programs.
- Appendix F is a list of COBOL reserved words.
- Appendix G gives both the EBCDIC and ASCII collating sequences.
- Appendix H is a list of statements flagged when the MIGR compiler option is specified.

The Glossary gives definitions of COBOL terms.

RELATED PUBLICATIONS

A knowledge of basic data processing techniques is required for the understanding of this manual. Such information can be found in the following publications:

- <u>Introduction to IBM Data Processing Systems</u>, GC20-1684
- <u>Introduction to IBM Direct Access Storage Devices and Organization Methods</u>, GC20-1649

The reader should also have a general knowledge of COBOL before using this manual. Useful background information can be found in the following publications:

- <u>American National Standard COBOL Coding</u>:

  <u>Card and Tape Applications Text</u>, SR29-0283

  <u>Coding Techniques and Disk Applications Text</u>, SR29-0284

  <u>Illustrations</u>, SR29-0285

  <u>Student Reference Guide</u>, SR29-0286

- <u>IBM OS COBOL Interactive Debug and (TSO) COBOL Prompter, General Information</u>, GC28-6454.

If information in these background publications conflicts with information given in this manual, the information in this manual must be considered correct in the writing of OS/VS COBOL programs. Any violation of the rules defined in this manual for using either the Operating System or the OS/VS COBOL compiler is considered an error.

Information on the 3886 OCR can be found in the following publications:

- OS/VS Program Planning Guide for IBM 3886 Optical Character Reader Model 1, GC21-5069

- IBM 3886 Optical Character Reader General Information Manual, GA21-9146

- IBM 3886 Optical Character Reader Input Document Design Guide and Specifications, GA21-9148

Parameters for COBOL DD statements are given in the publication:

- IBM System/360 Planning Guide for IBM 3505 Card Reader and IBM 3525 Card Punch on System/370, GC21-5027

A general knowledge of the IBM Operating System is desirable, but not required. The following publication gives such information:

- IBM System/370 System Summary, GA22-7001

ACKNOWLEDGMENT

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

> Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgment of the source, but need not quote this entire section.
>
> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.
>
> Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.
>
> The authors and copyright holders of the copyrighted material used herein
>
> > FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell
>
> have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such

authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

SUMMARY OF AMENDMENTS

RELEASE 2.4, AUGUST 1983

NEW PROGRAMMING FEATURE

The MIGR compiler option flags major COBOL language elements that are no longer supported or are supported differently by the VS COBOL II compiler, Program Number 5668-958. Appendix H, added to this book, lists the statements that are flagged when you specify MIGR.

SERVICE CHANGES

A variety of corrections and clarifications have been made throughout the text, along with a number of changes made to reflect APAR fixes.

DECEMBER, 1981

SERVICE CHANGES

This documentation for the VS COBOL for OS/VS product is revised as indicated. These are documentation changes only, and do not reflect changes to the program product itself.

Topic	Documentation Change	Page
BLOCK CONTAINS Clause	Description modified to include clarification of when the clause can or should be omitted	56
BLOCK or RECORD CONTAINS 0 clauses	Cannot be specified with SAME AREA or SAME RECORD AREA clause	351
CALL statement	USING option	357
COPY statement	Clarification of REPLACING option	292
DIVIDE statement	Clarification of identifiers following the key words GIVING and REMAINDER	166
EXIT statement	Programming Notes added	203
IF Statement	Programming Note deleted	119
INSPECT statement	For signed numeric items in an INSPECT statement, if the sign is a separate character, the byte containing the sign is not examined	173
LINAGE clause	Not allowed with the REPORT clause	60
MERGE statement	Maximum number of files that can be merged is noted	236
NOTE statement	Description added	404

Topic	Documentation Change	Page
OCCURS clause	Maximum table length specified	215
ON statement	ON statement compiler-generated counter resets when it reaches 16,777,215	400
PICTURE Clause	Maximum symbols in character-string noted	72
READ statement	IBM Extension allows use of records of various sizes with READ INTO	146
ROUNDED option	IBM Extension to the ROUNDED option Programming note about fractional exponents	161, 168
Subscripting and Indexing	Restrictions on subscripting and indexing expanded	213
SORT INPUT PROCEDURE option	Additional restriction specified	240
SORT OUTPUT PROCEDURE option	Additional restriction specified	241
TERMINATE statement	Warning about control variables in TERMINATE statement	278
TRANSFORM statement	Identifier-3 in a TRANSFORM statement can be an external decimal	189
UNSTRING statement	COUNT IN option clarified Programming note covers when the sending field is also the receiving field	183, 189
USAGE IS INDEX clause	IBM Extension allows use of SYNCHRONIZED when USAGE IS INDEX	397
WRITE statement	IBM Extension to mnemonic-name option	140

- The format summary in Appendix F has been deleted.

  Appendix F now contains a list of COBOL reserved words.

  Language format reference information is now included in a reference summary, IBM VS COBOL for OS/VS Reference Summary—Format, Status Key Values, and Reserved Words, GX26-3720.

- Other minor editorial changes have also been made throughout the manual.

CONTENTS

General Description 1
Language Level 1
Compiler Features 3
  Features Dependent on TSO 4
Format Notation 5
  IBM Extensions 7

Part 1. Language Considerations 9

COBOL Program Structure 10
The COBOL Divisions 10
    Identification Division 10
    Environment Division 10
    Data Division 10
    Procedure Division 10
Clauses and Statements 11
  Clause and Statement Specification Order 11

Structure of the Language 12
Character-Strings 13
  COBOL Words 13
    User-Defined Words 14
    System-Names 14
    Reserved Words 15
  Literals 16
    Nonnumeric Literals 16
    Numeric Literals 16
  Figurative Constants 17
  PICTURE Character-Strings 18
  Comments 18
Separators 18
  Overall Punctuation Rules 19
    Identification Division 19
    Environment Division 20
    Data Division 20
    Procedure Division 20

Standard COBOL Format 21
    Sequence Numbers 21
    Continuation Area 21
    Area A and Area B 22
    Continuation of Lines 23
    Comment Lines 24
    Blank Lines 24

Methods of Data Reference 25
Qualification 25
    Qualification Rules 26
Subscripting and Indexing 27
Explicit and Implicit References 27
  Data Attribute Specification 27
  Procedure Division Data References 27
  Transfers of Control 27

Part 2. Identification and Environment Divisions 29

Identification Division 30
PROGRAM-ID Paragraph 31
DATE-COMPILED Paragraph 31
Other Optional Paragraphs 31

Environment Division—Configuration Section 32
SOURCE-COMPUTER Paragraph 33
OBJECT-COMPUTER Paragraph 34
SPECIAL-NAMES Paragraph 34
  function-name-1 34
  function-name-2 35
  Alphabet-Name Clause 35

    - Alphabet-Name Clause Examples . . . 36
  - CURRENCY SIGN Clause . . . 37
  - DECIMAL POINT IS COMMA Clause . . . 37
  - Programming Notes . . . 37

Environment Division—Input-Output Section . . . 39

- File Processing Summary . . . 39
  - Data Organization . . . 39
    - VSAM or Physical Sequential Organization . . . 39
    - VSAM Indexed Organization . . . 40
    - VSAM Relative Organization . . . 40
  - Access Modes . . . 40
    - Sequential Files . . . 40
    - VSAM Indexed Files . . . 41
    - VSAM Relative Files . . . 41
- FILE-CONTROL Paragraph . . . 41
  - SELECT Clause . . . 43
    - Format 1 Considerations . . . 43
  - ASSIGN Clause . . . 43
  - RESERVE Clause . . . 44
  - ORGANIZATION Clause . . . 44
    - Format 1 Considerations . . . 44
    - Format 2 Considerations . . . 44
    - Format 3 Considerations . . . 44
  - ACCESS MODE Clause . . . 44
    - Format 1 Considerations . . . 44
    - Format 2 Considerations . . . 44
    - Format 3 Considerations . . . 45
  - PASSWORD Clause . . . 45
    - Format 1 Considerations . . . 45
    - Format 2 Considerations . . . 45
  - FILE STATUS Clause . . . 45
  - RECORD KEY Clause (Format 2) . . . 46
  - ALTERNATE RECORD KEY Clause (Format 2) . . . 46
  - RELATIVE KEY Clause (Format 3) . . . 47
- I-O-CONTROL PARAGRAPH . . . 47
  - RERUN Clause . . . 48
    - Format 1 Considerations . . . 48
    - Format 2 Considerations . . . 49
  - SAME Clause . . . 49
    - Programming Notes . . . 50
  - MULTIPLE FILE TAPE Clause . . . 50

Part 3. Data Division . . . 51

Data Division Concepts . . . 52

- External Data . . . 52
- Internal Data . . . 52
- Data Relationships . . . 53
- Data Division Organization . . . 53
  - File Section . . . 54
    - File Description Entries . . . 54
    - Record Description Entry . . . 54
  - Working-Storage Section . . . 54
    - Record Description Entries . . . 54
    - Data Item Description Entry . . . 54
  - Linkage Section . . . 55
  - Communication Section . . . 55
  - Report Section . . . 55

File Description Entry . . . 56

  - file-name . . . 58
  - BLOCK CONTAINS Clause . . . 58
    - CHARACTERS Option . . . 58
    - RECORDS Option . . . 59
  - RECORD CONTAINS Clause . . . 59
    - Programming Notes . . . 59
  - LABEL RECORDS Clause . . . 59
    - STANDARD Option . . . 59
    - OMITTED Option . . . 59
    - VSAM Considerations . . . 60
  - VALUE OF Clause . . . 60
  - DATA RECORDS Clause . . . 60
  - LINAGE Clause . . . 60

LINAGE integer-5/data-name-5 60
WITH FOOTING Option 60
LINES AT TOP Option 61
LINES AT BOTTOM Option 61
General Considerations 61
LINAGE-COUNTER Special Register 62
REPORT Clause 62
CODE-SET Clause 62

Data Description 64
Data Description Concepts 64
Level Concepts 64
Level-Numbers 64
Record Description Level-Numbers 64
Special Level-Numbers 66
Indentation 66
Classes of Data 66
Standard Alignment Rules 66
Standard Data Format 67
Character-String and Item Size 67
Signed Data 68
Operational Signs 68
Editing Signs 68
Data Description Entry 68
General Format 1 69
General Format 2 69
General Format 3 69
level-number 70
data-name/FILLER Clause 71
REDEFINES Clause 71
General Considerations 72
BLANK WHEN ZERO Clause 74
JUSTIFIED Clause 74
OCCURS Clause 74
PICTURE Clause 74
Symbols Used in the PICTURE Clause 75
Character-String Representation 79
Data Categories and PICTURE Considerations 80
PICTURE Clause Editing 82
SIGN Clause 87
SYNCHRONIZED Clause 88
Slack Bytes 89
Intrarecord Slack Bytes 89
Interrecord Slack Bytes 92
USAGE Clause 94
DISPLAY Option 94
Computational Options 95
VALUE Clause 97
General Considerations 98
Format 1 Considerations 99
Format 2 Considerations 99
RENAMES Clause 100
data-name-2 Option 101
data-name-2 THRU data-name-3 Option 101

Part 4. Procedure Division 103

Procedure Division Organization 104
Declaratives 104
Procedures 104
Procedure Division Structure 105
Categories of Statements 105
Categories of Sentences 108

Arithmetic Expressions 109
Arithmetic Operators 109

Conditional Expressions 111
Simple Conditions 111
Class Condition 111
Condition-Name Condition 112
Relation Condition 113
Comparison of Numeric Operands 114
Comparison of Nonnumeric Operands 114

Sign Condition 115
Switch-Status Condition 116
Complex Conditions 116
Negated Simple Conditions 117
Combined Conditions 117
Evaluation Rules—Combined Conditions 119
Abbreviated Combined Relation Conditions 120

Conditional Statements 121
IF Statement 121
Nested IF Statements 122

Declaratives 126
EXCEPTION/ERROR Declarative 127
File-Name Option 127
INPUT Option 127
OUTPUT Option 128
I-O Option 128
EXTEND Option 128
General Considerations 128
Programming Notes 129

Input/Output Statements 130
Common Processing Facilities 134
Status Key 134
Invalid Key Condition 136
INTO/FROM Identifier Option 136
Current Record Pointer 137
OPEN Statement 138
Format 1—Sequential Files 139
Format 2—VSAM Indexed Files 140
Format 3—VSAM Relative Files 141
WRITE Statement 141
Format 1—Physical Sequential Files 142
Format 2—VSAM Sequential Files 144
Format 3—VSAM Indexed and Relative files 144
START Statement 146
INVALID KEY Option 147
Status Key Considerations 147
VSAM Indexed Files 147
VSAM Relative Files 148
READ Statement 148
Sequential Access 150
Random Access 151
Dynamic Access 152
REWRITE Statement 152
Sequential Files 152
VSAM Indexed Files 153
VSAM Relative Files 153
DELETE Statement 154
Sequential Access Mode 154
Random or Dynamic Access Mode 154
Programming Notes 154
CLOSE Statement 155
Format 1—Physical Sequential Files 156
Format 2—VSAM Files 157
ACCEPT Statement 158
Format 1—Data Transfer 158
Format 2—System Information Transfer 159
Programming Notes 160
DISPLAY Statement 160
Programming Notes 162

Arithmetic Statements 163
Common Options 163
CORRESPONDING Option 163
GIVING Option 164
ROUNDED Option 164
SIZE ERROR Option 164
Arithmetic Statement Operands 165
Size of Operands 165
Overlapping Operands 165
Multiple Results 166
Programming Notes 166

ADD Statement 166
SUBTRACT Statement 167
MULTIPLY Statement 168
DIVIDE Statement 169
COMPUTE Statement 171
Programming Notes 171

Data Manipulation Statements 172
Overlapping Operands 172
MOVE Statement 172
Elementary Moves 173
Group Moves 175
INSPECT Statement 175
TALLYING Option 178
REPLACING Option 178
BEFORE/AFTER Options 179
INSPECT Statement Examples 179
Programming Notes 180
STRING Statement 181
STRING Statement Execution 181
STRING Statement Example 183
Programming Notes 185
UNSTRING Statement 185
Sending Field 186
Data Receiving Fields 186
UNSTRING Statement Execution 187
UNSTRING Statement Example 190
Programming Notes 192
TRANSFORM Statement 192
Programming Notes 193

Procedure Branching Statements 195
GO TO Statement 195
Format 1—Unconditional GO TO 195
Format 2—Conditional GO TO 195
Format 3—Altered GO TO 196
ALTER Statement 196
Programming Notes 197
Segmentation Information 197
PERFORM Statement 197
Format 1—Basic PERFORM 200
Format 2—TIMES Option 200
Format 3—Conditional PERFORM 200
Format 4—VARYING Option 200
Programming Notes 205
Segmentation Information 205
EXIT Statement 207
Programming Notes 207
STOP Statement 207
Programming Notes 208

Compiler-Directing Statements 209
ENTER Statement 209

Part 5. Special Features 211

Table Handling 212
Table Handling Concepts 212
Table Definition 212
Table References 214
Subscripting 215
Indexing 216
Restrictions on Subscripting and Indexing 217
Table Initialization 218
Table Handling—Data Division 219
OCCURS Clause 219
Format 1—Fixed Length Tables 220
Format 2—Variable Length Tables 220
ASCENDING/DESCENDING KEY Option—Formats 1 and 2 221
INDEXED BY Option—Formats 1 and 2 222
USAGE IS INDEX Clause 222
Table Handling—Procedure Division 223
Relation Conditions 223
SET Statement 224

Format 1—TO Option 224
Format 2—UP/DOWN BY Option 226
SEARCH Statement 226
Format 1—Serial Search 227
Format 2—Binary Search 229
Programming Notes 230
Table Handling Sample Program 231

Sort/Merge 235
Sort/Merge Concepts 235
SORT Concepts 235
MERGE Concepts 236
Sort/Merge—Environment Division 236
FILE-CONTROL Entry 237
I-O-CONTROL Paragraph 237
RERUN Clause 238
SAME SORT/SORT-MERGE AREA Clause 238
Sort/Merge—Data Division 238
Sort/Merge—Procedure Division 239
MERGE Statement 240
SORT Statement 240
SORT/MERGE Statement Options 241
ASCENDING/DESCENDING KEY Option 241
COLLATING SEQUENCE Option 242
USING Option 243
GIVING Option 243
SORT INPUT PROCEDURE Option 244
SORT/MERGE OUTPUT PROCEDURE Option 244
INPUT/OUTPUT PROCEDURE Control 245
RELEASE Statement (Sort Statement Only) 245
RETURN Statement 246
Sort/Merge Special Registers 247
SORT-RETURN 247
SORT-FILE-SIZE, SORT-MODE-SIZE, SORT-CORE-SIZE 247
Programming Notes 248
Sort/Merge Sample Program 248

Report Writer (IBM Extension) 251
Report Writer Concepts 251
Data Division Concepts 251
RD Entry 251
Report Group Description Entries 253
Procedure Division Concepts 255
INITIATE Statement 256
GENERATE Statement 256
TERMINATE Statement 256
USE BEFORE REPORTING Sentence 256
Report Writer—Environment Division 256
SPECIAL-NAMES Paragraph 256
FILE-CONTROL Entry 257
Report Writer—Data Division 257
File Section 257
REPORT Clause 257
RECORD CONTAINS Clause 258
Record Description Entries 258
Report Section 258
RD Entry 259
report-name 259
CODE Clause 259
CONTROL Clause 259
PAGE Clause 261
PAGE LIMIT integer-1 261
HEADING integer-2 261
FIRST DETAIL integer-3 261
LAST DETAIL integer-4 261
FOOTING integer-5 262
General Considerations 262
Report Group Description Entry 264
General Format 1—Level-01 Group Entry 264
General Format 2—Group Entry 265
General Format 3—Elementary Entry 265
General Format 4—Level-01 Elementary Entry 265
General Considerations 265
data-name Clause 267

TYPE Clause 267
LINE Clause 269
Report Group Types 270
NEXT GROUP Clause 271
Report Group Types 271
COLUMN Clause 272
PICTURE Clause 272
GROUP INDICATE Clause 273
SOURCE Clause 273
SUM Clause 274
UPON data-name Option 274
RESET Option 275
General Considerations 275
VALUE Clause 275
USAGE Clause 276
BLANK WHEN ZERO Clause 276
JUSTIFIED Clause 276
Report Writer—Procedure Division 276
OPEN Statement 276
INITIATE Statement 276
GENERATE Statement 277
Detail Reporting 277
Summary Reporting 277
Automatic Operations of the GENERATE Statement 277
TERMINATE Statement 280
USE BEFORE REPORTING Sentence 281
Print Suppression 282
Report Writer Special Registers 282
PAGE-COUNTER 283
LINE-COUNTER 283
Report Writer Sample Program 283

Segmentation Feature 288
Segmentation Concepts 288
Program Segments 288
Fixed Segments 288
Independent Segments 288
Segmentation Logic 289
Segmentation Control 289
COBOL Source Program Considerations 290
Segmentation—Environment Division 290
Segmentation—Procedure Division 290
Segmentation—Special Considerations 291
ALTER Statement 291
PERFORM Statement 291
SORT and MERGE Statements 291
Calling and Called Programs 292

Source Program Library 293
COPY Statement 293
SUPPRESS Option 294
REPLACING Option 294
COPY Statement Example 296
Programming Notes 297
Extended Source Program Library 297
BASIS Statement 297
INSERT/DELETE Statements 298
INSERT Statement 298
DELETE Statement 298

Subprogram Linkage Feature 300
Subprogram Linkage Concepts 300
Control Transfers 300
Common Data 300
COBOL Language Considerations 301
System Considerations 302
Subprogram Linkage—Data Division 303
Record Description Entries 304
Data Item Description Entries 304
Subprogram Linkage—Procedure Division 304
USING Option 305
CALL Statement 306
Segmentation Considerations 307
CANCEL Statement 307

ENTRY Statement 308
EXIT PROGRAM Statement 308
STOP RUN Statement 309
GOBACK Statement 309
Subprogram Linkage Feature Examples 310

Communication Feature 313
Communication Concepts 313
Queue Concepts 313
Input Queues and Queue Structures 313
Output Queues 315
Data Division Concepts 315
Procedure Division Concepts 315
Object Program Scheduling 317
Message and Message Segment Concepts 317
Communication—Data Division 317
CD Entry 318
Format 1, Options 1 and 2—Input CD Entry 319
Format 1—Option 1 320
Format 1—Option 2 324
Format 2—Output CD Entry 324
Communication—Procedure Division 326
ENABLE Statement 326
DISABLE Statement 327
ACCEPT MESSAGE COUNT Statement 328
RECEIVE Statement 329
SEND Statement 331
Format 1 Considerations 332
Format 2 Considerations 332

Debugging Features 335
COBOL Source Language Debugging 335
Compile-Time Switch 335
Object-Time Switch 335
USE FOR DEBUGGING Declarative 336
DEBUG-ITEM Special Register 339
DEBUGGING LINES 340
FIPS Flagger 341
1975 FIPS COBOL Flagging 343
1975 Full FIPS COBOL Flagging 343
1975 High-Intermediate FIPS COBOL Flagging 345
1975 Low-Intermediate FIPS COBOL Flagging 346
1975 Low FIPS COBOL Flagging 347
1972 FIPS COBOL Flagging 348
1972 Full FIPS COBOL Flagging 348
1972 High-Intermediate FIPS COBOL Flagging 350
1972 Low-Intermediate FIPS COBOL Flagging 351
1972 Low FIPS COBOL Flagging 351

Part 6. System Dependencies 353

System Dependencies 354
CD Entry for INPUT 354
FD ENTRY 354
BLOCK CONTAINS 0 CHARACTERS Clause 354
RECORD CONTAINS 0 CHARACTERS Clause 354
Recording Mode 355
FILE-CONTROL Entry 356
ASSIGN Clause 356
PASSWORD Clause 356
RESERVE Clause 356
I-O-CONTROL Entry 357
SAME Clause 357
Multiple File Tape Clause 357
Input/Output Statements 357
OPEN Statement 357
WRITE ADVANCING Statement 357
CLOSE Statement 357
Report Writer—RECORD CONTAINS Clause 357
SD Entry for Sort/Merge 358
Source Program Library 358
SPECIAL-NAMES Paragraph 358
Special Registers 358
RETURN-CODE 358

- SORT-CORE-SIZE 359
- SORT-MESSAGE 359
- SORT-RETURN 359
- WHEN-COMPILED 359

Status Key Value 96 360

Subprogram Linkage 360

- Linkage Section 360
- CALL Statement 360
- USING Option 361
- Program Termination Statements 361

Part 7. Supplementary Material 363

Appendix A. IBM OS Full ANS COBOL Items—(IBM Extension) 364

Section I—I/O Language Extensions 364

- File Processing Capabilities 364
- File Control Paragraph 365
  - ASSIGN Clause 365
  - ACTUAL KEY Clause 366
  - NOMINAL KEY Clause 366
  - RECORD KEY Clause 367
  - TRACK-AREA Clause 367
  - TRACK-LIMIT Clause 368
- I-O-CONTROL Paragraph 368
  - RERUN Clause 368
  - MULTIPLE FILE TAPE Clause 368
  - APPLY Clause 369

DATA DIVISION 370

- FD Entry 370
  - LABEL RECORDS Clause 370
  - RECORDING MODE Clause 371
- PROCEDURE DIVISION DECLARATIVES 372
  - Label Declaratives—Format 1 373
  - Error Declaratives—Format 2 375
- INPUT/OUTPUT Statements 378
  - OPEN Statement 383
  - START Statement 383
  - SEEK Statement 385
  - READ Statement 385
  - WRITE Statement 386
  - REWRITE Statement 389
  - CLOSE Statement 390
  - Sequential File Processing 391
  - Random File Processing 393
- ASCII FILE CONSIDERATIONS 393
  - ASSIGN Clause 393
  - RECORDING MODE Clause 394
  - LABEL Procedure Declarative 394
  - Relation Conditions 394

Section II—"New" and "Old" Language Extensions 394

- LANGUAGE CONSIDERATIONS 395
  - Floating-Point Numeric Literals 395
  - Special Registers 395
- IDENTIFICATION DIVISION 395
- DATA DESCRIPTION ENTRY 396
  - Level-Numbers 396
  - REDEFINES Clause 396
  - PICTURE Clause 396
  - USAGE Clause—Floating-Point Items 396
  - VALUE Clause 398
- PROCEDURE DIVISION 398
  - Section Headers 398
  - THEN Reserved Word 398
  - Floating-Point Numeric Operands 398
  - Arithmetic Expressions—Unary Operators 399
  - Comparisons 399
  - IF Statement 399
  - EXAMINE Statement 399
  - NOTE Statement 400
  - MOVE Statement 401
- TABLE HANDLING FEATURE 401
  - OCCURS Clause 401
  - USAGE IS INDEX Clause 402
  - SEARCH ALL Statement 402

DEBUGGING LANGUAGE 402
READY or RESET TRACE Statement 402
EXHIBIT Statement 402
ON Statement 404
COMPILE-TIME DEBUGGING PACKET 405
DEBUG CARD 405
SOURCE LISTING FORMAT CONTROL 406
EJECT Statement 406
SKIP1/2/3 Statements 406
Global Items 406
Quotation Mark Specification 406
Section III—"Old" Language Extensions 407
Environment Division 407
SPECIAL-NAMES Paragraph 407
SELECT OPTIONAL Clause 407
RESERVE ALTERNATE AREAS Clause 407
Data Division 407
JUSTIFIED in a VALUE clause 407
Procedure Division 408
NOT in Combined Relation Conditions 408
MOVE Statement Scaling 408
PERFORM Statement in Segmented Programs 408
UNSTRING Statement—DELIMITED BY ALL Option 408
Source Program Library 408
COPY Statement 408
Communication Feature 409
RECEIVE MESSAGE Statement 409

Appendix B. ASCII Considerations 410
Environment Division 410
OBJECT-COMPUTER and SPECIAL-NAMES Paragraphs 410
FILE-CONTROL Paragraph 411
I-O-CONTROL Paragraph 411
Data Division 411
FD Entry—CODE-SET Clause 411
Data Description Entries 411
SORT/MERGE Feature 411
Environment Division 411
DATA DIVISION 412
SORT and MERGE Statements 412

Appendix C. System/370 Unit Record Processing 413
3505/3525 Card Processing 413
3505 OMR Processing 413
3505/3525 RCE Processing 413
RCE and OMR Format Descriptor 413
ASSIGN Clause Specification 414
3525 Combined Function Processing 414
Environment Division Considerations 415
SPECIAL-NAMES Paragraph 415
Data Division Considerations 415
Procedure Division Considerations 415
OPEN Statement 416
READ Statement 416
WRITE Statement—Punch Function Files 416
WRITE Statement—Print Function Files 417
CLOSE Statement 417
3886 OCR Processing 418

Appendix D. Intermediate Results 419
Compiler Calculation of Intermediate Results 420

Appendix E. Sample File-Processing Programs 422
Sequential File Creation 422
Sequential File Updating and Extension 424
Indexed File Creation 426
Indexed File Updating 427
Relative File Creation 430
Relative File Updating 432
Relative File Retrieval 434

Appendix F. OS/VS COBOL Reserved Word List 436

Appendix G. EBCDIC and ASCII Collating Sequences . . . 446

EBCDIC Collating Sequence 446
ASCII Collating Sequence 450

Appendix H. COBOL Statements Flagged by Specifying MIGR 453
LANGLVL(1) 453
Communications 453
Report Writer 453
ISAM 453
Use for Debugging 454
BDAM 454
Other Statements 454
Arithmetic Items 455

Glossary 456

Index 470

FIGURES

1. Format Notation 5
2. COBOL Characters and Their Meanings 13
3. User-Defined Word Sets and Rules for Formation . . . 14
4. Separator Characters 18
5. COBOL Coding Form and Standard COBOL Format 21
6. Sequence of Elements in Area A and Area B 22
7. LINAGE Clause and Logical Page Depth 61
8. Level-Number Concepts 65
9. Classes and Categories of Data 66
10. PICTURE Clause Symbol Order 77
11. Examples of Numeric Items 80
12. Valid Editing for Each Data Category 82
13. Valid Simple Insertion Characters 82
14. Examples of Simple Insertion Editing 83
15. Examples of Special Insertion Editing 83
16. Editing Sign Control Results 84
17. Examples of Fixed Insertion Editing 84
18. Examples of Floating Insertion Editing 85
19. Examples of Zero Suppression and Replacement Editing 86
20. Insertion of Intraoccurrence Slack Bytes 91
21. Insertion of Interoccurrence Slack Bytes 92
22. Internal Representation of Numeric Items 96
23. RENAMES Clause—Valid and Invalid Specifications . 102
24. Conditional Statements and Their Categories 106
25. Imperative Statements and Their Categories 107
26. Compiler-Directing Statements and Their Categories 108
27. Binary and Unary Operators 109
28. Valid Arithmetic Symbol Pairs 110
29. Valid Forms of the Class Test 112
30. Relational Operators and Their Meanings 113
31. Permissible Numeric and Nonnumeric Comparisons . . 114
32. Logical Connectives and Their Meanings 116
33. Combined Conditions—Permissible Element Sequences 118
34. Logical Operators and Evaluation Results of Combined Conditions 118
35. Abbreviated Combined Relation-Condition Equivalents 120
36. Nested IF Statement Evaluation 123
37. Nested IF Statement—True/False Combinations . . . 124
38. Nested IF Statement—Flowchart 125
39. Sequential Files—Required and Optional Entries . . 131
40. VSAM Indexed Direct Access Storage Files—Required and Optional Entries 132
41. VSAM Relative Direct Access Storage Files—Required and Optional Entries 133
42. Status Key Values and Meanings—VSAM Files 135
43. Status Key Values and Meanings—Physical Sequential Files 136
44. READ Statement with Multiple Record Description . . 149
45. Physical Sequential File Types and CLOSE Statement Options 156
46. Meanings of Keys Used in Figure 45 157
47. VSAM Files and CLOSE Statement Options 158
48. Meanings of Keys Used in Figure 47 158
49. Valid and Invalid Elementary Moves 174
50. INSPECT Statement Execution Results 177
51. STRING Statement Execution Results 183
52. STRING Statement Example—Output 185
53. Results of UNSTRING Statement Execution 189
54. UNSTRING Statement Example—Input Data 191
55. Examples of Data Transformation 193
56. TRANSFORM Statement—FROM and TO Options 193
57. Valid PERFORM Statement Execution Sequences 199
58. Format 4 PERFORM Statement Logic—Varying One Identifier 202
59. Format 4 PERFORM Statement Logic—Varying Two Identifiers 204
60. Format 4 PERFORM Statement Logic—Varying Three Identifiers 206

61. Storage Layout for TABLE-THREE 214
62. Comparisons for Index-Names and Index Data Items . 224
63. Sending and Receiving Fields for Format 1 SET Statement 225
64. Format 1 SEARCH with Two WHEN Options 229
65. RD Entry—PAGE Clause Concepts 252
66. RD Entry—CONTROL Clause Concepts 253
67. Report Group Description Entry Concepts 255
68. Valid LINE Clause Specifications for Each PAGE Clause Range 262
69. PAGE Clause Ranges 263
70. Values Assumed for Omitted PAGE Clause Options . . 263
71. Valid Report Group Clauses—by Report Group TYPE . 266
72. TYPE Clause Presentation Rules 268
73. First GENERATE Statement—Sequence of Operations . 278
74. Subsequent GENERATE Statements—Sequence of Operations 279
75. TERMINATE Statement—Sequence of Operations 281
76. Pseudo-Text Continuation Lines 295
77. Common Data Items in Subprogram Linkage 301
78. Predefined Input Queue Structure Example 314
79. Communication Feature Relationships 316
80. Input CD Entry—Implicit Description 319
81. STATUS KEY Field—Possible Values 323
82. Output CD Entry—Implicit Description 325
83. End Indicator Codes 333
84. Execution of Debugging Declaratives 338
85. DEBUG-ITEM Subfield Contents 340
86. The 1974 Standard and 1975 FIPS COBOL 342
87. The 1968 Standard and 1972 FIPS COBOL 343
88. Equivalent I-O Terms—IBM COBOL and OS/VS 354
89. Valid Names of Function-Name-1 358
90. Program Termination Statements—Actions Taken . . . 362
91. OS ANS COBOL File Processing Capabilities 365
92. Information Supplied With the GIVING Option When an Error Declarative is Entered 377
93. Standard Sequential Files—Required and Optional Entries 379
94. Direct Files (Direct Access Storage Devices Only)—Required and Optional Entries 380
95. Indexed Files (Direct Access Storage Devices Only)—Required and Optional Entries 381
96. Relative Files (Direct Access Storage Devices Only)—Required and Optional 382
97. Values of Identifier-2 and Interpretations—POSITIONING Option 387
98. Values of Integer and Interpretations—POSITIONING Option 387
99. Sequential File Types and Options of the CLOSE Statement 391
100. Random File Types and Options of the CLOSE Statement 393
101. COPY Statement—"Old" Language Options 409
102. IBM 3525 Card Punch—Valid Function-Names 415
103. IBM 3525 Card Punch Combined Function Processing Order 416
104. IBM 3525 Card Punch Valid Function Names with WRITE AFTER ADVANCING Statement 417
105. Compiler Action on Intermediate Results 421

GENERAL DESCRIPTION

COBOL (COmmon Business Oriented Language) is a programming language that resembles English. As its name implies, COBOL is especially efficient in the processing of business problems. Such problems usually involve little algebraic or logical processing. Instead, they usually manipulate large files of data in a relatively simple way. That is, COBOL emphasizes the description and handling of data items and of input/output records.

COBOL language development and definition is the function of CODASYL (the Conference On DAta SYstems Languages). The standard of the language in the USA is American National Standard COBOL, X3.23-1974, as approved by the American National Standards Institute (ANSI). References in this manual to the 1974 standard are references to this standard. (Similarly, references to the 1968 standard are references to the former standard in the USA—American National Standard COBOL, X3.23-1968.)

Both standards have their international counterparts, from the International Organization for Standardization (ISO). For the 1974 standard, this is ISO 1989-1978. For the 1968 standard, this is ISO 1989-1972.

The OS/VS COBOL Compiler and Library is an IBM Program Product that accepts and compiles COBOL programs written in support of the 1974 standard, the 1968 standard, plus a number of IBM extensions. The following sections describe the language level implemented and language-independent compiler features.

LANGUAGE LEVEL

OS/VS COBOL is designed according to the specifications of the indicated levels of the following 1974 standard modules:

- NUCLEUS (2 NUC 1,2)—which provides improved internal processing capabilities. Extended data manipulation statements, enhanced arithmetic capabilities, user-specified collating sequences, and eased data grouping rules are all provided.

- TABLE HANDLING (2 TBL 1,2)—which eases rules on subscripting and allows mixed indexes and literal subscripts.

- SEQUENTIAL I-O(2 SEQ 1,2)—which implements VSAM ESDS processing, allows sequential files to be extended, and provides added page placement capabilities for files destined for printed output.

- RELATIVE I-O (2 REL 0,2)—which allows the user to specify relative record file organization—the records in such a file are stored and retrieved in the order of their relative record numbers. Storage and retrieval can be sequential or random. Relative I-O is implemented through the VSAM RRDS capabilities.

- INDEXED I-O (2 INX 0,2)—which gives added indexed file processing capabilities through VSAM KSDS processing. Records are stored according to an embedded prime record key; they can be retrieved through the prime record key or through embedded alternate record keys. Storage and retrieval can be sequential or random.

- SORT-MERGE (2 SRT 0,2)—which gives the capability of ordering the records in one or more files (sorting) and of combining two or more identically ordered files (merging).

The sort or merge can be upon either the EBCDIC or ASCII collating sequence, or upon a user-specified collating sequence.

- SEGMENTATION (2 SEG 0,2)—which allows the user to specify object program overlay requirements.

- LIBRARY (2 LIB 0,2)—which allows the user to replace all occurrences of a given text with alternate text during compilation. Multiple COBOL source libraries can be specified.

- DEBUG (2 DEB 0,2)—which provides the capability of monitoring object program execution through Declarative procedures, special debugging lines, and a special register, DEBUG-ITEM, which gives specific information about execution status.

- INTER-PROGRAM COMMUNICATION (2 IPC 0,2)—which allows a COBOL program to communicate with one or more other programs through transfers of control and access to common data items.

- COMMUNICATION (2 COM 0,2)—which provides the ability to communicate through a Message Control Program (MCP) with local or remote communications devices, and to access, process, and create partial and complete messages.

OS/VS COBOL also incorporates all language elements from IBM OS Full American National Standard COBOL, with two exceptions: the MESSAGE COUNT clause replaces the QUEUE DEPTH clause, and the ACCEPT MESSAGE COUNT statement replaces the IF MESSAGE statement. IBM OS Full American National Standard COBOL is designed in support of the highest level of the 1968 standard plus IBM extensions. All such language that differs from the 1974 Standard can be considered an extension to it. Major extensions are:

- Report Writer—which allows the user to produce reports by specifying what the physical appearance of the report should be; the Report Writer then produces the necessary procedures to generate the report. The IBM Report Writer is compatible with the 1968 standard.

- PASSWORD Clause—which provides file security for VSAM files.

- TRANSFORM Statement—which provides easy translation capabilities from one collating sequence to another.

- ENTRY Statement—which gives the user the ability to specify alternate entry points in a called program.

- WHEN-COMPILED Special Register—which provides a means of associating a compilation listing with both the object program and the output produced at execution.

The main text of this manual documents 1974 standard COBOL together with those IBM extensions especially useful with it. Some such IBM extensions are designed to complement the 1974 Standard language; others—such as the complete Report Writer chapter—are continued from IBM OS Full American National Standard COBOL.

Appendix A documents these language elements continued for compatibility purposes from IBM OS Full American National Standard COBOL (both 1968 standard language and IBM extensions). Other appendixes give useful supplemental information. The Glossary gives definitions of COBOL terms.

COMPILER FEATURES

The following language-independent features are made available with OS/VS COBOL:

Virtual Storage Access Method (VSAM) Support—which provides fast storage and retrieval of records, password protection, centralized and simplified data and space management, advanced error recovery facilities, plus system and user catalogs.

COBOL supports sequential files (through VSAM ESDS capabilities), indexed files with alternate indexes (through VSAM KSDS capabilities), and relative files (through VSAM RRDS capabilities).

Federal Information Processing Standard Flagger (FIPS)—which issues messages identifying nonstandard elements in a COBOL source program. The FIPS Flagger makes it possible to ensure that COBOL clauses and statements in an OS/VS COBOL source program conform to either the 1975 or 1972 Federal Information Processing Standard.

Lister Option—provides specially formatted listings with embedded cross-references for increased intelligibility and ease of use. Reformatted source deck is available as an option.

Verb Profiles—facilitates identifying and locating verbs in the COBOL source program. Options provide verb summary or verb cross-reference listing which includes verb summary.

Execution Time Statistics—maintains a count of the number of times each verb in the COBOL source program is executed during an individual program execution.

System/370 Device Support—any valid OS/VS device can be used with a OS/VS COBOL program. In most cases, support is transparent to the OS/VS COBOL program. There are special considerations for the following devices:

- 3886 OCR (Optical Character Reader)—this device reads multiline alphanumeric or numeric machine-printed documents or numeric hand-printed documents, with stacker selection. OS/VS COBOL support is through an object-time library subroutine.

- 3330, 3340 Disk Facilities—these devices are high-speed large-capacity disks, with the rotational positional sensing (RPS) feature. Use of the fixed block standard option, which can be specified at object time, results in much improved performance.

- Multifunction Card Devices—OS/VS COBOL supports the combined function processing available through these devices. Combined functions available are: read/punch, read/print, punch/print, and read/punch/print.

Language-independent features continued from OS Full American Nation Standard COBOL are:

Advanced Symbolic Debugging—provides faster and easier debugging for the COBOL programmer. At abnormal termination a formatted dump, using COBOL source data-names, is produced. Execution-time dynamic dumps at user-specified points in the Procedure Division can also be obtained. When the symbolic debugging feature is requested, optimized object code is automatically provided.

Optimized Object Code—can be requested, resulting in considerably smaller object programs than are produced without optimization. For COBOL programs that are not I/O bound, execution time is reduced.

COBOL Library Management Facility—allows installations running with multiple COBOL regions/partitions to save considerable main storage by sharing some or all of the COBOL library subroutine modules.

Syntax-checking Compilation—saves machine time and money while debugging source syntax errors. When unconditional syntax checking is requested, the source program is scanned for syntax errors and such error messages are generated, but no object code is produced. When conditional syntax checking is requested, a full compilation is produced if no messages or only W-level or C-level messages are generated; if one or more E-level or D-level messages are generated, no object code is produced. Selected test cases have shown that when object code is not generated, compilation time is reduced.

Optional alphabetically ordered cross-reference listings—significant performance improvement has been made to the current cross-reference option which preserves source statement order.

A flow trace option—which prints a formatted trace of the last procedures executed before an abnormal termination of execution. The number of procedures to be traced is specified by the user.

A statement number option—which provides the user with the number of the COBOL statement, and of the verb within the statement, being executed if an abnormal termination of execution occurs.

Expansion of the functions of the CLIST and DMAP compiler options—in addition to the condensed listing (CLIST) and the glossary (DMAP), global tables, literal pools, and register assignments are included.

Batch Compilation—more than one program or subprogram can be compiled with a single invocation of the compiler, resulting in a reduction in compilation time.

Separately located Installation Defaults—installation default options are separately located from other coding to improve maintainability and serviceability.

FEATURES DEPENDENT ON TSO

With Time Sharing Option (TSO), the terminal user may choose options to determine the characteristics of compiler output to the terminal. The user may direct to the terminal:

- Compilation progress and diagnostic messages
- The compiler's entire data listing set

The user can suppress either category or may suppress all output to the terminal.

In addition, if the user has recorded line numbers in the input data set, the compiler may be instructed to substitute these numbers for internal statement numbers in any diagnostic messages printed on the terminal. Also, when diagnostic messages are printed on the terminal, a message stating the total number of statements in error can be included at the request of the user.

In addition to the program development features included within the OS/VS COBOL Compiler and Library itself, there are two related Program Products, available under TSO, that greatly facilitate program development. Both reduce program turnaround time, and increase programmer productivity. These Program Products are: the TSO COBOL Prompter and COBOL Interactive Debug.

Both are described in IBM OS COBOL Interactive Debug and (TSO) COBOL Prompter, General Information.

FORMAT NOTATION

The notation used to illustrate COBOL formats is shown in the following box. Each numbered item in the sample format is keyed to an explanation in Figure 1. Further explanations of the notation follow the figure.

```
 Format

 1                2                    3
 |                |                    |
                  ┌              ┐   ┌              ┐
[[]   STATEMENT   < identifier-1 >   │ identifier-2 │ ...
                  │ literal-1    │   │ literal-2    │
                  └              ┘   └              ┘
                             4                     5                6
                             |                     |                |
    TO  identifier-m [ROUNDED] [identifier-n [ROUNDED] ] ...

     7                       8
     |                       |
    [ON SIZE ERROR   imperative-statement]  []]
```

No.	Symbol	Meaning	Example
1	Bracket in a box	A portion of syntax that, as an IBM extension, is optional.	[[] ... ... []]
2	Braces	Required choice among several items.	< identifier-n > < literal-n >
3	Brackets	Optional items.	[identifier-n] [literal-n]
4	<u>UNDERLINED UPPERCASE LETTERS</u>	Key words.	<u>ROUNDED</u>
5	Hyphen and digit Hyphen and letter	Text reference.	identifier-1 identifier-m
6	Ellipsis	Repeatable items.	...
7	UPPERCASE LETTERS	Optional reserved words.	ON
8	lowercase letters	User-defined words.	imperative

Figure 1. Format Notation

Further Explanations of the "Meaning" Column:

1. **Optional IBM extension portions of syntax** are indicated by a pair of brackets, each enclosed in a box:

   $\boxed{[}\quad \dots \quad \boxed{]}$

2. **Required choice to be made among several items** is indicated by a set of braces enclosing vertically stacked alternatives.

   One, and only one, of the enclosed items is required.

   $$\left\{\begin{matrix} x \\ y \\ z \end{matrix}\right\}$$

3. **Optional items** are enclosed in brackets.

   An item within brackets may be included or omitted, depending on the requirements of the program. When two or more items are stacked within brackets, one or none of them may be specified.

   $$[x] \qquad \text{or} \qquad \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

4. **Key words** are shown in UNDERLINED UPPERCASE LETTERS. Key words are **reserved words** that are required unless the portion of the format containing them is itself optional. If any key word is missing or misspelled, it is considered an error in the program.

5. **Text reference** is designated by a hyphen and a digit (-1) or a hyphen and a letter (-m) following a user-defined word. This suffix is only for text reference; it does not change the syntactical definition of the word.

6. **Repeatable items** are indicated by an ellipsis (...). Such an item can be coded once or any number of times. An ellipsis can follow a single word, or a group of lowercase or reserved words surrounded by brackets and/or braces. Portions of a format that are grouped within brackets or braces are presented as a unit, and, when repetition is specified, the entire unit must be repeated.

7. **Optional reserved words** are shown in UPPERCASE LETTERS but are not underlined. They are reserved words that are included only for readability and may be omitted without changing the logic of the program. If an optional word is included, it must be spelled correctly, or it is considered an error in the program.

8. **User-defined words** are printed in lowercase letters and represent information you supply.

Other notations have the following meanings:

- Arithmetic and logical operator characters (+, -, *, /, >, <, and =), in formats, are required, even though they are not underlined.

- All punctuation and other special characters appearing in formats (except braces, brackets, and ellipses) represent the actual occurrence of those characters. Where such punctuation characters are shown, they are required by the format; if they are omitted, there will be an error in the program. Additional punctuation may be specified, according to the punctuation rules contained in "Separators" on page 18.

- The required clauses and optional clauses (when written) must be written in the sequence shown in the format, unless the associated rules explicitly state otherwise.

IBM EXTENSIONS

IBM extensions within formats and figures are shown in boxes. For example:

Format

```
[STATEMENT IS data-name-2  |[data-name-3]| ].
```

Extensions within text are documented in separate paragraphs or sections. For example:

IBM Extension

IBM extensions in text are shown this way.

End of IBM Extension

If an entire section of syntax is an optional IBM extension, it is indicated by enclosing each bracket in a box at the start and the end of the extension. (See number 1 in Figure 1.)

Occasionally, an IBM extension directly contradicts a rule or restriction that immediately precedes it. The standard is presented first, because some programmers use COBOL without IBM extensions. The extension is then presented for those who do use them.

PART 1. LANGUAGE CONSIDERATIONS

- COBOL PROGRAM STRUCTURE
- STRUCTURE OF THE LANGUAGE
- STANDARD COBOL FORMAT
- METHODS OF DATA REFERENCE

COBOL PROGRAM STRUCTURE

Every COBOL source program is divided into four divisions. Each division must be placed in its proper sequence, and each must begin with a division header.

In subsequent sections of this publication, the rules for writing COBOL source programs and methods of data reference are given.

THE COBOL DIVISIONS

The four divisions of a COBOL source program, and their functions in solving a data processing problem, are:

Identification Division

The Identification Division names the program and, optionally, documents the date the program was written, the compilation date, and other pertinent information.

Environment Division

The Environment Division describes the computer(s) to be used and specifies the machine equipment and equipment features used by the program. This description includes a description of the relationship of files of data with actual input/output devices.

Data Division

The Data Division defines the nature and characteristics of all data the program is to process. This includes both the data used in input/output operations and the data developed for internal processing.

Procedure Division

The Procedure Division consists of executable statements that process the data in the manner the programmer defines. Unless the programmer defines some other order, the statements are executed in the order in which they are written.

CLAUSES AND STATEMENTS

Every COBOL source program is written in clauses and statements, each of which describes some specific aspect of the data processing problem solution:

- Clauses—written in the Environment and Data Divisions—specify attributes of entries. A series of clauses, ending with a period, is defined as an entry.

- Statements—written in the Procedure Division—specify an action to be taken by the object program. A series of statements, ending with a period, is defined as a sentence.

Each clause or statement in the program can be subdivided into smaller syntactical units called phrases or options. A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL clause or statement. An option is a phrase for which the programmer makes a choice of alternative required or optional wordings, depending on the meaning this phrase is to have.

Clauses, entries, statements, and sentences can be combined into paragraphs or sections. Paragraphs and sections each define some larger part of the data processing problem solution. Specific rules for the formation of each element are given in the documentation for each Division of the COBOL program.

CLAUSE AND STATEMENT SPECIFICATION ORDER

When specified, each required or optional clause or statement—even those treated as documentation—must be written in the sequence shown in its format, unless the associated rules explicitly state otherwise.

STRUCTURE OF THE LANGUAGE

In COBOL, the indivisible unit of data is the character. Fifty-one EBCDIC characters form the COBOL character set: the 26 letters of the English alphabet, the 10 Arabic numerals, and 15 special characters.

Individual COBOL characters are put together to form character-strings and separators.

A character-string is a character or sequence of contiguous characters that forms a word, a literal, a PICTURE character-string, or a comment. A character-string can be delimited only by a separator.

A separator is a contiguous string of one or more punctuation characters. A separator can be placed next to another separator or next to a character-string.

Except for comments and nonnumeric literals (which may use any character within the EBCDIC set), the 51 characters are the only valid characters in a COBOL program. Figure 2 shows the valid COBOL characters in ascending EBCDIC sequence and their usage in a COBOL program.

COBOL Character	Meaning	Use
	space	punctuation character
.	decimal point; period	editing character; punctuation character
<	less than	relation character
(	left parenthesis	punctuation character
+	plus symbol	arithmetic operator; sign; editing character
$	dollar sign	editing character
*	asterisk	arithmetic operator; editing character
)	right parenthesis	punctuation character
;	semicolon	punctuation character
-	minus symbol; hyphen	arithmetic operator; sign; editing character
/	stroke; slash	arithmetic operator; editing character; line control character
,	comma	punctuation character; editing character
>	greater than	relation character
=	equal sign	relation character; punctuation character
"	quotation mark	punctuation character
A-Z	alphabet	alphabetic character
0-9	Arabic numerals	numeric character

Notes:

1. All COBOL characters are considered to be alphanumeric characters.
2. For the quotation mark as previously implemented, see Appendix A.

Figure 2. COBOL Characters and Their Meanings

CHARACTER-STRINGS

COBOL character-strings form words, literals, PICTURE character-strings, and comments. These are described in the following paragraphs.

COBOL WORDS

A COBOL word can be a user-defined word, a system-name, or a reserved word. A COBOL word can belong to only one of these classes.

The maximum length of a COBOL word is 30 characters.

User-Defined Words

A user-defined word is a COBOL word supplied by the programmer. Valid characters in a user-defined word are:

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

The hyphen may not appear as the first or last character in a user-defined word.

A list of user-defined word sets, together with rules for their formation, is given in Figure 3.

The function of each user-defined word in any specific clause or statement is included in the prose description for each clause or statement.

User-Defined Word Sets	Rules For Formation
alphabet-name cd-name condition-name data-name record-name file-name index-name mnemonic-name report-name routine-name	Must contain at least one alphabetic character. Within each set the name must be unique, either because no other word is made up of an identical character-string, or because it can be made unique through qualification. (See the section on Methods of Data Reference.)
library-name program-name text-name	Same rules of formation as above. However, the system uses the first 8 characters as the identifying name; these first 8 characters, therefore, should be unique among library-names, program-names, and text-names.
paragraph-name section-name	Need not contain an alphabetic character. Other rules as in first paragraph above.
level-numbers: 01-49, 66, 77, 88 priority-numbers: 00-99	Must be a 1- or 2-digit integer. Need not be unique.

Figure 3. User-Defined Word Sets and Rules for Formation

System-Names

A system-name is an IBM-defined name which is used to communicate with the system. There are four types of system-names:

- Computer-Names
- Language-Names
- Function-Names
- Assignment-Names

The function of each system-name is described with the format in which it appears; each is defined in the Glossary.

Reserved Words

A reserved word is a COBOL word with fixed meaning(s) in a COBOL source program. A reserved word must not be specified as a user-defined word or as a system-name. Reserved words can be used only as specified in the formats for a COBOL source program.

There are six types of reserved words:

- Key words
- Optional words
- Connectives
- Special registers
- Special-Character words
- Figurative-Constants

Each type is described in the following paragraphs.

KEY WORDS: Words that are required within a given clause, entry, or statement. There are three types of key words:

- Verbs such as ADD, READ, ENTER
- Required words which appear in clause, entry, or statement formats, such as the word USING in the MERGE statement
- Words with a specific functional meaning, such as NEGATIVE or SECTION

OPTIONAL WORDS: Words that may be, but need not be, included in a clause, entry, or statement. When an optional word is omitted, the meaning of the COBOL program is unchanged.

CONNECTIVES: There are three types of connectives:

- Qualifier connectives (OF, IN), which associate a data-name, condition-name, text-name, or paragraph-name with its qualifier.
- Series connectives (the comma and semicolon), which link two or more consecutive operands. (An operand is a data item or literal that is acted upon by the COBOL program.)
- Logical connectives (AND, OR, AND NOT, OR NOT) used in specifying conditions.

SPECIAL REGISTERS: Compiler-generated storage areas whose primary use is to store information produced through one of the specific COBOL features. Each such storage area has a fixed name, and need not be further defined within the program. These special registers include the following:

- DATE, DAY, TIME (see ACCEPT statement in Procedure Division)
- LINAGE-COUNTER (see LINAGE clause in Data Division)
- DEBUG-ITEM (see Debugging Features chapter)

IBM Extension

LINE-COUNTER, PAGE-COUNTER (see Report Writer chapter)

End of IBM Extension

Other special registers are described in the chapter on System Dependencies.

SPECIAL-CHARACTER WORDS: Arithmetic operators (+ - / * **) or relation characters (<>=). Arithmetic operators are described in the chapter on Arithmetic Expressions. Relation characters are described in the relation condition description of the Conditional Expressions chapter.

FIGURATIVE CONSTANTS: Name and refer to specific constant values.

Figurative constants are described in the section on Figurative Constants.

LITERALS

A literal is a character-string whose value is specified either by the ordered set of characters of which it is composed, or by the specification of a figurative constant. There are two types of literals: nonnumeric and numeric.

Nonnumeric Literals

A nonnumeric literal is a character-string that can contain any allowable character from the EBCDIC set; its maximum length is 120 characters.

single or double

A nonnumeric literal must be enclosed in quotation marks (" "). These quotation marks are excluded from the literal.

page 406

Note: For the quotation mark as formerly implemented, see Appendix A.

Any punctuation characters included within a nonnumeric literal are part of the value of the literal. An embedded apostrophe must be represented by a pair of contiguous apostrophes (''); one apostrophe is then part of the value of the literal.

Every nonnumeric literal is of the alphanumeric category. (Data categories are defined in the PICTURE Clause description of the Data Division chapter.)

Numeric Literals

A numeric literal is a character-string whose characters are selected from the digits 0 through 9, a sign character (+ or -), and/or the decimal point. The following rules apply:

- One through 18 digits are allowed.
- Only one sign character is allowed. If a sign character is included, it must be the leftmost character of the literal. If the literal is unsigned, it is positive in value.
- Only one decimal point is allowed. If a decimal point is included, it is treated as an assumed decimal point (that is, as not taking up a character position in the literal). The decimal point may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is an integer. (The word integer appearing in a format represents a numeric literal of nonzero value that contains no sign and no decimal point; any other restrictions are included with the description of the format.)

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

Every numeric literal is of the numeric category. (Data categories are defined in the PICTURE Clause description of the Data Division chapter.)

FIGURATIVE CONSTANTS

Figurative constants are reserved words used to name and refer to specific constant values. The reserved words for figurative constants and their meanings are:

ZERO, ZEROES, ZEROS
Represents the value 0, or one or more occurrences of the character 0, depending on context. Can be numeric or nonnumeric, depending on context.

SPACE, SPACES
Represents one or more blanks or spaces. Must be nonnumeric.

HIGH-VALUE, HIGH-VALUES
Represents one or more occurrences of the character that has the highest value in the collating sequence used. For the EBCDIC (native) collating sequence, the character is hexadecimal "FF"; for other collating sequences, the actual character used depends on the collating sequence. When used in a COBOL program, HIGH-VALUE is treated as a nonnumeric literal.

LOW-VALUE, LOW-VALUES
Represents one or more occurrences of the character that has the lowest value in the collating sequence used. For the EBCDIC (native) collating sequence, the character is hexadecimal "00"; for other collating sequences, the actual character used depends on the collating sequence. When used in a COBOL program, LOW-VALUE is treated as a nonnumeric literal.

QUOTE, QUOTES
Represents one or more occurrences of the quotation mark character and must be nonnumeric. The word QUOTE (QUOTES) cannot be used in place of a quotation mark to enclose a nonnumeric literal.

ALL literal
Represents one or more occurrences of the string of characters composing the literal and must be nonnumeric. The literal must be either a nonnumeric literal or a figurative constant other than the ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only. The figurative constant ALL literal must not be used with the DISPLAY, INSPECT, STRING, STOP, or UNSTRING statements.

The singular and plural forms of a figurative constant are equivalent, and may be used interchangeably. For example, if DATA-NAME-1 is a 5-character data item, both following statements

```
MOVE SPACE TO DATA-NAME-1

MOVE SPACES TO DATA-NAME-1
```

will fill DATA-NAME-1 with five spaces.

A figurative constant may be used wherever a nonnumeric literal appears in a format. When a numeric literal appears in a format, only the figurative constant ZERO (ZEROS, ZEROES) may be specified.

The length of a figurative constant depends on the context of the program. The following rules apply:

- When a figurative constant is associated with a data item (as, for example, when it is moved to or compared with another item), the length of the figurative constant character-string is equal to the size of the associated data item.

- When a figurative constant is not associated with another data item (as, for example, in a DISPLAY, INSPECT, STRING, STOP, or UNSTRING statement), the length of the character-string is one character.

IBM Extension

When a figurative constant is associated with a TRANSFORM statement, the length of the character-string is one character.

End of IBM Extension

PICTURE CHARACTER-STRINGS

A PICTURE character-string consists of COBOL characters used as symbols in the PICTURE clause. The description of the PICTURE Clause in the Data Division chapter gives details.

COMMENTS

A comment is a character-string that can contain any combination of characters from the EBCDIC set, and which serves only as documentation. Comments take one of two forms:

- A comment-entry in the Identification Division
- A comment line (preceded by an asterisk (*) or a slash (/) in column 7) in any division of the program

Comment-entries are described in the chapter on the Identification Division. Comment lines are described in the chapter on Standard COBOL Format.

SEPARATORS

A separator is a string of one or more punctuation characters. The punctuation characters are shown in Figure 4.

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

Figure 4. Separator Characters

The following rules for the formation of separators apply (brackets enclose and identify each separator discussed in the following list):

- A space [ ] is always a separator, except as noted below. Anywhere a space is used as a separator, more than one space may be used.

- A space separator can immediately precede all separators except:

  - As specified in standard format rules (see the chapter on Standard COBOL Format).

  - The separator closing quotation mark. (In this case, a preceding space is considered part of the nonnumeric literal and not as a separator.) (Note that at least one space separator <u>must</u> precede an opening pseudo-text delimiter [==]; the space is not optional.)

- A space separator can immediately follow any separator except the opening quotation mark. (In this case, a following space is considered part of the nonnumeric literal and not as a separator.)

- The comma [,], semicolon [;], and period [.], when immediately followed by a space, are separators. These separators may appear only where explicitly allowed by COBOL rules. (See Overall Punctuation Rules below and Format Notation under the General Description chapter.)

- The left and right parentheses [(and)] are separators: Parentheses must appear as balanced pairs of left and right parentheses, delimiting subscripts, indexes, arithmetic expressions, or conditions.

- The quotation mark ["] is a separator. An opening quotation mark must be immediately preceded by a space or a left parenthesis. A closing quotation mark must be immediately followed by one of the following separators: space, comma, semicolon, period, or right parenthesis. Except when the literal is continued (see Continuation of Lines in the Standard COBOL Format chapter), quotation marks must appear as balanced pairs delimiting nonnumeric literals.

Note: For the quotation mark as previously implemented, see Appendix A. → page 406

- The pseudo-text delimiter [==] is a separator. 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 following separators: space, comma, semicolon, or period. Pseudo-text delimiters must appear as balanced pairs delimiting pseudo-text. (Further information is given in the description of the COPY Statement in the Source Program Library chapter.)

OVERALL PUNCTUATION RULES

Any punctuation character included in a PICTURE character-string, a comment character-string, or a nonnumeric literal is not considered to be a punctuation character but rather is considered to be part of the character-string or literal.

Punctuation rules for each division of the COBOL source program follow.

Identification Division

Commas and semicolons can be used in the comment-entries. Each paragraph must end with a period followed by a space.

Environment Division

Commas or semicolons may separate successive clauses and successive operands within clauses. The SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, and I-O-CONTROL paragraphs must each end with a period followed by a space. In the FILE-CONTROL paragraph, each File-Control entry must end with a period followed by a space.

Data Division

Commas or semicolons may separate successive clauses and operands within clauses. File (FD), Sort/Merge file (SD), and Communication Description (CD) entries, and data description entries must each end with a period followed by a space.

IBM Extension

Each report (RD) entry must end with a period followed by a space.

End of IBM Extension

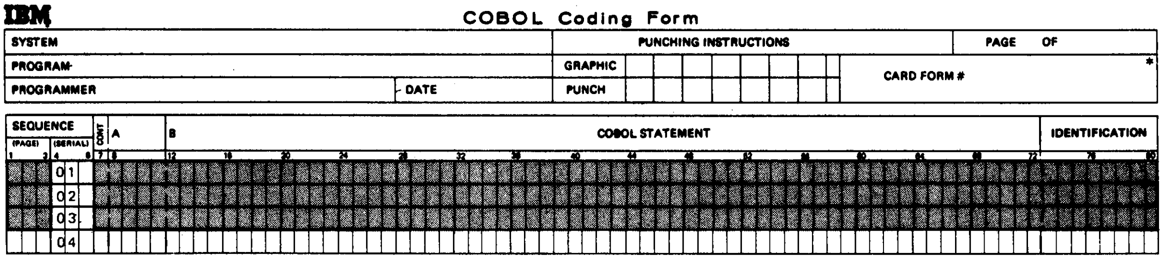
Procedure Division

Commas or semicolons may separate successive statements within a sentence, and successive operands within a statement. Each sentence and each procedure must end with a period followed by a space.

STANDARD COBOL FORMAT

COBOL programs must be written in standard COBOL format, as shown in Figure 5. The format is described in terms of an 80-character line (card image format). Source programs are written in standard COBOL format, and the output listing of the source program is printed in the same format.

Note: When the LISTER feature is specified, the output listing may appear in a space-saving 2-column format.



IBM COBOL Coding Form

SYSTEM | PUNCHING INSTRUCTIONS | PAGE OF

PROGRAM | GRAPHIC | CARD FORM # | *

PROGRAMMER | DATE | PUNCH

SEQUENCE (PAGE) (SERIAL) | CONT. | A | B | COBOL STATEMENT | IDENTIFICATION

1 3 4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80

01

02

03

04

Columns 1-6 represent the sequence number area.
Column 7 is the continuation area.
Columns 8-11 represent Area A.
Columns 12-72 represent Area B.
Columns 73-80 are used to identify the program.

Note: Areas A and B are used for writing COBOL source programs.

Figure 5. COBOL Coding Form and Standard COBOL Format

Sequence Numbers

Sequence numbers are written in the sequence number area. A sequence number is used to numerically identify each card image to be compiled by the COBOL compiler. The use of sequence numbers is optional. If used, a sequence number must consist of six digits in the sequence number area, columns 1 through 6.

If sequence numbers are present in the source program, they must be in ascending order, or a diagnostic message is issued.

IBM Extension

However, this IBM implementation allows the user to suppress sequence checking at compile time.

End of IBM Extension

Continuation Area

The continuation area is used to indicate the continuation of words and nonnumeric literals from the previous line onto the current line, to specify debugging lines, or to indicate that the text on this card image is to be treated as a comment. (See the following paragraphs on Continuation Lines and Comment Lines.)

Area A and Area B

Area A occupies columns 8 through 11. Area B occupies columns 12 through 72. COBOL elements that may begin in Area A, and specific COBOL elements that may follow them, are shown in Figure 6. Additional rules are given in the following paragraphs.

Elements That May Begin in Area A	Must be Followed Immediately By	Placement of Following Elements
division header	(Procedure Division only) USING Option	same or next line (Area B)
	section header, paragraph header, paragraph-name, or (in Procedure Division) key word DECLARATIVES	next line (Area A)
section header	COPY or USE statement	same or next line (Area B)
	paragraph header or paragraph-name (after COPY or USE, if specified)	next line (Area A)
paragraph header or paragraph-name	Environment Division entry or Procedure Division	same or next line (Area B)
level indicator	data-name	same line (Area B)
	(Report Section only) optional data-name	same line (Area B)
key word DECLARATIVES.	Declaratives section name	next line (Area A)
key words END DECLARATIVES.	section-header	next line (Area A)

Figure 6. Sequence of Elements in Area A and Area B

DIVISION HEADER: A division header, except when a USING option is specified with a Procedure Division header, must be immediately followed by a period. Except for the USING option, no text may appear on the same line.

SECTION HEADER: A section header, except when Procedure Division priority numbers are specified, must be immediately followed by a period. In the Environment and Procedure Divisions, a section consists of paragraphs. In the Data Division, a section consists of Data Division entries.

PARAGRAPH HEADER, PARAGRAPH-NAME: In the Environment Division, a paragraph consists of a paragraph header followed in Area B by one or more entries. An entry consists of one or more clauses. In the Procedure Division, a paragraph consists of a paragraph-name followed in Area B by one or more sentences. A sentence consists of one or more statements; a statement is a syntactically valid combination of a COBOL verb and its operands. Entries and sentences must be followed by a period followed by a space.

Successive entries or sentences begin in Area B of either the same line as the last entry or sentence, or of the next succeeding nonblank, noncomment line.

DATA DIVISION ENTRIES: Each Data Division entry begins with a level indicator or level-number, followed by a space, followed in Area B by a data-name, optionally followed by a sequence of independent clauses describing the item. Each clause, except the last, is followed by a space (or, optionally, by a comma or semicolon followed by a space). The last clause in the entry must be immediately followed by a period followed by a space.

IBM Extension

In the Report Section, the data-name entry is optional.

End of IBM Extension

Successive clauses begin in Area B of either the same line as the preceding clause, or of the next succeeding nonblank noncomment line.

A level indicator (FD, SD, CD) must begin in Area A followed by a space. (See the Data Division Organization chapter.)

IBM Extension

In the Report Section, the level indicator (RD) must begin in Area A, followed by a space.

End of IBM Extension

A level number is a one- or two-digit integer, with one of the values 1 through 49, 66, 77, or 88. At least one space must follow the level-number. (See the Data Description chapter.)

Level-numbers 01 and 77 must begin in Area A, followed by a space.

Level-numbers 02 through 49, 66, and 88 may begin in either Area A or Area B.

INDENTATION: Within an entry or sentence, successive lines in Area B may have the same format, or be indented to clarify program logic. The output listing is thus indented only if the input card images are indented. Indentation does not affect the syntax of the program. The amount of indentation can be chosen by the programmer, subject only to the restrictions on the width of Area B.

Note: When the LISTER feature is used, the output listing appears with standard LISTER indentation, no matter what indentation the source uses.

DECLARATIVES AND END DECLARATIVES: In the Procedure Division, the key words DECLARATIVES and END DECLARATIVES begin and end the declaratives portion of the source program. Each must begin in Area A followed immediately by a period; no other text may appear on the same line. After the key words END DECLARATIVES, no text may appear before the following section header. (See the Declaratives chapter.)

Continuation of Lines

Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next succeeding noncomment line. The line being continued is a continued line; the succeeding lines are continuation lines. Area A of a continuation line must contain only spaces.

If there is no hyphen (-) in the continuation area (column 7) of a line, the last character of the preceding line is assumed to be followed by a space.

If there is a hyphen in the continuation area of a line, the first nonblank character of this continuation line immediately follows the last nonblank character of the continued line without an intervening space.

If the continued line contains a nonnumeric literal without a closing quotation mark, all spaces at the end of the continued line (through column 72) are considered to be part of the literal. The continuation line must contain a hyphen in the continuation area, and the first nonblank character must be a quotation mark. The continuation of the literal begins with the character immediately following the quotation mark.

Comment Lines

A comment line is any line with an asterisk (*) or slash (/) in the continuation area (column 7) of the line. A comment line may be placed anywhere in the program following the Identification Division header. The comment may be written anywhere in Area A and Area B of that line, and may consist of any combination of characters from the EBCDIC set.

If the asterisk (*) is placed in the continuation area, this comment line is printed in the output listing immediately following the last preceding line.

If the slash (/) is placed in the continuation area, the current page of the output listing is ejected and the comment line is printed on the first line of the next page.

The asterisk or slash and the comment are produced only on the output listing. They are treated as documentation by the compiler.

Successive comment lines are allowed. Each must begin with the appropriate character in the continuation area.

Note: Rules for the formation of debugging lines are given in the Debugging Features chapter.

Blank Lines

Blank lines contain nothing but spaces from column 7 through column 72, inclusive. Except immediately preceding a continuation line, a blank line may appear anywhere in a program.

METHODS OF DATA REFERENCE

Every user-specified name defining an element in a COBOL program must be unique—either because no other name has a character-string of the same value, or because it can be made unique through qualification, subscripting, or indexing. In addition, references to data and procedures can be either explicit or implicit. This chapter gives the rules for qualification and for explicit and implicit references.

QUALIFICATION

A name can be made unique if it exists within a hierarchy of names, and the name can be identified by specifying one or more higher-level names in the hierarchy. The higher-level names are called _qualifiers_, and the process by which such names are made unique is called _qualification_.

Qualification is specified by placing after a user-specified name one or more phrases, each made up of the word OF or IN followed by a qualifier. (OF and IN are logically equivalent.)

Format 1—References to Data Division Names

```
┌ data-name-1    ┐   ┌ OF ┐
<                > [ <    > data-name-2 ] ...
└ condition-name ┘   └ IN ┘
```

Format 2—References to Procedure Division Names

```
                 ┌ OF ┐
paragraph-name [ <    > section-name]
                 └ IN ┘
```

Format 3—References to COPY Libraries

```
            ┌ OF ┐
text-name [ <    > library-name]
            └ IN ┘
```

In Data Division references, all qualifying data-names must be associated with a level-indicator or level-number. Therefore, two identical data-names must not appear as subordinate entries in a group item unless they can be made unique through qualification. Names associated with a level-indicator (FD, SD, CD, or RD) are the highest level in the hierarchy. Next highest are those associated with level-number 01. Names associated with level-numbers 02 through 49 are at successively lower levels in the hierarchy.

IBM Extension

In the Report Section, a report-name is the only available qualifier for report group description data-names and sum counters.

End of IBM Extension

In the Procedure Division, two identical paragraph-names must not appear in the same section. A section-name is the highest (and only) qualifier available for a paragraph-name.

In any hierarchy, the name associated with the highest level must be unique, and cannot be qualified. No matter what qualification is available, no name can be both a data-name and a procedure-name.

Enough qualification must be specified to make the name unique; however, it may not be necessary to specify all the levels of the hierarchy. For example, if there is more than one file whose records contain the field EMPLOYEE-NO, but only one of the files has a record named MASTER-RECORD, EMPLOYEE-NO OF MASTER-RECORD sufficiently qualifies EMPLOYEE-NO. EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE is valid but unnecessary.

Qualification Rules

The following rules for qualification apply:

- Each qualifier must be of a successively higher level, and must be within the same hierarchy as the name it qualifies.
- The same name must not appear at two levels in a hierarchy.
- If a data-name or condition-name is assigned to more than one data item, it must be qualified each time it is referred to (for the one exception, see the REDEFINES clause description in the Data Description chapter).
- A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.
- When it is being used as a qualifier, a data-name cannot be subscripted.
- A name can be qualified even though it does not need qualification.
- If there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.
- No duplicate section-names are allowed.
- No data-name can be the same as a section-name or a paragraph-name.
- Duplication of data-names must not occur in those places where the data-names cannot be made unique by qualification.
- The complete list of qualifiers for one data-name must not be the same as a partial list of qualifiers for another.

SUBSCRIPTING AND INDEXING

Subscripts and indexes can be used only when reference is made to an individual element within a table of elements that have not been assigned individual data-names. Subscripting and Indexing are explained in the chapter on Table Handling.

EXPLICIT AND IMPLICIT REFERENCES

COBOL source program references can be either explicit or implicit in three instances: data attribute specification, Procedure Division data references, and transfers of control.

DATA ATTRIBUTE SPECIFICATION

Explicit attributes are specified in actual COBOL coding.

If a data attribute is not an explicit attribute—that is, has not been specified in actual COBOL coding—it takes on a default value (one that the compiler assumes when an explicit specification is omitted). These default values are implicit attributes.

For example, the ACCESS MODE clause in the File-Control entry need not be specified; if it is omitted, the default is ACCESS MODE IS SEQUENTIAL, which is the implicit attribute. If, however, ACCESS MODE IS SEQUENTIAL is specified in COBOL coding, then it becomes an explicit attribute. (See the File-Control entry description in the Environment Division—Input-Output Section chapter.)

PROCEDURE DIVISION DATA REFERENCES

Procedure Division statements can refer to data items either explicitly or implicitly.

An explicit reference occurs when the data-name of the item is written in a COBOL statement, or when the data-name is copied into the program through a COPY statement. An implicit reference occurs when the data name is referred to by a COBOL statement without the name being written in that statement. For example, when a USE AFTER EXCEPTION/ERROR procedure for INPUT files is specified, there is an implicit reference to each file-name that identifies an input file. (See the description of the EXCEPTION/ERROR Declarative in the Declaratives chapter.)

TRANSFERS OF CONTROL

In the Procedure Division, program flow transfers control from statement to statement in the order in which they are written, unless there is an explicit control transfer or there is no next executable statement. (See Note below.) This normal program flow is an implicit transfer of control.

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

- After execution of the last statement of a procedure being executed under control of another COBOL statement. (COBOL statements that control procedure execution are: MERGE, PERFORM, SORT, and USE.)

- During SORT or MERGE statement execution, when control is transferred to any input or output procedure.

- During execution of any COBOL statement that causes execution of a Declarative procedure.

- At the end of execution of any Declarative procedure.

COBOL also provides explicit control transfers through the execution of any procedure branching or conditional statement. (Lists of procedure branching and conditional statements are given in the Procedure Division Structure chapter.)

Note: The term "next executable statement" refers to the next COBOL statement to which control is transferred according to the rules given above. Note that there is no next executable statement following:

- The last statement in a Declarative procedure that is not being executed under control of another COBOL statement.

- The last statement in a COBOL program when the procedure in which it appears is not being executed under control of another COBOL statement.

PART 2. IDENTIFICATION AND ENVIRONMENT DIVISIONS

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION—CONFIGURATION SECTION
- ENVIRONMENT DIVISION—INPUT-OUTPUT SECTION

IDENTIFICATION DIVISION

The Identification Division must be the first division in every COBOL source program. This division names the source program and the object program.

A source program is the initial COBOL program. An object program is the output from a compilation.

The user may also include in the Identification Division the date the program was written, the date of compilation, and other such documentary information about the program.

Format

```
┌ IDENTIFICATION DIVISION. ┐
│ ┌──────────────┐         │
< │ ID DIVISION. │          >
│ └──────────────┘         │
└                          ┘

PROGRAM-ID. program-name.

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

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

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

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

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

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

IBM Extension

However, this IBM implementation accepts the abbreviation ID DIVISION as a substitute for the standard Division header.

End of IBM Extension

The first paragraph of the Identification Division must be the PROGRAM-ID paragraph. The other paragraphs are optional, but when written must appear in the order shown in the format.

IBM Extension

However, this IBM implementation accepts the optional paragraphs in any order.

End of IBM Extension

The comment-entries in the optional paragraphs may be any combination of characters from the EBCDIC set, and may be written in Area B on one or more lines. Use of the hyphen in the continuation area is not permitted.

PROGRAM-ID PARAGRAPH

The PROGRAM-ID paragraph specifies the name by which the program is known to the system.

The PROGRAM-ID paragraph must be the first paragraph in the Identification Division.

Program-name is a user-defined word that identifies the object program to the system. The system uses the first 8 characters of program-name as the identifying name of the program; these first 8 characters, therefore, should be unique among program-names.

The system expects the first character of program-name to be alphabetic; if it is numeric, it is converted as follows:

- 0 to J
- 1 through 9 to A through I

The system does not include the hyphen as an allowable character; therefore, if the hyphen is the second through eighth character, it is converted to 0.

See the System Dependencies chapter for further information.

DATE-COMPILED PARAGRAPH

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

When a comment-entry is specified, the entire entry is replaced with the current date, even if the entry spans lines.

When the comment-entry is omitted, the compiler adds the current date to the line on which DATE-COMPILED is printed.

OTHER OPTIONAL PARAGRAPHS

The other paragraphs in the Identification Division may be included at the user's choice.

The comment-entries serve only as documentation, and do not affect the syntax of the program.

ENVIRONMENT DIVISION—CONFIGURATION SECTION

The Configuration Section describes the computer on which the source program is compiled, the computer on which the object program is executed, and, optionally, relates IBM-defined function-names to user-defined mnemonic names, specifies the collating sequence to be used, specifies a substitution for the currency sign, and/or interchanges the functions of the comma and the period.

IBM Extension

The Configuration Section is optional in a COBOL source program.

End of IBM Extension

In the Configuration Section, the comma or semicolon may, optionally, be used to separate successive clauses within a paragraph. In each paragraph there must be one and only one period, placed immediately after the last entry in the paragraph.

Format

```
  ENVIRONMENT DIVISION.

    [ CONFIGURATION SECTION.

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

    OBJECT-COMPUTER.   computer-name

                                 [ WORDS      ]
      [MEMORY SIZE integer     < CHARACTERS > ]
                                 [ MODULES    ]

      [PROGRAM COLLATING SEQUENCE IS alphabet-name]

      [SEGMENT-LIMIT IS priority-number].

  [SPECIAL-NAMES.

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

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

  [      ON STATUS IS condition-name-1]          ]
  |          [OFF STATUS IS condition-name-2]    |
  <                                              >  ] ... ]
  |      OFF STATUS IS condition-name-2          |
  [          [ON STATUS IS condition-name-1]     ]

  [alphabet-name IS

 [ STANDARD-1                                          ]
 |                                                     |
 | NATIVE                                              |
 |                [ [THROUGH]                 ]        |
 | literal-1      | <       >  literal-2      |        |
 |                | [THRU   ]                 |        |
 |                |                           |        |
 |                |  ALSO literal-3           |        |
 <                [     [ALSO literal-4] ...  ]        >  ] ...
 |                                                     |
 |                    [ [THROUGH]                 ]    |
 |                    | <       >  literal-6      |    |
 |   [literal-5       | [THRU   ]                 ]... |
 |                    |                           |    |
 |                    |  ALSO literal-7           |    |
 |                    [     [ALSO literal-8] ...  ]    |
 [                                                     ]

    [CURRENCY SIGN IS literal-9]

    [DECIMAL-POINT IS COMMA].  ]
```

SOURCE-COMPUTER PARAGRAPH

The SOURCE-COMPUTER paragraph describes the computer on which the source program is to be compiled.

Computer-name is a system-name in the form IBM-370 [-model-number].

The WITH DEBUGGING MODE clause is described in the Debugging Features chapter.

Except for the WITH DEBUGGING MODE clause, the SOURCE-COMPUTER paragraph is treated as documentation.

OBJECT-COMPUTER PARAGRAPH

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

Computer-name must be the first entry in the OBJECT-COMPUTER paragraph. Computer-name is a system-name in the form IBM-370 [-model-number].

The MEMORY SIZE clause can be used to document the amount of main storage needed to run the object program.

The PROGRAM COLLATING SEQUENCE clause specifies that the collating sequence used in this program (and this program only) is the collating sequence associated with the specified alphabet-name, which must be defined in the SPECIAL-NAMES paragraph. The program collating sequence is used to determine the truth value of the following nonnumeric comparisons:

- Those explicitly specified in relation conditions.
- Those explicitly specified in condition-name conditions.

IBM Extension

- Those implicitly specified by a CONTROL clause in an RD entry.

End of IBM Extension

The PROGRAM COLLATING SEQUENCE clause also applies to any nonnumeric merge or sort keys (unless the COLLATING SEQUENCE option is specified in the MERGE or SORT statement).

When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used. (See Appendix F for the complete EBCDIC collating sequence.)

The SEGMENT-LIMIT clause is described in the Segmentation chapter.

Except for the PROGRAM COLLATING SEQUENCE and SEGMENT-LIMIT clauses, the OBJECT-COMPUTER paragraph is treated as documentation.

Note: See Appendix B for ASCII considerations.

SPECIAL-NAMES PARAGRAPH

The SPECIAL-NAMES paragraph relates IBM-specified function-names to user-specified mnemonic-names; it also specifies a collating sequence to be associated with an alphabet-name, a substitute character for the currency sign, and that the functions of the comma and decimal point are to be interchanged in PICTURE clauses and numeric literals.

FUNCTION-NAME-1

Function-name-1 specifies system devices or standard system actions taken by the compiler.

The associated mnemonic-name is required; it follows the rules of formation for a user-specified name, and at least one character must be alphabetic. The mnemonic-name can be used in ACCEPT, DISPLAY, and WRITE statements. See the System Dependencies chapter for a list of valid function-names.

FUNCTION-NAME-2

Function-name-2 defines a one-byte program switch. Function-name-2 may be defined as UPSI-0 through UPSI-7. (Each UPSI is a User Program Status Indicator switch.) At least one condition-name must be associated with each specified UPSI switch.

Each condition-name follows the rules of formation for a user-specified name; at least one character must be alphabetic. The value associated with the condition-name is considered to be alphanumeric.

In the procedure division, the status of the UPSI switch is tested through the associated condition-name(s). Each condition-name is the equivalent of a level-88 item; the associated mnemonic-name, if specified, is considered the conditional variable and can be used for qualification. (See Switch Status Condition description in the chapter on Conditional Expressions.)

See the System Dependencies chapter for a further discussion.

ALPHABET-NAME CLAUSE

Provides a means of relating an alphabet-name to a specified character code set or collating sequence.

When the alphabet-name is specified in the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph, or in the COLLATING SEQUENCE option of the sort or merge statement, it specifies a collating sequence.

When the alphabet-name is specified in the FD entry CODE-SET clause, it specifies a character code set.

When STANDARD-1 is specified, the code set or collating sequence to be used is the ASCII character set (see Appendix B, ASCII considerations).

When NATIVE is specified, the code set or collating sequence to be used is the EBCDIC character set. This is also the character set or collating sequence used when the alphabet-name clause is omitted.

Note: Both the EBCDIC and ASCII collating sequences are given in Appendix G.

When the literal option is specified, the alphabet-name must not be referred to in a CODE-SET clause (see CODE-SET clause description in the FD Entry chapter).

When the literal option is specified, the collating sequence to be used is specified by the programmer according to the following rules:

- The order in which literals appear specifies the ordinal number, in ascending sequence, of the character(s) in this collating sequence.
- Each numeric literal specified must be an unsigned integer, and must have a value from 1 through the maximum number of characters in the EBCDIC character set (256). The value of each literal specifies the ordinal number of a character within the EBCDIC character set. (That is, the literal 112 represents the EBCDIC character ?, the literal 234 represents the EBCDIC character Z, the literal 241 represents the EBCDIC character 0, and so forth.)
- Each character in a nonnumeric literal represents that actual character in the EBCDIC set. If the nonnumeric literal contains more than one character, each character,

starting with the leftmost, is assigned a successively ascending position within this collating sequence.

- Any EBCDIC characters not explicitly specified assume positions in this collating sequence higher than any of the explicitly specified characters. The relative order of the unspecified characters within the EBCDIC set remains unchanged.

- Within one alphabet-name clause, a given character must not be specified more than once.

- Each nonnumeric literal associated with a THROUGH or ALSO option must be one character in length.

- When the THROUGH option is specified, the contiguous EBCDIC characters beginning with the character specified by literal-1 and ending with the character specified by literal-2 are assigned successively ascending positions in this collating sequence. This sequence may be either ascending or descending within the original EBCDIC sequence. (That is, if "Z" through "A" is specified, left-to-right, for this collating sequence the ascending values, left-to-right, for the capital letters are ZYXWVUTSRQPONMLKJIHGFEDCBA.)

- When the ALSO option is specified, the EBCDIC characters specified as literal-1, literal-3, literal-4, and so forth, are assigned to the same position in this collating sequence. (That is, if "D" ALSO "N" ALSO 112 ALSO "%" is specified, for this collating sequence the characters D, N, ?, and % are all considered to be in the same position in the collating sequence.)

- The character having the highest ordinal position in this collating sequence is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position, due to specification of the ALSO option, the last character specified is considered to be the HIGH-VALUE character for procedural statements such as DISPLAY, or as the sending field in a MOVE statement. (If the ALSO option example given above were specified as the high-order characters of the collating sequence, the HIGH-VALUE character would be %.)

- The character having the lowest ordinal position in this collating sequence is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position, due to specification of the ALSO option, the first character specified is the LOW-VALUE character. (If the ALSO option example given above were specified as the low-order characters of the collating sequence, the LOW-VALUE character would be D.)

Alphabet-Name Clause Examples

The following examples illustrate some uses for the alphabet-name clause.

If PROGRAM COLLATING SEQUENCE IS USER-SEQUENCE, and the <u>alphabet-name</u> clause is specified as USER-SEQUENCE IS "D," "E," "F," and two Data Division items are defined as follows:

```
77  ITEM-1 PIC X(3) VALUE "ABC".
77  ITEM-2 PIC X(3) VALUE "DEF".
```

Then the comparison IF ITEM-1 > ITEM-2 is <u>true</u>.

(Characters D, E, F are in ordinal positions 1, 2, 3 of this collating sequence. Characters A, B, C are in ordinal positions 197, 198, 199 of this collating sequence.)

If the alphabet-name clause is specified as USER-SEQUENCE is 1, 2, 3, 4, 5, 6, 10, 11 ... 254, 255, "7," ALSO "8," ALSO "9" (and all 255 EBCDIC characters have been specified), and two Data Division items are specified as follows:

```
77  ITEM-1 PIC X(3) VALUE HIGH-VALUE.
77  ITEM-2 PIC X(3) VALUE "787".
```

Then both of the following comparisons are <u>true</u>:

```
IF ITEM-1 = ITEM-2...
IF ITEM-2 = HIGH-VALUE...
```

(They compare as <u>true</u> because the values 7, 8, and 9 all occupy the same position in this USER-SEQUENCE collating sequence.)

If the alphabet-name clause is specified as USER-SEQUENCE IS "E," "D," "F," and a table in the Data Division is defined as follows:

```
05 TABLE A OCCURS 5 ASCENDING KEY IS KEY-A
     INDEXED BY INX-A.
   10 FIELD-A ... .
   10 KEY-A   ... .
```

If the contents in ascending sequence of each occurrence of KEY-A are A, B, C, D, E, G, the results of the execution of a SEARCH ALL statement for this table will be invalid, since the contents of KEY-A are not in ascending order. (The proper ascending order would be E, D, A, B, C, G.)

CURRENCY SIGN CLAUSE

Literal-9, which appears in the CURRENCY SIGN clause, represents the currency symbol in the PICTURE clause. Literal-9 must be a one-character nonnumeric literal, and must not be any of the following characters:

- Digits 0 through 9
- Alphabetic characters A B C D L P R S V X Z or the space
- Special characters * + - / , . ; ( ) = "

Note: For the quotation mark as formerly implemented, see Appendix A. (page 406)

When the CURRENCY SIGN clause is omitted, only the dollar sign ($) may be used as the PICTURE symbol for the currency sign.

DECIMAL POINT IS COMMA CLAUSE

When the DECIMAL POINT IS COMMA clause is specified, the functions of the period and the comma are exchanged in PICTURE character strings and in numeric literals.

PROGRAMMING NOTES

User Program Status Indicator (UPSI) switches are useful for processing special conditions within a program—such as year-beginning or year-ending processing. At the beginning of the Procedure Division, an UPSI switch can be tested; if it is ON, the special branch is taken. An example of this use of an UPSI switch is given in the Report Writer sample program.

The PROGRAM COLLATING SEQUENCE clause in conjunction with the alphabet-name clause can be used to specify EBCDIC nonnumeric comparisons for an ASCII-encoded tape file, or ASCII nonnumeric comparisons for an EBCDIC-encoded tape file.

The literal option of the alphabet-name clause can be used to process internal data in a collating sequence other than NATIVE or STANDARD-1.

ENVIRONMENT DIVISION—INPUT-OUTPUT SECTION

The Input-Output Section defines each file, identifies its external storage medium, assigns the file to one or more input/output devices, and specifies information needed for efficient transmission of data between the external medium and the COBOL program.

The Input-Output Section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files with the external media; and the I-O-CONTROL paragraph, which defines special input/output techniques to be used.

General Format—Input-Output Section

```
[INPUT-OUTPUT SECTION.

 [ FILE-CONTROL. file-control entry

   [file-control entry] ...  ]

[ I-O-CONTROL.  input-output-control entry] ]
```

The exact contents of the Input-Output Section depend on the file organization and access methods used. The following summary gives some background in the file processing techniques available in COBOL.

FILE PROCESSING SUMMARY

The method used to process a file in a COBOL program depends on its data organization and on the access mode used.

Data organization is the permanent logical structure of the file, established at the time the file is created. Three types of data organization are available: sequential, indexed, and relative.

An access mode is the manner in which records in a file are to be processed. Three access modes are available: sequential, random, and dynamic.

The following paragraphs describe both the types of data organization available, and access modes.

DATA ORGANIZATION

In a COBOL program, data organization can be sequential, indexed, or relative.

VSAM or Physical Sequential Organization

With this organization, each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. The predecessor-successor relationships are established by the physical order in which records are placed in the file at file creation time. Once established, these relationships do not change, except that a file can be extended. Records can be fixed or variable in length; there are no keys.

Physical sequential and VSAM sequential files have sequential organization.

VSAM Indexed Organization

With this organization, each record in the file has one or more embedded keys, each of which is associated with an index. Each index provides a logical path to the data records according to the contents of the associated embedded record key data items. Indexed files must be direct access storage files. Records can be fixed or variable in length.

Each record in an indexed file must have an embedded prime record key. When records are inserted, updated, or deleted they are identified solely by the values of their prime record keys. Consequently, the value in each prime record key data item must be unique and must not be changed when the record is updated.

In addition, each record in an indexed file may contain one or more embedded alternate record keys. Each alternate key provides an alternative access path for record retrieval. The programmer can specify that the values contained in an alternate record key need not be unique.

The key used for any specific input/output request is known as the <u>key of reference</u>.

VSAM indexed files have indexed organization.

VSAM Relative Organization

With this organization, each record in the file is identified by its relative record number. The file can be thought of as a serial string of areas, each of which may contain one logical record. Each of these areas is identified by a relative record number; record storage and retrieval are based on this number. For example, the first record area is addressed by relative record number 1, and the 10th is addressed by relative record number 10, whether or not records have been written in the second through ninth record areas. Relative files must be direct access storage files. Records can be fixed or variable in length.

VSAM relative files have relative organization.

ACCESS MODES

Three access modes are available in COBOL: sequential, random, and dynamic.

<u>Sequential access mode</u> allows reading and writing records of a file in a serial manner; the order of reference is implicitly determined by the position of a record in the file.

<u>Random access mode</u> allows reading and writing records in a programmer-specified manner; the control of successive references to the file is expressed by specifically defined keys supplied by the user.

<u>Dynamic access mode</u> allows a specific input/output request to determine the access mode. Therefore, records may be processed sequentially and/or randomly.

Sequential Files

Files with sequential organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the records were originally written.

VSAM Indexed Files

All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order of the prime record key or alternate record key value. The order of retrieval of records within a set of records having duplicate alternate 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 prime record key or alternate record key in that record key data item. If a set of records has duplicate alternate record key values, only the first record written is available.

In the dynamic access mode, the programmer may change as necessary from sequential access to random access using appropriate forms of input/output statements.

VSAM Relative Files

All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records which currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in a RELATIVE KEY data item; the RELATIVE KEY must not be defined within the record description entry for this file.

In the dynamic access mode, the programmer may change as necessary from sequential access to random access using appropriate forms of input/output statements.

FILE-CONTROL PARAGRAPH

The FILE-CONTROL paragraph associates each file in the COBOL program with an external medium, and allows specification of file organization, access mode, and other information.

Format 1—Sequential File Entries

```
FILE-CONTROL.

    SELECT [OPTIONAL] file-name

    ASSIGN TO assignment-name-1 [assignment-name-2] ...

                        ┌       ┐
                        │ AREA  │
   [RESERVE integer     │       │ ]
                        │ AREAS │
                        └       ┘

      [ORGANIZATION IS SEQUENTIAL]   assumed if omitted

      [ACCESS MODE IS SEQUENTIAL]    assumed if omitted
  ┌─────────────────────────────┐
  │[PASSWORD IS data-name-1]    │
  └─────────────────────────────┘

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

Note: PASSWORD is only for VSAM sequential files.

Format 2—VSAM Indexed File Entries

```
FILE-CONTROL.

    SELECT file-name
    ASSIGN TO assignment-name [assignment-name-2] ...

                      [ AREA  ]
   [RESERVE integer   |       |   ]
                      [ AREAS ]

    ORGANIZATION IS INDEXED

                      [ SEQUENTIAL ]
  [ACCESS MODE IS     < RANDOM     >   ]
                      | DYNAMIC    |
                      L            J

   RECORD KEY IS data-name-3

       +---------------------------+
       | [PASSWORD IS data-name-1] |
       +---------------------------+

  [ALTERNATE |[| RECORD |]| KEY IS data-name-4

       +---------------------------+
       | [PASSWORD IS data-name-5] |
       +---------------------------+

      [WITH DUPLICATES] ] ...

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

Format 3—VSAM Relative File Entries

```
FILE-CONTROL.

    SELECT file-name

    ASSIGN TO assignment-name-1 [assignment-name-2] ...

                       [ AREA  ]
    [RESERVE integer   |       |  ]
                       [ AREAS ]

    ORGANIZATION IS RELATIVE

                       [ SEQUENTIAL [RELATIVE KEY IS data-name-6] ]
                       |                                          |
                       | [ RANDOM  ]                              |
    [ACCESS MODE IS    < <         > RELATIVE KEY IS data-name-7  > ]
                       | | DYNAMIC |                              |
                       L L         J                              J

   +---------------------------+
   | [PASSWORD IS data-name-1] |
   +---------------------------+

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

Each file described in an FD or SD entry in the Data Division must be described in one and only one entry in the File-Control paragraph.

The key word FILE-CONTROL may appear only once, at the beginning of the File-Control paragraph. The word FILE-CONTROL must begin in Area A, and be followed by a period followed by a space.

Each FILE-CONTROL entry must begin in Area B with a SELECT clause. The order in which other clauses appear is not significant.

IBM Extension

There is one exception to this rule. For indexed files, the PASSWORD clause, if specified, must immediately follow the RECORD KEY or ALTERNATE RECORD KEY data-name with which it is associated. (In the ALTERNATE RECORD KEY clause, the key word RECORD is optional.)

End of IBM Extension

Optionally, each clause within a File-Control entry may be separated from the next by a comma or semicolon followed by a space. Each File-Control entry ends with a period followed by a space.

Each data-name must appear in a Data Division data description entry; each must be fixed length, and be at a fixed position from the beginning of the record in which it appears. Each data-name may be qualified; it must not be subscripted or indexed.

SELECT CLAUSE

Each file-name specified in a SELECT clause must have an FD or SD entry in the Data Division. A file-name must conform to the rules for a COBOL user-defined name, must contain at least one alphabetic character, and must be unique within this program.

Format 1 Considerations

The OPTIONAL phrase may be specified only for input files with sequential organization. It must be specified for those input files that are not necessarily present each time the object program is executed.

For the SELECT OPTIONAL clause as previously implemented, see Appendix A. For Sort/Merge considerations, see the chapter on the Sort/Merge feature. For ASCII considerations, see Appendix B.

ASSIGN CLAUSE

The ASSIGN clause associates the file with an external medium. The assignment-name makes the association between the file and the external medium; it has the following formats:

• Physical sequential files	[comments-] [S-]name
• VSAM sequential files	[comments-]AS-name
• VSAM indexed or relative files	[comments-]name

The name field must conform to the rules for formation of a program-name.

See also the chapter on System Dependencies.

Any assignment-name after the first is treated as documentation.

RESERVE CLAUSE

See the chapter on System Dependencies for information.

ORGANIZATION CLAUSE

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

When the ORGANIZATION clause is omitted, ORGANIZATION IS SEQUENTIAL is assumed.

Format 1 Considerations

When ORGANIZATION IS SEQUENTIAL is specified or implied, a predecessor-successor relationship of the records in the file is established by the order in which records are placed in the file when it is created or extended.

Format 2 Considerations

When ORGANIZATION IS INDEXED is specified, the position of each logical record in the file is determined by indexes created with the file and maintained by the system. The indexes are based on embedded keys within the file's records.

Format 3 Considerations

When ORGANIZATION IS RELATIVE is specified, the position of each logical record in the file is determined by its relative record number.

ACCESS MODE CLAUSE

The ACCESS MODE clause defines the manner in which the records of the file are made available for processing.

When this clause is omitted, ACCESS IS SEQUENTIAL is assumed.

Format 1 Considerations

For files with sequential organization, records in the file are accessed in the sequence established when the file is created or extended. Whether ACCESS IS SEQUENTIAL is specified or omitted, sequential access is always assumed.

Format 2 Considerations

For files with indexed organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the sequence of ascending record key values within the currently used key of reference.

When ACCESS IS RANDOM is specified, the value placed in a record key data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

Format 3 Considerations

For files with relative organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the ascending sequence of relative record numbers of existing records in the file.

When ACCESS IS RANDOM is specified, the value placed in the RELATIVE KEY data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

PASSWORD CLAUSE

IBM Extension

The PASSWORD clause controls object-time access to the file.

Both data-name-1 and data-name-5 are password data items; each must be defined in the Working-Storage section as an alphanumeric item. The first 8 characters are used as the password; a shorter field is padded with blanks to 8 characters. Each password data item must be equivalent to one that is externally defined.

When the PASSWORD clause is specified, at object time the PASSWORD data item must contain the valid password for this file before the file can be successfully opened.

Format 1 Considerations

The PASSWORD clause is only valid for VSAM sequential files.

Format 2 Considerations

When the PASSWORD clause is specified, it must immediately follow the RECORD KEY or ALTERNATE RECORD KEY data-name with which it is associated.

See also the System Dependencies chapter.

End of IBM Extension

FILE STATUS CLAUSE

The FILE STATUS clause allows the user to monitor the execution of each input/output request for the file.

Data-name-2 is the status key data item. Data-name-2 must be defined in the Data Division as a two-character alphanumeric item. Data-name-2 must not be defined in the File Section.

IBM Extension

However, this IBM implementation allows data-name-2 to be defined as a 2-character external-decimal unsigned integer; it is treated as an alphanumeric item. It must not be defined as the object of an OCCURS DEPENDING ON clause. Also, data-name-2 must not be defined in the Report Section.

End of IBM Extension

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input/output request that explicitly or implicitly refers to this file. The value indicates the status of execution of the statement. (See "Status Key" in the Input/Output Statements chapter.)

page 323

RECORD KEY CLAUSE (FORMAT 2)

The RECORD KEY clause specifies the data item within the record that is the prime record key for an indexed file. The values contained in the prime record key data item must be unique among records in the file.

Data-name-3 is the prime RECORD KEY data item. It must be described as a fixed-length alphanumeric item within a record description entry associated with the file.

IBM Extension

However, this IBM implementation allows data-name-3 to be defined as an external-decimal unsigned integer; it is treated as an alphanumeric item.

End of IBM Extension

The data description of data-name-3 and its relative location within the record must be the same as the ones used when the file was defined.

ALTERNATE RECORD KEY CLAUSE (FORMAT 2)

The ALTERNATE RECORD KEY clause specifies a data item within the record that provides an alternative path to the data in an indexed file.

Data-name-4 is an ALTERNATE RECORD KEY data item. It must be described as a fixed-length alphanumeric item.

IBM Extension

However, this IBM implementation allows data-name-4 to be defined as an external-decimal unsigned integer; it is treated as an alphanumeric item.

End of IBM Extension

The data description of data-name-4 and its relative location within the record must be the same as the ones used when the file was defined.

The leftmost character position of data-name-4 must not be the same as the leftmost character position of the RECORD KEY or of any other ALTERNATE RECORD KEY.

If the DUPLICATES option is not specified, the values contained in the ALTERNATE RECORD KEY data item must be unique among records in the file.

If the DUPLICATES option is specified, the values contained in the ALTERNATE RECORD KEY data item may be duplicated within any records in the file. In sequential access, the records with duplicate keys are retrieved in the order they were placed in the file. In random access, only the first record written of a series of records with duplicate keys can be retrieved.

RELATIVE KEY CLAUSE (FORMAT 3)

The RELATIVE KEY clause specifies the relative record number for a specific logical record within a relative file.

Data-name-6 or data-name-7 is the RELATIVE KEY data item. It must be defined as an unsigned integer data item, and must not be defined in a record description entry associated with this relative file. That is, the RELATIVE KEY is not part of the record.

When ACCESS IS SEQUENTIAL is specified, data-name-6, the RELATIVE KEY data item, need not be specified, unless the START statement is to be used. When the START statement is issued, the system uses the contents of the RELATIVE KEY data item to determine the record at which sequential processing is to begin.

If a value is placed in the RELATIVE KEY data item, and a START statement is not issued, the value is ignored and processing begins with the first record in the file.

When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, data-name-7, the RELATIVE KEY data item, must be specified. For each random processing request, the contents of the RELATIVE KEY data item are used to communicate a relative record number to the system.

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph specifies when checkpoints are to be taken, the storage areas to be shared by different files, and the location of files on a multiple file reel.

Format

Format 1—Physical Sequential Files

```
I-O-CONTROL.

   [RERUN ON assignment-name

                  ┌                          ┐
                  | integer-1 RECORDS        |
                  |                          |
                  |            ┌      ┐      |
         EVERY   <|            | REEL |      |> OF file-name-1] ...
                  | [END OF]  <        >     |
                  |            | UNIT |      |
                  |            └      ┘      |
                  └                          ┘

            ┌             ┐
   [SAME    | RECORD      |   AREA
            | SORT        |
            | SORT-MERGE  |
            └             ┘

    FOR file-name-2  ┌              ┐
                    <  file-name-3   > ... ] ...
                     └              ┘

  [MULTIPLE FILE TAPE CONTAINS

   file-name-4  [POSITION integer-2]

   [file-name-5  [POSITION integer-3]] ... ] ...
```

Format 2—VSAM Files

```
I-O-CONTROL.

   [RERUN ON assignment-name

      EVERY integer-1 RECORDS OF file-name-1] ...

               ┌              ┐
               │ RECORD       │
   [SAME       │ SORT         │  AREA
               │ SORT-MERGE   │
               └              ┘

      FOR file-name-2 {file-name-3} ... ] ...
```

The I-O-CONTROL paragraph is optional in a COBOL program.

The key word I-O-CONTROL may appear only once, at the beginning of the I-O-CONTROL paragraph. The word I-O-CONTROL must begin in Area A, and be followed by a period followed by a space.

IBM Extension

The order in which I-O-CONTROL paragraph clauses are written is not significant.

End of IBM Extension

Optionally, each clause within the I-O-CONTROL entry may be separated from the next by a comma or semicolon followed by a space. The I-O-CONTROL entry ends with a period followed by a space.

RERUN CLAUSE

The RERUN clause specifies that checkpoint records are to be taken.

More than one RERUN clause, subject to the restrictions given with each option, may be specified.

assignment-name specifies the external medium for the checkpoint file. It must not be the same assignment-name as that specified in any ASSIGN clause. It has the format:

```
[comments-][S-]name
```

that is, it must be a physical sequential file. It must reside on a tape or a direct access storage device.

Note: For ASCII considerations, see Appendix B.

Format 1 Considerations

This format is used when file-name-1 indentifies a physical sequential file. File-name-1 names the file for which checkpoint records are to be taken.

integer-1 RECORDS Option: This option specifies that a checkpoint record is to written for every integer-1 record of file-name-1 that are processed.

When multiple integer-1 RECORDS clauses are specified, no two of them may specify the same file-name-1.

END OF REEL/UNIT Option: This option specifies that a checkpoint record is to be written whenever end-of-volume for file-name-1 occurs. The terms REEL and UNIT are interchangeable.

When multiple END OF REEL/UNIT clauses are specified, no two of them may specify the same file-name-1.

Format 2 Considerations

When file-name-1 identifies a VSAM (sequential, indexed, or relative) only the integer-1 records option is valid. This option specifies that a checkpoint record is to be written for every integer-1 record of file-name-1 that is processed.

When multiple integer-1 RECORDS clauses are written, no two of them may specify the same file-name-1.

SAME CLAUSE

The SAME clause specifies that two or more files are to use the same main storage area during processing.

The files named in a SAME clause need not have the same organization or access.

The following discussion describes only the SAME RECORD AREA and SAME AREA options. The SAME SORT AREA and SAME SORT-MERGE area options are discussed in the chapter on the Sort-Merge Feature.

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage area for processing the current logical record. All of the files may be open at the same time. A logical record in the shared storage area is considered to be:

- A logical record of each opened output file in this SAME RECORD AREA clause, and

- A logical record of the most recently read input file in this SAME RECORD AREA clause.

For physical sequential files, if the SAME AREA clause does not contain the RECORD option, the area being shared includes all storage areas assigned to the files; therefore, it is not valid to have more than one of these files open at one time.

For VSAM files, the SAME AREA clause has the same meaning as the SAME RECORD AREA clause.

More than one SAME clause may be included in a program; however:

1. A specific file-name must not appear in more than one SAME AREA clause.

2. A specific file-name must not appear in more than one SAME RECORD AREA clause.

3. 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 that SAME RECORD AREA clause. However, that SAME RECORD AREA clause may contain additional file-names that do not appear in that SAME AREA clause.

4. The rule that in the SAME AREA clause only one file may be open at one time takes precedence over the SAME RECORD AREA rule that all the files may be open at the same time.

   See the System Dependencies chapter.

Programming Notes

Specification of the SAME AREA clause saves space in the object program; note, however, the restrictions when it is used.

Specification of the SAME RECORD AREA clause does not necessarily save space in the object program. Because the input/output areas of named files are identical, and all are available to the user, this clause allows transfer of data from one file to another with minimal data manipulation.

MULTIPLE FILE TAPE CLAUSE

The MULTIPLE FILE TAPE clause is treated as documentation; it specifies that two or more files share the same physical reel of tape. For further information, see the System Dependencies chapter.

PART 3. DATA DIVISION

- DATA DIVISION CONCEPTS
- FILE DESCRIPTION ENTRY
- DATA DESCRIPTION

DATA DIVISION CONCEPTS

The Data Division of a COBOL source program describes all the data to be processed by the object program. Two types of data can be processed: external data and internal data.

EXTERNAL DATA

External data is contained in files or, in communications applications, in messages. Messages are discussed in the Communications chapter; only files are discussed here.

A file is a collection of data records existing on some input/output device. A file can be thought of as a group of physical records; it can also be thought of as a group of logical records.

A physical record is a unit of data that is treated as an entity when moved into or out of auxiliary storage. The size of a physical record is determined by the particular input/output device on which it is stored. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A logical record is a unit of data whose subdivisions have a logical relationship. A logical record may itself be a physical record (that is, be contained completely on one physical unit of data); several logical records may be contained within one physical record, or one logical record may extend across physical records.

Data Division source language describes the relationship between physical and logical records.

The file description (FD) entry specifies the physical aspects of the data (such as the size relationship between physical and logical records, the size and name(s) of the logical record(s), labeling information, and so forth). See also, SD entry in the SORT/MERGE—Data Division section.

Record description entries, which follow the FD entry for a specific file, describe the logical records in the file, including the category and format of data within each field of the logical record, different values the data might be assigned, and so forth.

Once the relationship between physical and logical records has been established, only logical records are made available to the COBOL programmer. For this reason, unless the term "physical record(s)" is used, a reference in this manual to "records" means logical records.

INTERNAL DATA

The logic of the program may develop additional data within storage. Such data is called internal data.

The concept of logical records applies to both internal data and external data. Internal data can, therefore, be grouped into logical records, and be defined by a series of record description entries. Items that need not be so grouped can be defined in independent data description entries.

DATA RELATIONSHIPS

The relationships of all data to be used in a program are defined in the Data Division, through a system of level indicators and level-numbers.

A level indicator, together with its descriptive entry, identifies each file, or report description in a program. Level indicators are the highest level of any data hierarchy with which they are associated; they are described more fully in the next section, on Data Division Organization.

A level-number, together with its descriptive entry, indicates the properties of specific data. Level-numbers can be used to describe a data hierarchy; they can indicate that this data has a special purpose, and while they can be associated with—and subordinate to—level indicators, they can also be used independently to describe internal data or data common to two or more programs. The chapter on Data Description explains level-numbers.

DATA DIVISION ORGANIZATION

The Data Division is divided into sections. Each has a specific logical function within a COBOL source program, and each may be omitted from the source program when that logical function is not needed.

General Format—Data Division

```
DATA DIVISION.

[FILE SECTION.

[file-description entry

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

[WORKING-STORAGE SECTION.

┌                             ┐
│ data item description entry │
│                             │ ... ]
│ record-description entry    │
└                             ┘

[LINKAGE SECTION.

┌                             ┐
│ data item description entry │
│                             │ ... ]
│ record-description entry    │
└                             ┘

[COMMUNICATION SECTION.

[communication description entry

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

┌──────────────────────────────────────────────────────┐
│ [REPORT SECTION.                                      │
│                                                       │
│ [report description entry                             │
│                                                       │
│ [report-group description entry] ... ] ... ]          │
└──────────────────────────────────────────────────────┘
```

When written in the source program, the Data Division sections must appear in the order shown.

FILE SECTION

The File Section contains a description of all externally stored data (FD), and a description of each sort-merge-file (SD) used in the program.

The File Section must begin with the header FILE SECTION followed by a period. The File Section contains file description entries and sort-merge-file description entries, each one followed by its associated record description entry (or entries).

File Description Entries

In a COBOL program, the File Description Entries (beginning with the level indicators FD and SD) represent the highest level of organization in the File Section. The file description entry provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file. For the format and the clauses required in a file description entry, see the File Description Entry chapter.

For a complete discussion of the sort-file-description entry, see the chapter on the Sort/Merge feature.

Record Description Entry

The Record Description Entry consists of a set of data description entries which describe the particular record(s) contained within a particular file. For the format and the clauses required within the record description entry, see the Data Description chapter.

More than one record description entry may be specified; each is an alternative description of the same storage area.

Data areas described in the File Section should not be considered available for processing unless the file containing the data area is open.

WORKING-STORAGE SECTION

The Working-Storage Section may describe data records which are not part of external data files but are developed and processed internally.

The Working-Storage Section contains record description entries and data description entries for independent data items.

The Working-Storage Section must begin with the section header WORKING-STORAGE SECTION followed by a period.

Record Description Entries

Data entries in Working-Storage that have a definite hierarchic relationship to one another must be grouped into records structured by level number.

Data Item Description Entry

Independent items in Working-Storage that have no hierarchic 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 independent elementary items. Each is defined in a separate data item description entry that begins with the special level number 77.

LINKAGE SECTION

The Linkage Section describes data made available from another program.

Record description entries and data item description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved since the data area exists elsewhere. Any data description clause may be used to describe items in the Linkage Section, with one exception: The VALUE clause may not be specified for other than level-88 items.

For additional information, see the chapters on Subprogram Linkage and on System Dependencies.

COMMUNICATION SECTION

The Communication Section contains Communication Description entries for input and for output, and optional record description entries. The Communication Section is discussed in the Communication Feature chapter.

REPORT SECTION

IBM Extension

The Report Section contains report description (RD) entries and report group description entries for every report named in the REPORT clause. The Report Section is discussed in the Report Writer chapter.

End of IBM Extension

In a COBOL program, the file description entry (FD) is the highest level of organization in the File Section.

Format 1—Physical Sequential Files

```
FILE SECTION.

FD file-name
                                                ┌ CHARACTERS ┐
   [BLOCK CONTAINS [integer-1 TO] integer-2     <            >   ]
                                                | RECORDS    |
                                                └            ┘
   [RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]
           ┌ RECORD IS    ┐ ┌ STANDARD ┐
  LABEL    <              > <          >
           | RECORDS ARE  | | OMITTED  |
           └              ┘ └          ┘
                                    ┌ data-name-1 ┐
  [VALUE OF system-name-1 IS        <             >
                                    | literal-1   |
                                    └             ┘
                                ┌ data-name-2 ┐
      [system-name-2 IS         <             >  ] ... ]
                                | literal-2   |
                                └             ┘
         ┌ RECORD IS    ┐
  [DATA  <              > data-name-3 [data-name-4] ... ]
         | RECORDS ARE  |
         └              ┘
                 ┌ data-name-5 ┐
  [LINAGE IS     <             >  LINES
                 | integer-5   |
                 └             ┘
                                ┌ data-name-6 ┐
      [WITH FOOTING AT          <             >  ]
                                | integer-6   |
                                └             ┘
                          ┌ data-name-7 ┐
      [LINES AT TOP       <             >  ]
                          | integer-7   |
                          └             ┘
                             ┌ data-name-8 ┐
      [LINES AT BOTTOM       <             >  ]
                             | integer-8   |
                             └             ┘
 ┌──────────────────────────────────────────────────────────────┐
 │    ┌ REPORT IS    ┐                                          │
 │  [ <              >  report-name-1 [report-name-2] ... ]     │
 │    | REPORTS ARE  |                                          │
 │    └              ┘                                          │
 └──────────────────────────────────────────────────────────────┘

[CODE-SET IS alphabet-name].
```

Format 2—VSAM Files (Sequential, Indexed, Relative)

```
FILE SECTION.

FD  file-name

                                                 ┌ CHARACTERS ┐
   [BLOCK CONTAINS [integer-1 TO] integer-2      <            >   ]
                                                 | RECORDS    |
                                                 └            ┘

   [RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS]

         ┌ RECORD IS   ┐ ┌ STANDARD ┐
   LABEL <             > <          >
         | RECORDS ARE | | OMITTED  |
         └             ┘ └          ┘

                                      ┌ data-name-1 ┐
   [VALUE OF system name-1 IS         <             >
                                      | literal-1   |
                                      └             ┘

                                 ┌ data-name-2 ┐
       [system-name-2 IS         <             >   ] ... ]
                                 | literal-2   |
                                 └             ┘

          ┌ RECORD IS   ┐
   [DATA  <             >  data-name-3 [data-name-4] ... ] .
          | RECORDS ARE |
          └             ┘
```

The file description entry must begin with the level indicator FD followed by a space.

The clauses which follow file-name are optional in many cases; the order of their appearance is not significant.

The last clause in the FD entry must be immediately followed by a period followed by a space.

One or more record description entries must follow the FD entry. When more than one record description entry is specified, each entry implies a redefinition of the same storage area.

IBM Extension

There is one exception to the preceding rule: When the REPORT clause is specified, no record description entry need be specified.

The DATA RECORDS clause and the REPORT clause may both be specified for the same FD entry.

The LINAGE clause and the REPORT clause must not both be specified for the same FD entry.

End of IBM Extension

FILE-NAME

The file-name must follow the level indicator (FD), and must be the same as that specified in the associated File Control entry.

The file-name must follow the rules of formation for a user-defined word; at least one character must be alphabetic. The file-name must be unique within this program.

BLOCK CONTAINS CLAUSE

The BLOCK CONTAINS clause specifies the size of a physical record.

When neither the CHARACTERS nor RECORDS option is specified, the CHARACTERS option is assumed.

When the BLOCK CONTAINS clause is omitted, the compiler assumes that records are not blocked. The clause should be omitted if the records in the file are not blocked. Even if each physical record contains one and only one complete logical record, coding BLOCK CONTAINS 1 RECORD would result in fixed blocked records.

The BLOCK CONTAINS clause can be omitted when the associated File Control entry specifies a VSAM file; the concept of blocking has no meaning for VSAM files and the clause will be treated as a comment.

For all other types of files, the BLOCK CONTAINS clause is required.

Integer-1 and integer-2 must be nonzero unsigned integers. For use of BLOCK CONTAINS 0, see the System Dependencies chapter.

CHARACTERS Option

The CHARACTERS option is the default option.

When this option is specified or implied, the physical record size is specified as the number of character positions required to store the physical record, no matter what USAGE the characters within the data record have.

If only integer-2 is specified, it specifies the exact character size of the physical record. When integer-1 and integer-2 are both specified, they represent the minimum and maximum character size of the physical record, respectively.

Integer-1 and integer-2 must include any control bytes and padding (areas not contained within a logical record) contained in the physical record.

The CHARACTERS option must be used when:

- The physical record contains padding.
- Logical records are grouped so that an inaccurate physical record size could be implied. (For example, if the user describes a variable-length record of 100 characters, yet each time writes a block of 4, one 50-character record is written followed by three 100-character records. If the RECORDS option was specified, the compiler would calculate the block size as 420 characters instead of the actual size, 370 characters. This calculation includes block and record descriptors.)

See the System Dependencies chapter.

RECORDS Option

When this option is specified, the physical record size is expressed as the number of logical records contained in each physical record.

The compiler assumes that the block size must provide for integer-2 records of maximum size, and provides any additional space needed for control bytes.

RECORD CONTAINS CLAUSE

The RECORD CONTAINS clause specifies the size of a file's data records.

The RECORD CONTAINS clause is never required, because the size of each record is completely defined in the record description entries. When this clause is specified, the following notes apply:

- Integer-3 and integer-4 must be unsigned integers.
- When both integer-3 and integer-4 are specified, integer-3 specifies the size of the smallest data record, and integer-4 specifies the size of the largest data record.
- Integer-4 must not be specified alone unless all the records are the same size; in this case, integer-4 specifies the exact number of characters in the record.
- The record size must be specified as the number of character positions needed to store the record internally; that is, in terms of the number of bytes occupied internally by its characters, regardless of the number of characters used to represent the item within the record. The size of a record is determined according to the rules for obtaining the size of a group item. (See the descriptions of the USAGE clause and of Slack Bytes for further information.)

See the chapter on System Dependencies.

Programming Notes

When the RECORD CONTAINS clause is omitted, the record lengths are determined by the compiler from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable length item to calculate the record length.

LABEL RECORDS CLAUSE

The LABEL RECORDS clause specifies whether or not labels are present.

The LABEL RECORDS clause is required in every FD entry.

STANDARD Option

This option specifies that labels conforming to system specifications exist for this file.

This option may be specified for files assigned to magnetic tape devices. It must be specified for sequential disk files.

OMITTED Option

This option specifies that no labels exist for this file.

This option must be specified for files assigned to unit record devices. It may also be specified for files assigned to magnetic tape devices.

VSAM Considerations

For VSAM files, the LABEL RECORDS clause is treated as documentation. Label processing, therefore, is not performed.

VALUE OF CLAUSE

The VALUE OF clause serves only as documentation; it particularizes the description of an item in the label records associated with this file.

The syntax of this clause is accepted by the compiler and treated as documentation. In this implementation, no specific system-names are provided.

DATA RECORDS CLAUSE

The DATA RECORDS clause serves only as documentation for the names of data records associated with this file.

The specification of more than one data-name indicates that this file contains more than one type of data record, that is, two or more record descriptions for this file occupy the same storage area. These records need not have the same description or length. The order in which the data-names are listed is not significant.

Data-name-3 and data-name-4 are the names of record description entries associated with this file.

The DATA RECORDS clause is never required.

LINAGE CLAUSE

The LINAGE clause specifies the depth of a logical page in terms of number of lines; optionally, it also specifies the line number at which the footing area begins, in addition to the top and bottom margins of the logical page. (Note that there is not necessarily a relationship between the logical page size and the physical page size.)

The LINAGE clause may be specified only for physical sequential files, opened OUTPUT, or I-O.

All integers must be unsigned. All data-names must be described as unsigned integer data items.

LINAGE integer-5/data-name-5

Integer-5 or the value in data-name-5 specifies the number of lines that can be written and/or spaced on this logical page. The area of the page these lines represent is called the page body. The value must be greater than zero.

WITH FOOTING Option

Integer-6 or the value in data-name-6 specifies the first line number of the footing area within the page body. The footing line number must be greater than zero, and not greater than the last line of the page body. The footing area extends between those two lines.

LINES AT TOP Option

Integer-7 or the value in data-name-7 specifies the number of lines in the top margin of the logical page. The value may be zero.

LINES AT BOTTOM Option

Integer-8 or the value in data-name-8 specifies the number of lines in the bottom margin of the logical page. The value may be zero.

Figure 7 illustrates the use of each option of the LINAGE clause.

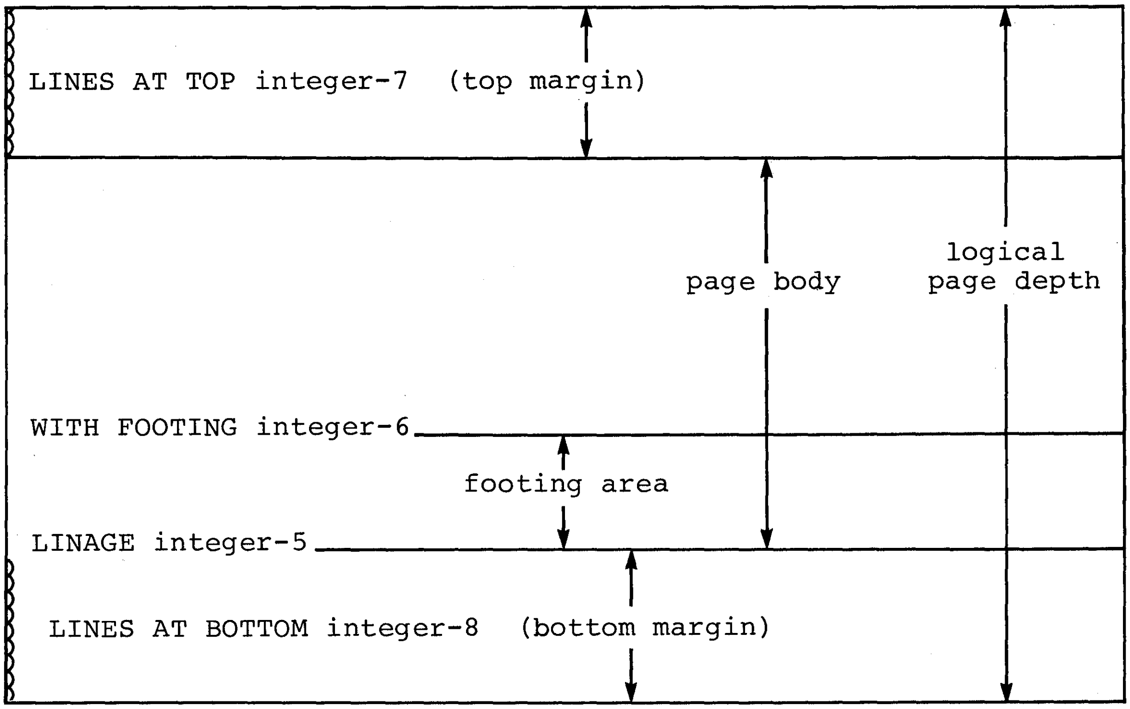


Figure 7. LINAGE Clause and Logical Page Depth

General Considerations

The logical page size specified in the LINAGE clause is the sum of all values specified in each option except the FOOTING option. If the LINES AT TOP and/or the LINES AT BOTTOM option is omitted, the assumed value is zero. Each logical page immediately follows the preceding logical page, with no additional spacing provided.

If the FOOTING option is omitted, its assumed value is equal to that of the page body (integer-5 or data-name-5).

At the time an OPEN OUTPUT statement is executed, the values of integer-5, integer-6, integer-7, and integer-8, if specified, are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file.

These values are then used for all logical pages printed for this file during a given execution of the program.

Data-name-5, data-name-6, data-name-7, and data-name-8, if specified, cause the following actions to take place:

- Their values at the time an OPEN OUTPUT statement is executed are used to determine the page body, the first footing line, the top margin, and the bottom margin for the first logical page only.

- Their values at the time a WRITE ADVANCING statement causes page ejection are used to determine the page body, first footing line, top margin, and bottom margin for the next succeeding logical page only.

LINAGE-COUNTER Special Register

For each FD entry containing a LINAGE clause, a separate LINAGE-COUNTER special register is generated. When more than one is generated, the user must qualify each LINAGE-COUNTER with its related file-name.

The implicit description of LINAGE-COUNTER is one of the following:

- If the LINAGE clause specifies data-name-1, LINAGE-COUNTER has the same PICTURE and USAGE as data-name-1.

- If the LINAGE clause specifies integer-1, LINAGE-COUNTER is a binary item with the same number of digits as integer-1.

The value in LINAGE-COUNTER at any given time is the line number at which the device is positioned within the current page. LINAGE-COUNTER may be referred to in Procedure Division statements; it must not be modified by them.

LINAGE-COUNTER is initialized to one when an OPEN statement for this file is executed.

LINAGE-COUNTER is automatically modified by any WRITE statement for this file. (See WRITE Statement in the Input/Output Statements chapter for details.)

REPORT CLAUSE

IBM Extension

The REPORT clause specifies the names of any reports associated with this file. (The REPORT clause is described in the chapter on the Report Writer feature.)

The LINAGE clause is not allowed with the REPORT clause.

End of IBM Extension

CODE-SET CLAUSE

The CODE-SET clause specifies the character code used to represent data on a magnetic tape file. When the CODE-SET clause is specified, alphabet-name identifies the character code convention used to represent data on the input/output device.

Alphabet-name must be defined in the SPECIAL-NAMES paragraph as STANDARD-1 (for ASCII-encoded files), or as NATIVE (for EBCDIC-encoded files). When NATIVE is specified, the compiler treats the CODE-SET clause as documentation.

The CODE-SET clause also specifies the algorithm for converting the character codes on the input/output medium from/to the internal EBCDIC character set.

When the CODE-SET clause is specified, all data in this file must have USAGE DISPLAY, and, if signed numeric data is present, it must be described with the SIGN IS SEPARATE clause.

When the CODE-SET clause is omitted, the EBCDIC character set is assumed for this file. The CODE-SET clause is valid only for magnetic tape files.

Note: For ASCII considerations, see Appendix B.

DATA DESCRIPTION

All of the data used in a COBOL program is described using a uniform system of representation. The basic concepts of data description are discussed in this chapter, as well as the actual COBOL clauses used to describe data.

DATA DESCRIPTION CONCEPTS

Most of the data processed by a COBOL program is presented in hierarchically arranged records. This is because most data must be divided into separate subdivisions for processing.

For example, in a department store's customer file, one complete record could contain all data pertaining to one customer. Subdivisions within that record could be: customer name, customer address, account number, department number of sale, unit amount of sale, dollar amount of sale, previous balance, plus other pertinent information.

To subdivide these records, COBOL uses a hierarchic concept of levels.

LEVEL CONCEPTS

Because records must be divided into logical subdivisions, the concept of levels is inherent in the structure of a record. Once a record has been subdivided, it can be further subdivided to provide more detailed data references.

The basic subdivisions of a record (that is, those fields not further subdivided) are called elementary items. Consequently, a record can be made up of a series of elementary items, or it may itself be an elementary item.

It may be necessary to refer to a set of elementary items; therefore elementary items can be combined into group items. Groups themselves can be combined into a more inclusive group that contains two or more subgroups. Because of this, within one hierarchy of data items, an elementary item can belong to more than one group item.

LEVEL-NUMBERS

A system of level-numbers specifies the organization of elementary and group items into records. Special level-numbers are also used; they identify data items used for special purposes.

Record Description Level-Numbers

Each group and elementary item in a record requires a separate entry, and each must be assigned a level-number. The following level-numbers are used to structure records:

01
: This level-number specifies the record itself, and is the most inclusive level-number possible. A level-01 entry may be either a group item or an elementary item.

02-49
: These level-numbers specify group and elementary items within a record. Less inclusive data items are assigned higher (not necessarily consecutive) level-numbers.

A group item includes all group and elementary items following it until a level-number less than or equal to the level-number of this group is encountered.

All elementary or group items immediately subordinate to one group item must be assigned identical level-numbers higher than the level-number of this group item.

Figure 8 illustrates the concept. Note that all groups immediately subordinate to the level-01 entry have the same level-number. Note also that elementary items from different subgroups do not necessarily have the same level number, and that elementary items can be specified at any level within the hierarchy.

```
The COBOL record-description entry                Is subdivided as
written as follows                                indicated below:

01  RECORD-ENTRY.                            <----this entry includes---┐
                                                                        |
    05  GROUP-1.                          <----this entry includes---┐  |
                                                                     |  |
        10  SUBGROUP-1.             <----this entry includes---┐     |  |
                                                               |     |  |
            15  ELEM-1 PIC ... .                               |     |  |
            15  ELEM-2 PIC ... .                               V     |  |
                                                                     |  |
        10  SUBGROUP-2.             <----this entry includes---┐     |  |
                                                               |     |  |
            15 ELEM-3 PIC ... .                                |     |  |
            15 ELEM-4 PIC ... .                                V     V  |
                                                                        |
    05  GROUP-2                          <----this entry includes----┐  |
                                                                     |  |
        15  SUBGROUP-3.             <----this entry includes---┐     |  |
                                                               |     |  |
            25 ELEM-5 PIC ... .                                |     |  |
            25 ELEM-6 PIC ... .                                |     |  |
                                                               V     |  |
        15  SUBGROUP-4 PIC ... .      this entry includes itself     V  |
                                                                        |
     05 GROUP-3 PIC ... .                  this entry includes itself   V

Its storage arrangement is illustrated below:

|<----------------------------RECORD-ENTRY----------------------------------->|
|                                                                             |
|<-----------GROUP-1-------------->|<-------------GROUP-2------->|             |
|                                  |                             |      |      |
|<--SUBGROUP-1-->|<--SUBGROUP-2--->|<--SUBGROUP-3--->|           |             |
┌───────┬────────┬────────┬────────┬────────┬────────┬───────────┬─────────────┐
│ELEM-1 │ ELEM-2 │ ELEM-3 │ ELEM-4 │ ELEM-5 │ ELEM-6 │ SUBGROUP-4│   GROUP-3   │
└───────┴────────┴────────┴────────┴────────┴────────┴───────────┴─────────────┘
```

Figure 8. Level-Number Concepts

Special Level-Numbers

Special level-numbers identify items that do not structure a record. The special level-numbers are:

66
: Identifies elementary or group items described by a RENAMES clause; such items regroup previously-defined data items.

(For an example, see the RENAMES clause description.)

77
: Identifies independent Working-Storage or Linkage Section items which are not subdivisions of other items, and which are not themselves subdivided.

88
: Identifies any condition-name entry that is associated with a particular value of a conditional variable.

(For an example, see the VALUE clause description.)

Note: Level-77 and level-01 entries in the Working Storage Section and Linkage Section must be given unique data-names, since neither can be qualified. Subordinate data-names, if they can be qualified, need not be unique.

Indentation

Successive data description entries may begin in the same column as preceding entries, or may be indented according to level-number. Indentation is useful for documentation, but does not affect the action of the compiler.

CLASSES OF DATA

All data used in a COBOL program can be divided into three classes and five categories. Every elementary item in a program belongs to both one of the classes and one of the categories. Every group item belongs to the alphanumeric class, even if the subordinate elementary items belong to another class and category. Figure 9 shows the relationship of data classes and categories.

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

Figure 9. Classes and Categories of Data

Standard Alignment Rules

The Standard Alignment rules for positioning data in an elementary item depend on the data category of the receiving item (that is, the item into which the data is placed).

NUMERIC ITEMS: For such receiving items, the following rules apply:

- The data is aligned on the assumed decimal point and, if necessary, truncated or padded with zeros. (An assumed decimal point is one that has logical meaning but that does not exist as an actual character in the data.)

- If an assumed decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated as in the preceding rule.

NUMERIC EDITED ITEMS: The data is aligned on the decimal point, and (if necessary) truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

ALPHANUMERIC, ALPHANUMERIC EDITED, ALPHABETIC: For these data categories, the following rules apply:

- The data is aligned at the leftmost character position, and (if necessary) truncated or padded with spaces at the right.

- If the JUSTIFIED clause is specified for this receiving item, the above rule is modified as described in the JUSTIFIED clause.

Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, the properties of the data are described in relation to a standard data format rather than a machine-oriented format.

The standard data format uses the decimal system to represent numbers, no matter what base is used by the system, and uses the remaining characters of the COBOL character set to describe nonnumeric data.

Character-String and Item Size

In COBOL, the size of an elementary item is determined through the number of character positions specified in its PICTURE character-string. In storage, however, the size is determined by the actual number of bytes the item occupies, as determined by the combination of its PICTURE character-string and its USAGE clause.

Usually, when an arithmetic item is moved from a longer field into a shorter one, this compiler truncates the data to the number of characters represented in the shorter item's PICTURE character-string.

For example, if a sending field with PICTURE S99999, and containing the value +12345, is moved to a COMPUTATIONAL (binary) receiving field with PICTURE S99, the data is truncated to +45.

IBM Extension

However, this IBM implementation, as a compile-time option, may be instructed in such an operation to truncate only those digits that will overflow the receiving field. When this option is used, the result obtained in the preceding example is +2345, because a 2-byte COMPUTATIONAL item can contain the equivalent of four decimal digits of data. Note that care must be used when using this option, because there are times when the data may contain a negative sign.

End of IBM Extension

SIGNED DATA

There are two categories of algebraic signs used in COBOL: operational signs and editing signs.

Operational Signs

Operational signs are associated with signed numeric items, and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause, and optionally upon its SIGN clause. Zero is considered a unique value, regardless of the operational sign. An unsigned field is always assumed to be positive or zero.

Editing Signs

Editing signs are associated with numeric edited items; editing signs are PICTURE symbols that identify the sign of the item in edited output.

DATA DESCRIPTION ENTRY

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

General Format 1

```
                 [ data-name  ]
level-number     <            > clause
                 | FILLER     |
                 L            J

    [REDEFINES clause]

    [BLANK WHEN ZERO clause]

    [JUSTIFIED clause]

    [OCCURS clause]

    [PICTURE clause]

    [SIGN clause]

    [SYNCHRONIZED clause]

    [USAGE clause]

    [VALUE clause].
```

General Format 2

```
  66  data-name-1  RENAMES clause.
```

General Format 3

```
  88  condition-name  VALUE clause.
```

The maximum length for a fixed-length Working-Storage or Linkage Section entry is 131071 bytes. The maximum length for all other entries is 32767 bytes.

GENERAL FORMAT 1

This format is used for record description entries in all Data Division sections, and for level-77 entries in the Working-Storage and Linkage Sections. The following rules apply:

- The level-number can be any number from 01 through 49, or 77. Level-numbers 01 through 09 can be written as 1 through 9.
- The clauses may be written in any order, with two exceptions:
  - The data-name/FILLER entry must immediately follow the level-number.
  - When specified, the REDEFINES clause must immediately follow the data-name clause.
- The PICTURE clause must be specified for every elementary item except index data items. (Index data items are discussed in the Table Handling chapter.)
- The BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses are only valid for elementary items.
- A space, or a comma or semicolon followed by a space, must separate clauses.
- Each entry must end with a period followed by a space.

GENERAL FORMAT 2

This format regroups previously-defined items. The following rules apply:

- A level-66 entry can neither rename another level-66 entry, nor can it rename a level-01, level-77, or level-88 entry.
- All level-66 entries associated with one record must immediately follow the last data description entry in that record.
- The entry must end with a period followed by a space.

Additional details are given in the description of the RENAMES Clause.

GENERAL FORMAT 3

This format describes condition-names.

A condition-name is a user-specified name that associates value(s) and/or range(s) of values with a conditional variable.

A conditional variable is a data item that can assume one or more values, which can in turn be associated with a condition-name.

The following rules for condition-name entries apply:

- Any entry beginning with level-number 88 is a condition-name entry.
- The condition-name entries associated with a particular conditional variable must immediately follow the conditional variable entry. The conditional variable can be any elementary data description entry except another condition-name or an index data item.

- A condition-name can be associated with a group item data description entry, in this case:
  - The condition-name value must be specified as a nonnumeric literal or figurative constant.
  - The size of the condition-name value must not exceed the sum of the sizes of all the elementary items within the group.
  - No element within the group may contain a JUSTIFIED or SYNCHRONIZED clause.
  - No USAGE other than USAGE IS DISPLAY may be specified within the group.
- Condition-names can be specified both at the group level and at subordinate levels within the group.
- The relation test implied by the definition of a condition-name at the group level is performed in accordance with the rules for comparison of nonnumeric operands regardless of the nature of elementary items within the group.
- A space, or a comma or semicolon followed by a space, must separate successive operands.
- Each entry must end with a period followed by a space.

Examples of both elementary and group condition-name entries are given in the description of the VALUE clause.

LEVEL-NUMBER

The level-number specifies the hierarchy of data within a record, and also identifies special-purpose data entries.

Format

level-number

The following rules for level-numbers apply:

- A level-number begins a data description entry, a renamed or redefined item, or a condition-name entry.
- Level-numbers 01 and 77 must begin in Area A followed by a space.
- Level-numbers 02 through 49, 66, and 88 may begin in either Area A or Area B and must be followed by a space.
- Single-digit level-numbers 1 through 9 may be substituted for level-numbers 01 through 09.

See also the description of level-numbers in the Data Description Concepts section, and the description of Data Division Entries in the chapter on Standard COBOL Format.

DATA-NAME/FILLER CLAUSE

A data-name explicitly identifies the data being described; the key word FILLER specifies an item that is not explicitly referred to in a program.

Format

```
┌           ┐
│ data-name │
<           >
│ FILLER    │
└           ┘
```

In a data description entry, a data-name or the key word FILLER must be the first word following the level-number.

┌──────────── IBM Extension ────────────┐

See the data-name clause description in the Report Writer chapter for the one exception to this rule.

└──────────── End of IBM Extension ────────────┘

A data-name identifies a data item used in the program. The data item may assume a number of different values during program execution.

The key word FILLER specifies an elementary item in a record that is never explicitly referred to.

┌──────────── IBM Extension ────────────┐

However, this IBM implementation allows a FILLER item to be a group item.

└──────────── End of IBM Extension ────────────┘

Under no circumstances may a FILLER item be explicitly referred to.

The key word FILLER may be used with a conditional variable, if explicit reference is never made to the conditional variable but only to values it may assume.

In a MOVE CORRESPONDING statement, or in an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, FILLER items are ignored.

REDEFINES CLAUSE

The REDEFINES clause describes the same storage area as capable of containing different data items.

Format

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

Note: level-number and data-name-1 are not part of the REDEFINES clause itself, and are only included in the format for clarity.

When specified, the REDEFINES clause must be the first entry following data-name-1.

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

Data-name-2 is the redefined item.

Data-name-1 identifies an alternate description for the same area, or the redefining item.

Implicit redefinition is assumed when more than one level-01 entry subordinate to an FD Entry or CD Entry is written. In such level-01 entries, the REDEFINES clause must not be specified.

Redefinition begins at data-name-1 and ends when a level-number less than or equal to that of data-name-2 is encountered. No entry having a level-number numerically lower than those of data-name-1 and data-name-2 may occur between these entries. For example:

```
05    A PICTURE X(6).
05    B REDEFINES A.
      10 B-1    PICTURE X(2).
      10 B-2    PICTURE 9(4).
05    C  PICTURE 99V99.
```

In this example, A is the redefined item, and B is the redefining item. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

The data description entry for data-name-2, the redefined item, cannot contain a REDEFINES clause or an OCCURS clause (with or without the DEPENDING ON option). However, the redefined item may itself be subordinate to an item that contains either clause. If the redefined item is subordinate to an OCCURS clause, data-name-2 in the REDEFINES clause (the redefined item) must not be subscripted or indexed.

Neither the redefined item nor the redefining item, or any items subordinate to them, can contain an OCCURS DEPENDING ON clause.

When data-name-1, the redefining item, is specified with a level-number other than 01, it must specify a storage area of the same size as the redefined item data-name-2.

Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must immediately follow the description of the redefined area without intervening entries that define new character positions. Multiple redefinitions must all use the data-name of the original entry that defined this storage area. For example:

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

The redefining entry (identified by data-name-1), and any subordinate entries, must not contain any VALUE clauses. This rule does not apply to condition-name entries.

General Considerations

Data items within an area can be redefined without their lengths being changed. For example:

```
05   NAME-2.
     10    SALARY PICTURE XXX.
     10    SO-SEC-NO PICTURE X(9).
     10    MONTH PICTURE XX.
05   NAME-1 REDEFINES NAME-2.
     10    WAGE PICTURE XXX.
     10    EMP-NO PICTURE X(9).
     10    YEAR PICTURE XX.
```

Data items can also be rearranged within an area. For example:

```
05   NAME-2.
     10    SALARY PICTURE XXX.
     10    SO-SEC-NO PICTURE X(9).
     10    MONTH PICTURE XX.
05   NAME-1 REDEFINES NAME-2.
```

```
    10     EMP-NO PICTURE X(6).
    10     WAGE PICTURE 999V999.
    10     YEAR PICTURE XX.
```

When an area is redefined, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and never supersedes a previous description. Therefore, if B REDEFINES C has been specified, either of the two procedural statements MOVE X TO B and MOVE Y TO C could be executed at any point in the program.

In the first case, the area described as B would assume the value of X. In the second case, the same physical area (described now as C) would assume the value of Y. Notice that, if the second statement is executed immediately after the first, the value of Y replaces the value of X in the one storage area.

The usage of a redefining data item need not be the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

```
05    B              PICTURE  99 USAGE DISPLAY VALUE 8.
05    C REDEFINES B PICTURE S99 USAGE COMPUTATIONAL.
05    A              PICTURE S99 USAGE COMPUTATIONAL.
```

The bit configuration of the DISPLAY value 8 is 1111 0000 1111 1000. Redefining B does not change the bit configuration of the data in the storage area. Therefore, the two statements, ADD B TO A and ADD C TO A give different results. In the first case, the value 8 is added to A (because B has USAGE DISPLAY). In the second statement, the value -48 is added to A (because C has USAGE COMPUTATIONAL) and the bit configuration (truncated to 2 decimal digits) in the storage area has the binary value -48.

Unexpected results may occur when a redefining item is moved to a redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is executed). Unexpected results may also occur when a redefined item is moved to a redefining item (as, from the previous example, if the statement MOVE C TO B is executed).

The REDEFINES clause may be specified for an item within the scope of an area being redefined (that is, an item subordinate to a redefined item). For example:

```
05  REGULAR-EMPLOYEE.
    10     LOCATION PICTURE A(8).
    10     GRADE PICTURE X(4).
    10     SEMI-MONTHLY-PAY PICTURE 9999V99.
    10     WEEKLY-PAY REDEFINES SEMI-MONTHLY-PAY
             PICTURE 999V999.

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

The REDEFINES clause may also be specified for an item subordinate to a redefining item. For example:

```
05  REGULAR-EMPLOYEE.
    10     LOCATION PICTURE A(8).
    10     GRADE PICTURE X(4).
    10     SEMI-MONTHLY-PAY PICTURE 999V999.

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

BLANK WHEN ZERO CLAUSE

The BLANK WHEN ZERO clause specifies that an item is to be filled entirely with spaces when its value is zero.

Format

```
BLANK WHEN ZERO
```

The BLANK WHEN ZERO clause may be specified only for elementary numeric or numeric edited items. When it is specified for a numeric item, the item is considered to be a numeric edited item.

If the BLANK WHEN ZERO clause is specified, the item contains nothing but spaces when its value is zero.

The BLANK WHEN ZERO clause must not be specified for level-66 or level-88 items.

The BLANK WHEN ZERO clause and the asterisk (*) as a suppression symbol must not be specified for the same entry.

JUSTIFIED CLAUSE

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

Format

```
┌ JUSTIFIED ┐
<           >  RIGHT
| JUST      |
└           ┘
```

The JUSTIFIED clause may be specified only at the elementary level. JUST is an abbreviation for JUSTIFIED, and has the same meaning.

The JUSTIFIED clause must not be specified for a numeric item, or for any item for which editing is specified.

When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item, and:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.

- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

When the JUSTIFIED clause is omitted, the rules for standard alignment are followed (see Standard Alignment Rules in the Data Description Concepts section).

The JUSTIFIED clause must not be specified with level-66 (RENAMES) and level-88 (condition-name) entries.

OCCURS CLAUSE

The OCCURS clause specifies tables whose elements can be referred to by indexing or subscripting. It is described in the chapter on Table Handling. — page 212

PICTURE CLAUSE

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

Format

```
┌ PICTURE ┐
<         >  IS character-string
| PIC     |
└         ┘
```

The PICTURE clause must be specified for every elementary item except an index data item. The PICTURE clause may be specified only at the elementary level. PIC is an abbreviation for PICTURE and has the same meaning.

The character-string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the data item.

The character-string may contain a maximum of 30 characters.

Symbols Used in the PICTURE Clause

The functions of each PICTURE clause symbol are defined in the following list; more detailed explanations are given in the following sections. Any punctuation character appearing within the PICTURE character-string is not considered a punctuation character, but rather as a PICTURE character-string symbol.

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

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

P The P indicates an assumed decimal scaling position, and is used to specify the location of an assumed decimal point when the point is not within the number which appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or in items that appear as arithmetic operands. In any operation converting data from one form of internal representation to another, if the item being converted is described with the PICTURE symbol P, each digit position described by a P is considered to contain the value zero, and the size of the item is considered to include these zero digit positions.

For example, PICTURE PPP99 DISPLAY defines a 2-character item whose value may range from .00001 through .00099, or zero; while 99PPP DISPLAY defines a 2-character item whose value may range from 1000 through 99000, or zero.

The scaling position character P may appear only to the left or right of the other characters in the string as a continuous string of Ps within a PICTURE description. The sign character S and the assumed decimal point V are the only characters which may appear to the left of a leftmost string of Ps. Because the scaling position character P implies an assumed decimal point (to the left of the Ps if the Ps are leftmost PICTURE characters and to the right of the Ps if the Ps are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

S The symbol S is used in a PICTURE character string to indicate the presence (but not the representation nor, necessarily, the position) of an operational sign, and must be written as the leftmost character in the PICTURE string. An operational sign indicates whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER option.

V The V is used in a character string to indicate 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, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.

X Each X in the character string represents a character position which may contain any allowable character from the EBCDIC set.

Z Each Z in the character string represents a leading numeric character position; when that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.

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

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

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

, Each comma in the character string represents a character position into which a comma will be inserted. This character is counted in the size of the item. The comma insertion character cannot be the last character in the PICTURE character string.

. When a period appears in the character string, it is an editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period will be inserted. This character is counted in the size of the item. The period insertion character cannot be the last character in the PICTURE character string.

Note: For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is specified 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.

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

* Each asterisk (check protect symbol) in the character string represents a leading numeric character position into which an asterisk will be placed when

that position contains a zero. Each asterisk (*) is counted in the size of the item.

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

Figure 10 gives the order in which PICTURE clause symbols must be specified.

FIRST SYMBOL / SECOND SYMBOL		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ -}	{+ -}	{CR DB}	$	{Z *}	{Z *}	{+ -}	{+ -}	$	$	9	A X	S	V	P	P
NON-FLOATING INSERTION SYMBOLS	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x
	.	x	x	x	x		x			x	x		x		x		x					
	{+ -}																					
	{+ -}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	$						x															

$ is the default value for the currency symbol.

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

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Non-floating insertion symbols + and -, floating insertion symbols Z, *, +, -, and $ and other symbol P appear twice in the above PICTURE character precedence table. 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 table represents its use to the right of the decimal point position.

Braces ({}) indicate items that are mutually exclusive.

Figure 10 (Part 1 of 3). PICTURE Clause Symbol Order

FIRST SYMBOL SECOND SYMBOL		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ -}	{+ -}	{CR DB}	$	{Z *}	{Z *}	{+ -}	{+ -}	$	$	9	A X	S	V	P	P
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
	$	x	x	x	x		x								x							
	$	x	x	x	x	x	x								x	x				x		x

$ is the default value for the currency symbol.

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

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Non-floating insertion symbols + and -, floating insertion symbols Z, *, +, -, and $ and other symbol P appear twice in the above PICTURE character precedence table. 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 table represents its use to the right of the decimal point position.

Braces ({}) indicate items that are mutually exclusive.

Figure 10 (Part 2 of 3). PICTURE Clause Symbol Order

FIRST SYMBOL / SECOND SYMBOL		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+ -}	{+ -}	{CR DB}	$	{Z *}	{Z *}	{+ -}	{+ -}	$	$	9	A X	S	V	P	P
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

$ is the default value for the currency symbol.

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

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Non-floating insertion symbols + and -, floating insertion symbols Z, *, +, -, and $ and other symbol P appear twice in the above PICTURE character precedence table. 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 table represents its use to the right of the decimal point position.

Braces ({}) indicate items that are mutually exclusive.

Figure 10 (Part 3 of 3). PICTURE Clause Symbol Order

Character-String Representation

The following symbols may appear more than once in one PICTURE character-string

A B P X Z 9 0 / , + - * $

Each time one of these symbols appears in the character-string, it represents an occurrence of that character or set of allowable characters in the data item.

An integer enclosed in parentheses immediately following any of these symbols specifies the number of consecutive occurrences of that symbol (a maximum of 32767).

For example, the following two PICTURE clause specifications are equivalent:

PICTURE IS $99999.99CR

PICTURE IS $9(5).99CR

The following symbols may each appear only once in one PICTURE character-string:

S V . CR DB

Data Categories and PICTURE Considerations

The allowable combinations of PICTURE symbols determine the data category of the item. Rules for each category follow.

ALPHABETIC ITEMS: The following rules apply:

- The PICTURE character-string can contain only the symbols A and B.
- The contents of the item in standard data format must consist of any of the 26 letters of the English alphabet and the space character.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

NUMERIC ITEMS: The following rules apply:

- The PICTURE character-string can contain only the symbols 9, P, S, and V.
- The number of digit positions must range from 1 through 18, inclusive.
- The contents of the item in standard data format must contain a combination of the 10 Arabic numerals, and, if signed, a representation of the operational sign.
- The USAGE of the item can be DISPLAY or COMPUTATIONAL.

IBM Extension

This IBM implementation also allows the USAGE to be COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

If a VALUE clause is specified for an elementary numeric item, the literal must be numeric. If a VALUE clause is specified for a group item consisting of elementary numeric items, the group is considered alphanumeric, and the literal must therefore be nonnumeric. The literal is treated exactly as specified; no editing is performed. Examples of numeric items are shown in Figure 11.

PICTURE	Valid Range of Values
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and +1000 through +999000 or zero

Figure 11. Examples of Numeric Items

Notes:

1. ASCII considerations are given in Appendix B.

2. System/360 and System/370 architecture do not support packed negative zero representation. Therefore, zero must be plus zero or zero; negative zero is not supported by this compiler.

ALPHANUMERIC ITEMS: The following rules apply for the alphanumeric category:

- The PICTURE character-string must consist of either:
  - Entirely the symbol X, or
  - Combinations of the symbols A, X, and 9. (A character-string containing all As or all 9s does not define an alphanumeric item.)

The item is treated as if the character-string contained only the symbol X.

- The contents of the item in standard data format may be any allowable characters from the EBCDIC character set.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

ALPHANUMERIC EDITED ITEMS: The following rules apply:

- The PICTURE character-string can contain the following symbols:

  A X 9 B 0 /

  The string must contain at least one of the following combinations:

  - At least one B and at least one X
  - At least one 0 and at least one X
  - At least one X and at least one /
  - At least one A and at least one 0
  - At least one A and at least one /
- The contents of the item in standard data format may be any allowable character from the EBCDIC character set.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.

NUMERIC EDITED ITEMS: The following rules apply:

- The PICTURE character-string can contain the following symbols:

  B P V Z 9 0 / , . + - CR DB * $

  The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 10), and the editing rules (see following section). The following additional rules also apply:

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

    B / Z 0 , . * + - CR DB

  - The number of digit positions represented in the character-string must be in the range of 1 through 18, inclusive.
  - The total number of character positions in the string (including editing characters) must not exceed 127.
- The contents of those character positions representing digits in standard data format must be one of the 10 Arabic numerals.
- USAGE DISPLAY must be specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.

PICTURE Clause Editing

There are two general methods of performing editing in a PICTURE clause: by insertion or by suppression and replacement.

There are four types of insertion editing: simple insertion, special insertion, fixed insertion, and floating insertion. There are two types of suppression and replacement editing: zero suppression and replacement with asterisks.

The type of editing allowed for an item depends on its data category. Figure 12 shows which type of editing is valid for each category.

Category	**Type of Editing**
Alphabetic	Simple insertion
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion
Numeric Edited	All

Each type of editing is discussed in detail in the following paragraphs.

Figure 12. Valid Editing for Each Data Category

SIMPLE INSERTION EDITING: This type of editing is valid for alphabetic, alphanumeric edited, and numeric edited items. The valid insertion symbols for each category are shown in Figure 13.

Category	**Valid Insertion Symbols**
Alphabetic	B
Alphanumeric Edited	B 0 /
Numeric Edited	B 0 / ,

Figure 13. Valid Simple Insertion Characters

Each insertion symbol is counted in the size of the item, and represents the position within the item where the equivalent characters will be inserted.

Examples of simple insertion editing are shown in Figure 14.

PICTURE	Value of Data	Edited Result
X(10)/XX	ALPHANUMER01	ALPHANUMER/01
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC
A(5)BA(5)	ALPHABETIC	ALPHA BETIC
99,B999,B000	1234	01, 234, 000
99,999	12345	12,345

Figure 14. Examples of Simple Insertion Editing

SPECIAL INSERTION EDITING: This type of editing is only valid for numeric edited items.

The period (.) is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point will be inserted.

The actual decimal point and the symbol V as the assumed decimal point must not both be specified in one PICTURE character-string.

Examples of special insertion editing are shown in Figure 15.

PICTURE	Value of Data	Edited Results
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

Figure 15. Examples of Special Insertion Editing

FIXED INSERTION EDITING: This type of editing is valid only for numeric edited items. The following insertion symbols are used:

$ (currency symbol)

+-CR DB (editing sign control symbols)

- In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in one PICTURE character-string.
- Unless it is preceded by a + or - symbol, the currency symbol ($) must be the leftmost character position in the character-string.
- When either + or - is used as a symbol, it must represent either the leftmost or rightmost character position in the character-string.
- When CR or DB is used as a symbol, it must represent the rightmost two character positions in the character-string.
- Editing sign control symbols produce results depending on the value of the data item, as shown in Figure 16.

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

Figure 16. Editing Sign Control Results

Examples of fixed insertion editing are shown in Figure 17.

PICTURE	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	-6555.555	-6555.55
9999.99-	+1234.56	1234.56
$999.99	-123.45	$123.45
-$999.99	-123.456	-$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	-123.45	$0123.45DB

Figure 17. Examples of Fixed Insertion Editing

FLOATING INSERTION EDITING: This type of editing is only valid for numeric edited items. The following symbols are used:

$ + -

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion characters.

Floating insertion editing is specified by using a string of at least two of the allowable floating insertion symbols to represent leftmost character positions in which these actual characters can be inserted.

The leftmost floating insertion symbol in the character-string represents the leftmost limit at which this actual character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this actual character can appear.

The second leftmost floating insertion symbol in the character-string represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data may replace all characters at or to the right of this limit.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character-string. If the period (.) special insertion symbol is included within the floating string, it is considered to be part of the character-string.

In a PICTURE character-string there are two ways to represent floating insertion editing, and in which editing is performed:

1. Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is performed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data, or of the decimal point, whichever is farther left. The character positions

to the left of the inserted character are filled with spaces.

2. All of the numeric character positions are represented by the floating insertion symbol. When editing is performed, then:

   - If the value of the data is zero, the entire data item will contain spaces.

   - If the value of the data is nonzero, the result is the same as in rule 1.

To avoid truncation, the minimum size of the PICTURE character-string must be:

- The number of character positions in the sending item, plus

- The number of nonfloating insertion symbols in the receiving item, plus

- One character for the floating insertion symbol. Even if the size of the PICTURE character string is larger than the minimum needed to accommodate the sending field, the PICTURE character string must be constructed so that the leftmost comma is preceded by at least one numeric character position to represent the digit and by one character to represent the floating insertion character. For example, if the sending field is defined as:

  ```
  01 SEND-FIELD  PIC  9(5)V99
  ```

  and the receiving field is defined as:

  ```
  77 RECV-FLD  PIC $,$$$,$$$,$$$.99
  ```

  The edited result will not be correct. To yield the correct results, the receiving field must be defined as:

  ```
  77 RECV-FLD  PIC  $$,$$$,$$$,$$$.99
  ```

Note: A single insertion symbol to the left of a simple or fixed insertion symbol followed by a string of floating insertion symbols is not considered part of the floating character-string. That is, the leftmost + in the following character-string:

```
+,+++,+++.++
```

is considered a fixed insertion symbol and not a floating insertion symbol.

Examples of floating insertion editing are shown in Figure 18.

PICTURE	Value of Data	Edited Result
$$$$.99	.123	$.12
$$$9.99	.12	$0.12
$$,$$$,999.99	-1234.56	$1,234.56
++,+++,999.99	-123456.789	-123,456.78
$$,$$$,$$$.99CR	-1234567	$1,234,567.00CR
++,+++,+++.+++	0000.00	

Figure 18. Examples of Floating Insertion Editing

ZERO SUPPRESSION AND REPLACEMENT EDITING: This type of editing is only valid for numeric edited items. The symbols Z and * are used.

The following symbols are mutually exclusive as floating replacement symbols in one PICTURE character-string:

Z * + - $

Zero suppression editing is specified by using a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be performed.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating editing symbols are considered part of the string. If the period (.) special insertion symbol is included within the floating editing string, it is considered to be part of the character-string.

In a PICTURE character-string there are two ways to represent zero suppression, and in which editing is performed:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is performed, any leading zero in the data that appears in the same character position as a suppression symbol is replaced by the replacement character. Suppression stops at the leftmost character that:
  - Does not correspond to a suppression symbol
  - Contains nonzero data
  - Is the decimal point
- All of the numeric character positions in the PICTURE character-string are represented by the suppression symbols. When editing is performed, and the value of the data is nonzero, the result is the same as in the preceding rule. If the value of the data is zero, then:
  - If Z has been specified, the entire data item will contain spaces.
  - If * has been specified, the entire data item, except the actual decimal point, will contain asterisks.

Note: The asterisk (*) as a suppression symbol and the BLANK WHEN ZERO clause must not be specified for the same entry.

Examples of zero suppression and replacement editing are shown in Figure 19.

PICTURE	Value of Data	Edited Result
****.**	0000.00	****.**
ZZZZ.ZZ	0000.00	
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	0000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.**+	-123.45	**123.45-
,*,***.**+	+12345678.9	12,345,678.90+
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B*,***,***.**BBDB	-12345.67	$ ***12,345.67 DB

Figure 19. Examples of Zero Suppression and Replacement Editing

SIGN CLAUSE

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

Format

```
               ┌ LEADING  ┐
[SIGN IS]     <            >  [SEPARATE CHARACTER]
               │ TRAILING │
               └          ┘
```

The SIGN clause may be specified only for a signed numeric data description entry (that is, one whose PICTURE character-string contains an S), or for a group item which contains at least one such elementary entry. USAGE IS DISPLAY must be specified, explicitly or implicitly.

Only one SIGN clause may apply to any one data description entry.

The SIGN clause is required only when an explicit description of the properties and/or position of the operational sign is necessary.

When specified, the SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER option is not specified, then:

- The operational sign is assumed to be associated with the LEADING or TRAILING digit position, whichever is specified, of the elementary numeric data item. (In this instance, specification of SIGN IS TRAILING is the equivalent of the standard action of the compiler.)
- The character S in the PICTURE character string is not counted in determining the size of the item (in terms of Standard Data Format characters).

If the SEPARATE CHARACTER option is specified, then:

- The operational sign is assumed to be the LEADING or TRAILING character position, whichever is specified, of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of Standard Data Format characters).
- + is the character used for the positive operational sign.
- - is the character used for the negative operational sign.
- At object time if one of the characters + or - is not present in the data an error occurs, and the program abnormally terminates.

Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also specified for such an entry, and conversion is necessary for computations or comparisons, the conversion takes place automatically.

If no SIGN clause is specified for a signed numeric data description entry, the position and mode of representation for the operational sign are determined as explained in the USAGE clause description.

Note: ASCII considerations for the SIGN clause are given in Appendix B.

SYNCHRONIZED CLAUSE

The SYNCHRONIZED clause specifies the alignment of an elementary item on a proper boundary in storage.

Format

```
┌ SYNCHRONIZED ┐   ┌ LEFT  ┐
<              >   │       │
│ SYNC         │   │ RIGHT │
└              ┘   └       ┘
```

SYNC is an abbreviation for SYNCHRONIZED and has the same meaning.

The SYNCHRONIZED clause ensures efficiency when performing arithmetic operations on an item. The SYNCHRONIZED clause is never required; however, performance is improved when SYNCHRONIZED is specified for binary items used in arithmetic.

The SYNCHRONIZED clause may appear only at the elementary level.

When specified, the LEFT and the RIGHT options are treated as documentation.

The length of an elementary item is not affected by the SYNCHRONIZED clause.

When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is synchronized.

When the item is aligned, the character positions between the last item assigned and the current item are known as slack bytes. These unused character positions are included in the size of any group to which the synchronized elementary item belongs.

The proper boundary used to align the item to be synchronized depends on the format of the item as defined by the USAGE clause.

When the SYNCHRONIZED clause is specified, the following actions are taken:

For a COMPUTATIONAL item:

- If its PICTURE is in the range of S9 through S9(4), the item is aligned on a halfword (even) boundary.

- If its PICTURE is in the range of S9(5) through S9(18), the item is aligned on a fullword (multiple of 4) boundary.

For a DISPLAY item, The SYNCHRONIZED clause is treated as documentation.

IBM Extension

The SYNCHRONIZED clause treats a COMPUTATIONAL-4 item as the equivalent of a COMPUTATIONAL item and a COMPUTATIONAL-3 item as the equivalent of a DISPLAY item.

End of IBM Extension

When the SYNCHRONIZED clause is specified for an item that also contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that redefines it. For example, if the programmer writes:

```
02 A PICTURE X(4).
02 B REDEFINES A PICTURE S9(9) COMP SYNC.
```

Data item A must begin on a fullword boundary.

When the SYNCHRONIZED clause is specified for a binary item that is the first elementary item subordinate to an item that contains a REDEFINES clause, the item must not require the addition of slack bytes.

When SYNCHRONIZED is not specified for binary items, no space is reserved for slack bytes.

In the File Section, the compiler assumes that all level-01 records containing SYNCHRONIZED items are aligned on a doubleword boundary in the buffer. The user must provide the necessary interrecord slack bytes to ensure alignment.

In the Working-Storage Section, the compiler aligns all level-01 entries on a doubleword boundary.

For the purposes of aligning binary items in the Linkage Section, all level-01 items are assumed to begin on doubleword boundaries. Therefore, if the user issues a CALL statement, such operands of any USING clause within it must be correspondingly aligned.

SLACK BYTES

There are two types of slack bytes: intrarecord slack bytes and interrecord slack bytes.

Intrarecord slack bytes are unused character positions preceding each synchronized item in the record.

Interrecord slack bytes are unused character positions added between blocked logical records.

Intrarecord Slack Bytes

For an output file, or in the Working-Storage Section, the compiler inserts intrarecord slack bytes to ensure that all SYNCHRONIZED items are on their proper boundaries. For an input file, or in the Linkage Section, performance is improved if binary items are properly aligned.

Because it is important for the user to know the length of the records in a file, the algorithm the compiler uses to determine whether slack bytes are required and, if they are required, the number of slack bytes to add, is as follows:

- The total number of bytes occupied by all elementary data items preceding the binary item are added together, including any slack bytes previously added.

- This sum is divided by m, where:

    m = 2 for binary items of 4-digit length or less

    m = 4 for binary items of 5-digit length or more

- If the remainder (r) of this division is equal to zero, no slack bytes are required. If the remainder is not equal to zero, the number of slack bytes that must be added is equal to m - r.

These slack bytes are added to each record immediately following the elementary data item preceding the binary item. They are defined as if they were an item with a level number equal to that of the elementary item that immediately precedes the SYNCHRONIZED binary item, and are included in the size of the group which contains them.

For example:

```
01  FIELD-A.
    05    FIELD-B  PICTURE X(5).
    05    FIELD-C.
    05    FIELD-D PICTURE XX.
*               10    Slack-Bytes  PICTURE X.  Inserted by
*                     compiler
          10    FIELD-E  COMPUTATIONAL PICTURE S9(6) SYNC.

01  FIELD-L.
    05    FIELD-M  PICTURE X(5).
    05    FIELD-N  PICTURE XX.
*   05    Slack-Bytes  PICTURE X.  Inserted by compiler
    05    FIELD-O.
          10    FIELD-P COMPUTATIONAL PICTURE S9(6) SYNC.
```

Slack bytes may also be added by the compiler when a group item is defined with an OCCURS clause and contains within it a SYNCHRONIZED binary data item. To determine whether slack bytes are to be added, the following action is taken:

- The compiler calculates the size of the group, including all the necessary intrarecord slack bytes.

- This sum is divided by the largest m required by any elementary item within the group.

- If r is equal to zero, no slack bytes are required. If r is not equal to zero, m - r slack bytes must be added.

The slack bytes are inserted at the end of each occurrence of the group item containing the OCCURS clause. For example, if a record is defined as follows:

```
01  WORK-RECORD.
    05    WORK-CODE  PICTURE X.
    05    COMP-TABLE OCCURS 10 TIMES.
          10    COMP-TYPE PICTURE X.
*         10    Ia-slack-Bytes  PIC XX.  Inserted by compiler
          10    COMP-PAY PICTURE S9(4)V99 COMP SYNC.
          10    COMP-HRS PICTURE S9(3) COMP SYNC.
          10    COMP-NAME  PICTURE X(5).
```

The record will appear in storage as shown in Figure 20.

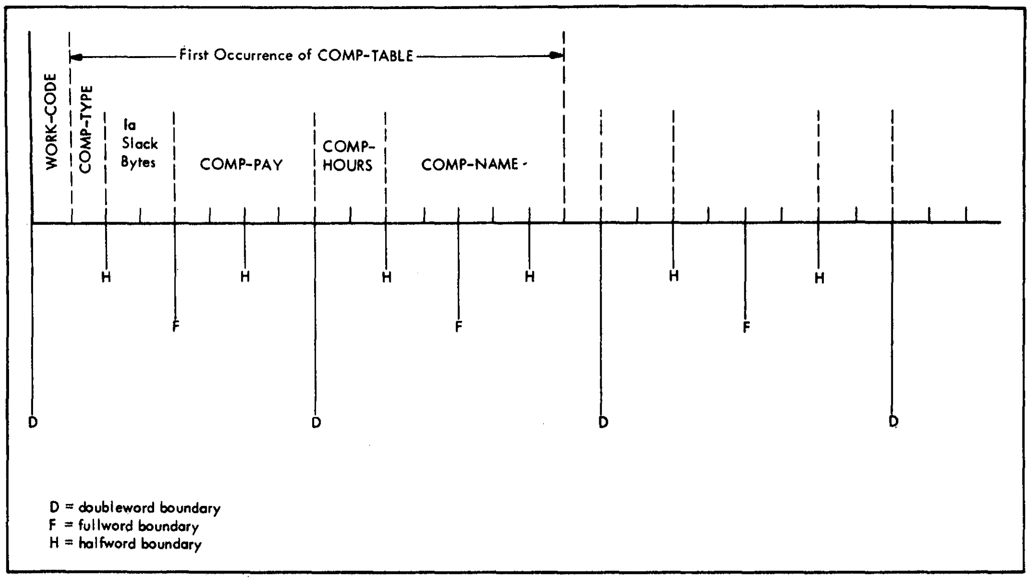


Figure 20. Insertion of Intraoccurrence Slack Bytes

In order to align COMP-PAY and COMP-HRS upon their proper boundaries, the compiler has added two intraoccurrence slack bytes (shown above as Ia-slack-Bytes).

However, without further adjustment, the second occurrence of COMP-TABLE would now begin one byte before a doubleword boundary, and the alignment of COMP-PAY and COMP-HRS would not be valid for any occurrence of the table after the first. Therefore, the compiler must add interoccurrence slack bytes at the end of the group, as though the record had been written:

```
01  WORK-RECORD.
    05    WORK-CODE.  PICTURE X.
    05    COMP-TABLE OCCURS 10 TIMES.
          10    COMP-TYPE PICTURE X.
*         10    Ia-Slack-Bytes  PIC XX.  Inserted by compiler
          10    COMP-PAY PICTURE S9(4)V99 COMP SYNC.
          10    COMP-HRS PICTURE S9(3)  COMP SYNC.
          10    COMP-NAME PICTURE X(5).
*         10    Ie-Slack-Bytes  PIC XX.  Inserted by compiler
```

so that the second (and each following) occurrence of COMP-TABLE begins one byte beyond a doubleword boundary. The storage layout for the first occurrences of COMP-TABLE will now appear as shown in Figure 21.

Each succeeding occurrence within the table will now begin at the same relative position as the first.

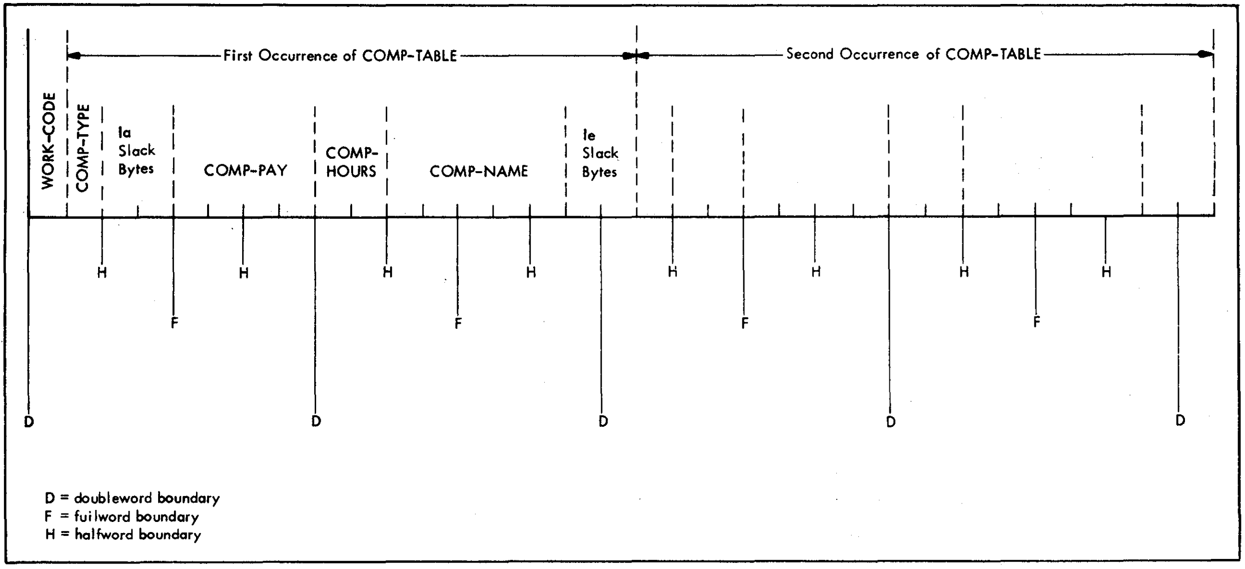


Figure 21. Insertion of Interoccurrence Slack Bytes

Where SYNCHRONIZED binary data items follow an entry containing an OCCURS DEPENDING ON clause, slack bytes are added on the basis of the field occurring the maximum number of times. If the length of this field is not divisible by the <u>m</u> required for the data, only certain values of the data-name that is the object of the DEPENDING ON option will give proper alignment of the fields. These values are those for which the length of the data field multiplied by the number of occurrences plus slack bytes that have been calculated based on the maximum number of occurrences is divisible by <u>m</u>.

For example:

```
01  FIELD-A.
    05    FIELD-B  PICTURE 99.
    05    FIELD-C  PICTURE X OCCURS 20 TO 99 TIMES
            DEPENDING ON FIELD-B.
*   05    Slack-Bytes  PICTURE X.  Inserted by compiler
    05    FIELD-D COMPUTATIONAL  PICTURE S99 SYNCHRONIZED.
```

In this example, when references to FIELD-D are required, FIELD-B is restricted to odd values only.

```
01  FIELD-A.
    05    FIELD-B  PICTURE 99.
    05    FIELD-C  PICTURE XX OCCURS 20 TO 99 TIMES
            DEPENDING ON FIELD-B.
*   05    Slack-Bytes  PICTURE X.  Inserted by compiler
    05    FIELD-D  PICTURE S99 COMP SYNC.
```

In this example, all values of FIELD-B give proper references to FIELD-D.

Interrecord Slack Bytes

If the file contains blocked logical records that are to be processed in a buffer, and any of the records contain binary entries for which the SYNCHRONIZED clause is specified, performance is better if the user adds any interrecord slack bytes needed for proper alignment.

The lengths of all the elementary data items in the record, including all intrarecord slack bytes, are added. For variable-length records, it is necessary to add four bytes for the count field. The total is then divided by the highest value of m for any one of the elementary items in the record.

If r (the remainder) is equal to zero, no interrecord slack bytes are required. If r is not equal to zero, $m - r$ slack bytes are required. These slack bytes may be specified by writing a level-02 FILLER at the end of the record.

Example: The following example shows the method of calculating both intrarecord and interrecord slack bytes. Consider the following record description:

```
01   COMP-RECORD.
     05    A-1     PICTURE X(5).
     05    A-2     PICTURE X(3).
     05    A-3     PICTURE X(3).
     05    B-1     PICTURE S9999  USAGE COMP SYNCHRONIZED.
     05    B-2     PICTURE S99999 USAGE COMP SYNCHRONIZED.
     05    B-3     PICTURE S9999  USAGE COMP SYNCHRONIZED.
```

This number of bytes in A-1, A-2, and A-3 totals 11. B-1 is a 4-digit COMPUTATIONAL item and therefore one intrarecord slack byte must be added before B-1. With this byte added, the number of bytes preceding B-2 total 14. Because B-2 is a COMPUTATIONAL item of five digits in length, two intrarecord slack bytes must be added before it. No slack bytes are needed before B-3.

The revised record description entry now appears as:

```
01   COMP-RECORD.
     05    A-1     PICTURE X(5).
     05    A-2     PICTURE X(3).
     05    A-3     PICTURE X(3).
*    05    Slack-Byte-1  PICTURE X.  Inserted by compiler
     05    B-1     PICTURE S9999 USAGE COMP SYNCHRONIZED.
*    05    Slack-Byte-2  PICTURE XX.  Inserted by compiler
     05    B-2     PICTURE S99999 USAGE COMP SYNCHRONIZED.
     05    B-3     PICTURE S9999  USAGE COMP SYNCHRONIZED.
```

There is a total of 22 bytes in COMP-RECORD, but, from the rules given in the preceding discussion, it appears that m = 4 and r = 2. Therefore, to attain proper alignment for blocked records, the user must add two interrecord slack bytes at the end of the record.

The final record description entry appears as:

```
01   COMP-RECORD.
     05    A-1     PICTURE X(5).
     05    A-2     PICTURE X(3).
     05    A-3     PICTURE X(3).
*    05    Slack-Byte-1  PICTURE X.  Inserted by compiler
     05    B-1     PICTURE S9999 USAGE COMP SYNCHRONIZED.
*    05    Slack-Byte-2  PICTURE XX.  Inserted by compiler
     05    B-2     PICTURE S99999 USAGE COMP SYNCHRONIZED.
     05    B-3     PICTURE S9999  USAGE COMP SYNCHRONIZED.
*    05    FILLER PICTURE XX.  Following are interrecord
*          slack bytes added by the user.
```

USAGE CLAUSE

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

Format

```
                ┌                        ┐
                │   DISPLAY              │
                │                        │
                │   INDEX                │
                │                        │
                │   COMPUTATIONAL        │
                │   COMP                 │
[USAGE IS]      <  ┌──────────────────┐  >
                │  │COMPUTATIONAL-3   │  │
                │  │COMP-3            │  │
                │  │                  │  │
                │  │COMPUTATIONAL-4   │  │
                │  │COMP-4            │  │
                │  └──────────────────┘  │
                └                        ┘
```

The USAGE clause can be specified for an entry at any level. However, if it is specified at the group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the usage of a group to which the elementary item belongs.

The USAGE clause specifies the format in which data is represented in storage. The format may be restricted if certain Procedure Division statements are used.

When the USAGE clause is not specified at either the group or elementary level, it is assumed that the usage is DISPLAY.

USAGE INDEX is discussed in the chapter on Table Handling.

COMPUTATIONAL-1 and COMPUTATIONAL-2 are discussed in Appendix A.

DISPLAY Option

The DISPLAY option can be explicit or implicit. It specifies that the data item is stored in character form, one character per eight-bit byte. This corresponds to the form in which information is represented for initial card input or for printed or punched output. USAGE IS DISPLAY is valid for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric edited
- Numeric edited
- External decimal (numeric)

Alphabetic, alphanumeric, alphanumeric edited, and numeric edited items are discussed in the description of the PICTURE clause.

EXTERNAL DECIMAL ITEMS: These items are sometimes referred to as zoned decimal items. Each digit of a number is represented by a single byte. The four high-order bits of each byte are zone bits; the four high-order bits of the low-order byte represent the sign of the item. The four low-order bits of each byte contain the value of the digit. When external decimal items are used for computations, the compiler performs the necessary conversions.

The maximum length of an external decimal item is 18 digits.

The PICTURE character-string of an external decimal item may contain only 9s, the operational sign symbol S, the assumed decimal point V, and one or more Ps.

Examples of external decimal items are shown in Figure 22.

Note: For ASCII considerations, see Appendix B.

Computational Options

The term "computational" refers to the following options of the USAGE clause:

COMPUTATIONAL or COMP (binary)

IBM Extension

COMPUTATIONAL-3 or COMP-3 (internal decimal). (COMPUTATIONAL-4 or COMP-4 (binary).

End of IBM Extension

A computational item represents a value to be used in arithmetic operations and must be numeric. If the USAGE of a group item is described with any of these options, it is the elementary items within the group that have this usage. The group itself is considered nonnumeric, and cannot be used in numeric operations.

The maximum length of a computational item is 18 decimal digits.

The PICTURE of a computational item may contain only:

9 (one or more numeric character positions)

S (one operational sign)

V (one implied decimal point)

P (one or more decimal scaling positions)

Examples of computational items are shown in Figure 22.

Numeric Type	PICTURE and USAGE and optional SIGN clause	Value	Internal Representation*
External Decimal	PIC S9999 DISPLAY	+1234	F1 F2 F3 C4
		-1234	F1 F2 F3 D4
		1234	F1 F2 F3 C4
	PIC 9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+1234	C1 F2 F3 F4
		-1234	D1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4
		-1234	60 F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234	F1 F2 F3 F4 4E
		-1234	F1 F2 F3 F4 60
Binary	PIC S9999 {COMP COMP-4}	+1234	04 D2
		-1234	FB 2E
	PIC 9999 {COMP COMP-4}	+1234	04 D2
		-1234	04 D2
Internal Decimal	PIC S9999 COMP-3	+1234	01 23 4C
		-1234	01 23 4D
	PIC 9999 COMP-3	+1234	01 23 4F
		-1234	01 23 4F

*The internal representation of each byte is shown as two hexadecimal digits. The bit configuration for each digit is:

Hex. Digit	Bit Configuration	Hex. Digit	Bit Configuration
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

(Hexadecimal 4E and 60 represent the EBCDIC + and - signs, respectively.)

Note: The leftmost bit of a binary number represents the sign: 0 is positive, 1 is negative. Negative numbers are shown in twos complement form.

Figure 22. Internal Representation of Numeric Items

COMPUTATIONAL Option: This option is specified for binary data items. Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PICTURE Clause	Storage Occupied
1 through 4	2 bytes (halfword)
5 through 9	4 bytes (fullword)
10 through 18	8 bytes (2 fullwords—not necessarily a doubleword)

The leftmost bit of the storage area is the operational sign.

Note: The COMPUTATIONAL option is system dependent and usually is assigned to representations that yield the greatest efficiency when performing arithmetic operations on that system; for this compiler, the COMPUTATIONAL option is binary.

IBM Extension

COMPUTATIONAL-3 Option: This option is specified for internal decimal items. Such an item appears in storage in packed decimal format. There are two digits per byte, with the sign contained in the rightmost four bits of the rightmost byte. Such an item may contain any of the digits 0 through 9, plus a sign, representing a value not exceeding 18 decimal digits.

For internal decimal items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive, but that does not represent an overpunch.

End of IBM Extension

IBM Extension

COMPUTATIONAL-4 Option: This option is specified for system-independent binary items. For this compiler, it is the equivalent of COMPUTATIONAL.

End of IBM Extension

VALUE CLAUSE

The VALUE clause specifies the initial contents of a data item, or the value(s) associated with a condition-name.

Format 1

```
[VALUE IS literal]
```

Format 2

```
                  ┌ VALUE IS   ┐
[88 condition-name <             >
                  └ VALUES ARE ┘

       literal-1 [ ┌ THROUGH ┐
                   <           > literal-2]
                   └ THRU    ┘

             literal-3 [ ┌ THROUGH ┐
                         <           > literal-4 ] ... ].
                         └ THRU    ┘
```

Note: Level-number 88 and condition-name are not part of the Format 2 VALUE clause itself, and are included in the format only for clarity.

The use of the VALUE clause varies with the Data Division section in which it is specified.

File and Linkage Sections: The VALUE clause must be used only in condition-name entries.

Working-Storage and Communication Sections: The VALUE clause is used in condition-name entries. It is also used to specify the initial value of any data item; the item assumes the specified value at the beginning of program execution. If the initial value is not explicitly specified, it is unpredictable.

IBM Extension

REPORT SECTION: The VALUE clause is used to specify the constant value of an elementary report group description entry. Details are given in the chapter on Report Writer.

End of IBM Extension

General Considerations

The VALUE clause must not be specified for any item whose length is variable.

For group entries, the VALUE clause must not be specified if the entry also contains any of the following clauses: JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

The VALUE clause must not conflict with other clauses in the data description entry, or in the data description of this entry's hierarchy. The following rules apply:

- Wherever a literal is specified, a figurative constant may be substituted.

- If the item is numeric, all VALUE clause literals must be numeric literals. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves with one additional restriction: The literal must not have a value that requires truncation of nonzero digits. If the literal is signed, the associated PICTURE character-string must contain a sign symbol (S).

- 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 for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99000 or zero. For PICTURE PPP99, the literal must be within the range .00000 through .00099.

- If the item is an elementary or group alphabetic, alphanumeric, alphanumeric edited, or numeric edited item, all VALUE clause literals must be nonnumeric literals. The literal is aligned according to the alignment rules, with one additional restriction: The number of characters in the literal must not exceed the size of the item. (See the section on Standard Alignment Rules.)

- The functions of the editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item. Therefore, any editing characters must be included in the literal. For example, if the item is defined as PICTURE +999.99 and the value is to be +12.34 then the VALUE clause should be specified as VALUE '+012.34'.

- A maximum of 65,535 bytes may be initialized by means of a single VALUE clause.

Format 1 Considerations

This format specifies the initial value of a data item in storage. Initialization is independent of any specified BLANK WHEN ZERO or JUSTIFIED clause.

A Format 1 VALUE clause must not be specified for an entry that contains or is subordinate to an entry that contains a REDEFINES or OCCURS clause.

If the VALUE clause is specified at the group level, the literal must be a nonnumeric literal or a figurative constant. The group area is initialized without consideration for the subordinate entries within this group. In addition, the VALUE clause must not be specified for subordinate entries within this group.

Format 2 Considerations

This format associates a value, values, and/or range(s) of values with a condition-name. Each such condition-name requires a separate level-88 entry.

The VALUE clause is required in a condition-name entry, and must be the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Therefore, every level-88 entry must always be preceded either by the entry for the conditional variable, or by another level-88 entry (when several condition-names apply to one conditional variable). Each such level-88 entry implicitly has the PICTURE characteristics of the conditional variable.

Every condition-name can be qualified by the name of its associated conditional variable, and by the conditional variable's qualifiers. If the associated conditional variable requires subscripts or indexes, each procedural reference to the condition-name must be subscripted or indexed as required for the conditional variable.

When only literal-1 is specified, the condition-name is associated with a single value.

When literal-1, literal-3, etc, are specified, the condition-name is associated with several single values.

When literal-1 THRU literal-2 is specified, the condition-name is associated with one range of values.

When literal-1 THRU literal-2 literal-3 THRU literal-4, and so forth, are specified, the condition-name is associated with more than one range of values. Literal-1 must be less than literal-2, literal-3 less than literal-4, and so forth.

Note that one or more single values and one or more ranges of values may be specified in a single Format 2 VALUE clause.

(The key words THRU and THROUGH are equivalent.)

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable. In the following example, CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables; the associated condition-names immediately follow the level-number 88. The PICTURE associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal. The PICTURE associated with CITY limits the condition-name value to a 3-character nonnumeric literal. Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed 5 characters, and the literal (because this is a group item) must be nonnumeric:

```
05   CITY-COUNTY-INFO.
          88  BRONX                   VALUE "03NYC".
          88  BROOKLYN                VALUE "24NYC".
          88  MANHATTAN               VALUE "31NYC".
          88  QUEENS                  VALUE "41NYC".
          88  STATEN-ISLAND           VALUE "43NYC".
     10   COUNTY-NO                PICTURE 99.
          88  DUTCHESS                VALUE 14.
          88  KINGS                   VALUE 24.
          88  NEW-YORK                VALUE 31.
          88  RICHMOND                VALUE 43.
     10   CITY                     PICTURE X(3).
          88  BUFFALO                 VALUE "BUF".
          88  NEW-YORK-CITY           VALUE "NYC".
          88  POUGHKEEPSIE            VALUE "POK".
05   POPULATION...
```

Additional condition-name entry rules are given in the description of General Format 3 at the beginning of the Data Description section.

Condition-names are used procedurally in Condition-name Conditions, which are described in the chapter on Conditional Expressions.

RENAMES CLAUSE

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

Format

```
66 data-name-1 RENAMES  data-name-2

        ┌ THROUGH ┐
    [   <          >  data-name-3].
        │ THRU    │
        └         ┘
```

Note: Level-number 66 and data-name-1 are not part of the RENAMES clause itself, and are included in the format only for clarity.

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with one logical record must immediately follow that record's last data description entry.

data-name-1: Identifies an alternative grouping of data items.

A level-66 entry cannot rename a level-01, level-77, level-88, or another level-66 entry.

Data-name-1 cannot be used as a qualifier; it can be qualified only by the names of level indicator entries or level-01 entries.

data-name-2 and data-name-3: Identify the original grouping of elementary data items; that is, they must name elementary or group items within the associated level-01 entry, and must not be the same data-name. Both data-names may be qualified.

The OCCURS clause must not be specified in the data entries for data-name-2 and data-name-3, or for any group entry to which they are subordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item occupying storage between data-name-2 and data-name-3.

data-name-2 Option

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

data-name-2 THRU data-name-3 Option

When data-name-3 is specified, data-name-1 is a group item that includes all elementary items:

- Starting with data-name-2 (if it is an elementary item) or the first elementary item within data-name-2 (if it is a group item), and

- Ending with data-name-3 (if it is an elementary item) or the last elementary item within data-name-3 (if it is a group item).

(The key words THRU and THROUGH are equivalent.)

The leftmost character in data-name-3 must not precede that in data-name-2; the rightmost character in data-name-3 must follow that in data-name-2. This means that data-name-3 cannot be subordinate to data-name-2.

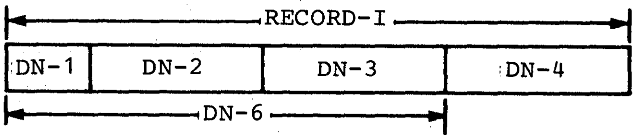
Figure 23 illustrates valid and invalid RENAMES clause specifications.

COBOL Specifications

Storage Layouts

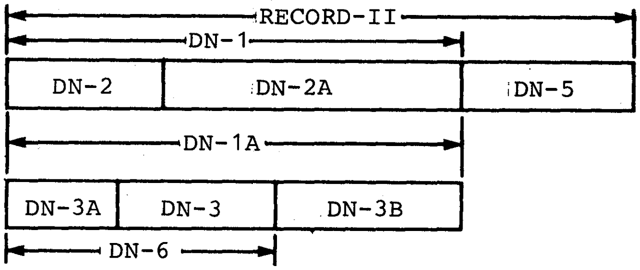
Example 1 (Valid)

```
    01   RECORD-I.
         05 DN-1... .
         05 DN-2... .
         05 DN-3... .
         05 DN-4... .
    66   DN-6 RENAMES DN-1 THROUGH DN-3.
```



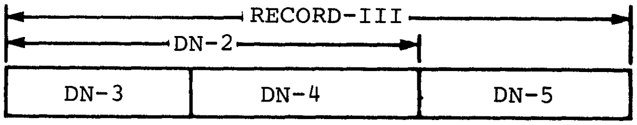
Example 2 (Valid)

```
    01   RECORD-II.
         05 DN-1.
            10 DN-2... .
            10 DN-2A... .
         05 DN-1A REDEFINES DN-1.
            10 DN-3A... .
            10 DN-3... .
            10 DN-3B... .
         05 DN-5... .
    66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



Example 3 (Invalid)

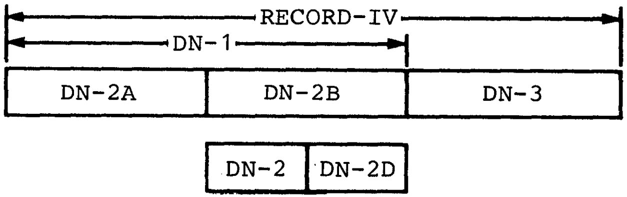
```
    01   RECORD-III.
         05 DN-2.
            10 DN-3... .
            10 DN-4... .
         05 DN-5... .
    66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



DN-6 is indeterminate

Example 4 (Invalid)

```
    01   RECORD-IV.
         05 DN-1.
            10 DN-2A... .
            10 DN-2B... .
            10 DN-2C REDEFINES DN-2B.
               15 DN-2... .
               15 DN-2D... .
         05 DN-3... .
    66   DN-4 RENAMES DN-1 THROUGH DN-2.
```



DN-4 is indeterminate

Figure 23. RENAMES Clause—Valid and Invalid Specifications

PART 4. PROCEDURE DIVISION

- PROCEDURE DIVISION ORGANIZATION
- ARITHMETIC EXPRESSIONS
- CONDITIONAL EXPRESSIONS
- CONDITIONAL STATEMENTS
- DECLARATIVES
- INPUT/OUTPUT STATEMENTS
- ARITHMETIC STATEMENTS
- DATA MANIPULATION STATEMENTS
- PROCEDURE BRANCHING STATEMENTS
- COMPILER-DIRECTING STATEMENTS

PROCEDURE DIVISION ORGANIZATION

The Procedure Division is required in every COBOL source program. The Procedure Division consists of optional Declaratives, and procedures that contain the sections and/or paragraphs, sentences, and statements which solve a data processing problem.

DECLARATIVES

When Declarative sections are specified, they must be grouped at the beginning of the Procedure Division. Declarative sections are preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES. The chapter on Declaratives gives additional information.

If Declarative sections are specified, the entire Procedure Division must be divided into sections.

IBM Extension

If Declarative sections are specified, the entire Procedure Division need not be divided into sections.

End of IBM Extension

PROCEDURES

A procedure is a paragraph or group of paragraphs, a section or a group of sections within the Procedure Division. A procedure-name is a user-defined name that identifies a section or a paragraph.

A section consists of a section header followed by no, one, or more than one successive paragraph. A section-header is a section-name followed by the key word SECTION, followed optionally by a priority-number, followed by a period and a space. (Priority-numbers are explained in the chapter on the Segmentation Feature.) A section-name is a user-defined word that identifies a section. A section-name, since it cannot be qualified, must be unique. A section ends immediately before the next section header, or at the end of the Procedure Division, or, in the Declaratives portion, at the key words END DECLARATIVES.

A paragraph consists of a paragraph-name followed by a period followed by a space, followed by no, one, or more than one successive sentence. A paragraph-name is a user-defined word that identifies a paragraph. A paragraph-name, since it can be qualified, need not be unique. A paragraph ends immediately before the next paragraph-name or section header, or at the end of the Procedure Division, or, in the Declaratives portion, at the key words END DECLARATIVES. If one paragraph in a program is contained within a section, then all paragraphs must be contained in sections.

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

A statement is a syntactically valid combination of words (identifiers, figurative constants, and so forth) and symbols (literals, relational-operators, and so forth) beginning with a COBOL verb.

An identifier consists of the word or words necessary to make unique reference to a data item, through qualification, subscripting, or indexing. In any Procedure Division reference (except the class test), the contents of an identifier must be compatible with the class specified through its PICTURE clause, or results are unpredictable.

Note: A level-88 (condition-name) entry, because it is not a data item, cannot be an identifier; the associated conditional variable, however, can be an identifier.

Execution begins with the first statement in the Procedure Division, excluding Declaratives. Statements are executed in the order in which they are presented for compilation, unless the rules indicate some other order. The end of the Procedure Division and the physical end of the program is that physical position in a source program after which no further procedures appear.

PROCEDURE DIVISION STRUCTURE

The structure of the Procedure Division is shown in the following format.

Format 1

PROCEDURE DIVISION [USING identifier-1 [identifier-2] ...]

[DECLARATIVES.

{section-name SECTION [priority-number]. USE sentence.

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

{section-name SECTION [priority-number].
[paragraph-name. [sentence] ...] ... } ...

Format 2

PROCEDURE DIVISION [USING identifier-1 [identifier-2]...].

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

CATEGORIES OF STATEMENTS

Three categories of statements are used in COBOL: conditional statements, imperative statements, and compiler-directing 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. (See the chapter on Conditions.) Figure 24 lists COBOL conditional statements.

Decision

IF

Arithmetic

ADD...ON SIZE ERROR
COMPUTE...ON SIZE ERROR
DIVIDE...ON SIZE ERROR
MULTIPLY...ON SIZE ERROR
SUBTRACT...ON SIZE ERROR

Data Movement

STRING...ON OVERFLOW
UNSTRING...ON OVERFLOW

Subprogram Linkage

CALL...ON OVERFLOW

Ordering

RETURN...AT END

Input/Output

DELETE...INVALID KEY
READ...AT END
READ...INVALID KEY
RESERVE...NO DATA KEY
REWRITE...INVALID KEY
START...INVALID KEY
WRITE...AT END-OF-PAGE
WRITE...INVALID KEY

Procedure Branching

CALL...ON OVERFLOW

Table Handling

SEARCH

Figure 24. Conditional Statements and Their Categories

An <u>imperative statement</u> specifies that an unconditional action is to be taken by the object program. An imperative statement may also consist of a series of imperative statements. Figure 25 lists COBOL imperative statements.

Arithmetic

ADD[1]
COMPUTE[1]
DIVIDE[1]
INSPECT (TALLYING)
MULTIPLY[1]
SUBTRACT[1]

Data Movement

ACCEPT (DATE,DAY,TIME)
ACCEPT...MESSAGE COUNT[7]
INSPECT (REPLACING)
MOVE
STRING[3]
TRANSFORM
UNSTRING[3]

Ending

STOP RUN
EXIT PROGRAM[7]
GOBACK[7]

Subprogram Linkage[7]

CALL[3]
CANCEL
ENTRY

Input/Output

ACCEPT identifier
CLOSE
DELETE[2]
DISABLE[7]
DISPLAY
ENABLE[7]
OPEN
READ[5]
RECEIVE[4] [7]
REWRITE[2]
SEND[7]
START[2]
STOP literal
WRITE[6]

Ordering[6]

MERGE
RELEASE
SORT

Procedure Branching

ALTER
CALL[3] [7]
EXIT
GO
PERFORM

Report Writer[7]

GENERATE
INITIATE
TERMINATE

Table Handling[7]

SET

[1] Without the SIZE ERROR option
[2] Without the INVALID KEY option
[3] Without the ON OVERFLOW option
[4] Without the NO DATA option
[5] Without the AT END or INVALID KEY options
[6] Without the INVALID KEY or END-OF-PAGE options
[7] Discussed in separate chapters of the Special Features Section

Figure 25. Imperative Statements and Their Categories

A compiler-directing statement causes the compiler to take a specific action during compilation.

Figure 26 lists the COBOL compiler-directing statements.

Library[1]	Declarative[2]	Documentation
COPY	USE	ENTER
BASIS INSERT DELETE		

[1] Discussed in the Source Program Library chapter

[2] Discussed in the Procedure Division, Debugging Features, and Report Writer chapters

Figure 26. Compiler-Directing Statements and Their Categories

CATEGORIES OF SENTENCES

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

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

An imperative sentence is an imperative statement (which may consist of a series of imperative statements) followed by a period followed by a space.

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

ARITHMETIC EXPRESSIONS

Arithmetic expressions are used as operands of certain conditional and arithmetic statements.

An arithmetic expression may consist of any of the following:

1. An identifier described as a numeric elementary item

2. A numeric literal

3. Identifiers and literals, as defined in items 1 and 2, separated by arithmetic operators

4. Two arithmetic expressions, as defined in items 1, 2, and/or 3, separated by an arithmetic operator

5. An arithmetic expression, as defined in items 1, 2, 3, and/or 4, enclosed in parentheses

Any arithmetic expression may be preceded by a unary operator.

Identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

ARITHMETIC OPERATORS

Five binary arithmetic operators and two unary arithmetic operators may be used in arithmetic expressions as shown in Figure 27. They are represented by specific characters that must be both preceded and followed by a space.

Binary Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Operator	Meaning
+	Multiplication by + 1
-	Multiplication by - 1

Figure 27. Binary and Unary Operators

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated.

Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the least inclusive to the most inclusive set.

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

1. Unary operator

2. Exponentiation

3. Multiplication and Division

4. Addition and Subtraction

Parentheses either eliminate ambiguities in logic where consecutive operations appear at the same hierarchic level or modify the normal hierarchic sequence of execution when this is necessary. When the order of consecutive operations at the same hierarchic level is not completely specified by parentheses, the order is from left to right.

Figure 28 shows permissible arithmetic symbol pairs. An arithmetic symbol pair is the appearance of two such symbols in sequence.

An arithmetic expression may begin only with a left parenthesis, a unary operator, or a variable (that is, an identifier or a literal). It may end only with a right parenthesis or a variable. An arithmetic expression must contain at least one reference to an identifier or a literal.

There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression, with each left parenthesis placed to the left of its corresponding right parenthesis.

Second Symbol ------------ First \ Symbol \	Variable (identifier	* / ** + -	unary + unary -	(	)
Variable (identifier or literal)	-	p	-	-	p
* / ** + -	p	-	p	p	-
unary + or unary -	p	-	-	p	-
(	p	-	p	p	1
)	-	p	-	-	p

Figure 28. Valid Arithmetic Symbol Pairs

CONDITIONAL EXPRESSIONS

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions are specified in IF, PERFORM, and SEARCH statements. A conditional expression can be specified in both simple and complex conditions. Both simple and complex conditions can be enclosed within any number of paired parentheses; this does not change the category of the condition.

SIMPLE CONDITIONS

There are five simple conditions: class, condition-name, relation, sign, and switch-status condition. A simple condition has a truth value of true or false. When a simple condition is enclosed in paired parentheses, its truth value is not changed.

CLASS CONDITION

The class condition determines whether a data item is alphabetic or numeric.

Format

```
                      ┌               ┐
                      | NUMERIC       |
identifier IS [NOT]  <                 >
                      | ALPHABETIC    |
                      └               ┘
```

The identifier being tested must be described implicitly or explicitly as USAGE DISPLAY.

IBM Extension

However, this IBM implementation allows the identifier being tested to be described as USAGE COMPUTATIONAL-3.

End of IBM Extension

The identifier being tested is determined to be numeric only if the contents consist of any combination of the digits 0 through 9.

If its PICTURE does not contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

If its PICTURE does contain an operational sign, the identifier being tested is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

In the EBCDIC collating sequence, valid embedded operational signs are hexadecimal A, B, C, D, E, and F; for items described with the SIGN IS SEPARATE clause, valid operational signs are + (hexadecimal 4E) and - (hexadecimal 60).

The NUMERIC test cannot be used with an identifier described as alphabetic or as a group item that contains one or more signed elementary items.

IBM Extension

This implementation allows use of a group item whose subordinates are signed.

End of IBM Extension

The identifier 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 ALPHABETIC test cannot be used with an identifier described as numeric.

Figure 29 shows valid forms of the class test.

Type of Identifier	**Valid Forms of the Class Test**	
Alphabetic	ALPHABETIC	NOT ALPHABETIC
Alphanumeric, Alphanumeric Edited, or Numeric Edited	ALPHABETIC NUMERIC	NOT ALPHABETIC NOT NUMERIC
External-Decimal or Internal-Decimal	NUMERIC	NOT NUMERIC

Figure 29. Valid Forms of the Class Test

CONDITION-NAME CONDITION

A condition-name condition causes a conditional variable to be tested to determine if its value is equal to (any of) the value(s) associated with the condition-name.

Format

condition-name

A condition-name is used in conditions as an abbreviation for the relation condition. The rules for comparing a conditional variable with a condition name value are the same as those specified for relation conditions.

If the condition-name is associated with a range of values (or with several ranges of values), the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. 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.

The following example illustrates the usage of condition-names and conditional variables:

```
02  GRADE-ID  PIC  99.

  88   PRIMARY-OTHER  VALUE 1 THRU 3, 5, 6.
  88   PRIMARY-FOUR   VALUE 4.
  88   JUNIOR-HI      VALUE 7 THRU 9.
  88   SENIOR-HI      VALUE 10 THRU 12.
```

GRADE-ID is the conditional variable; PRIMARY-OTHER, PRIMARY-FOUR, JUNIOR-HI, and SENIOR-HI are condition-names. For individual records in the file, only one of the values specified in the condition-name entries can be present. To determine the grade level of a specific record, any of the following can be coded:

```
IF PRIMARY-OTHER...(which tests for values 1, 2, 3, 5, 6.)
IF PRIMARY-FOUR... (which tests for value 4.)
IF JUNIOR-HI...    (which tests for values 7, 8, 9.)
IF SENIOR-HI...    (which tests for values 10, 11, 12.)
```

Depending on the evaluation of the condition-name condition, alternative paths of execution are taken by the object program.

RELATION CONDITION

A relation condition causes a comparison between two operands, either of which may be an identifier, a literal, or an arithmetic expression.

Format

```
                        ┌               ┐
                        | GREATER THAN  |
                        | >             |
operand-1 IS [NOT]     <  LESS THAN     >  operand-2
                        | <             |
                        | EQUAL TO      |
                        | =             |
                        └               ┘
```

Operand-1 is the subject of the relation condition; operand-2 is the object of the relation condition.

Operand-1 and operand-2 may each be an identifier, a literal, or an arithmetic expression. The relation condition must contain at least one reference to an identifier.

Except when two numeric operands are compared, operand-1 and operand-2 must have the same USAGE.

The relational operator specifies the type of comparison to be made. Figure 30 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space.

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

Figure 30. Relational Operators and Their Meanings

Detailed rules for numeric and nonnumeric comparisons are given in the following paragraphs. If either of the operands is a group item, nonnumeric comparison rules apply. Figure 31 summarizes the permissible comparisons.

Second Operand First Operand	NONNUMERIC OPERANDS		NUMERIC OPERANDS		
	GR AL AN ANE NE	FC(1) & NNL	ZR & NL	ED	BI AE ID
NONNUMERIC OPERANDS					
Group (GR) Alphabetic (AL) Alphanumeric (AN) Alphanumeric Edited (ANE) Numeric Edited (NE)	NN	NN	NN	NN	
Figurative Constant (1) (FC) and Nonnumeric Literal (NNL)	NN			NN	
NUMERIC OPERANDS					
Figurative Constant ZERO (ZR) and Numeric Literal (NL)	NN			NU	NU
External Decimal (ED)	NN	NN	NU	NU	NU
Binary (BI) Arithmetic Expression (AE) Internal Decimal (ID)			NU	NU	NU

NN = comparison as described for nonnumeric operands
NU = comparison as described for numeric operands
blank = comparison is not allowed

(1) = FC includes all figurative constants except ZERO

Figure 31. Permissible Numeric and Nonnumeric Comparisons

Comparison of Numeric Operands

For numeric class operands, their algebraic values are compared. The length (number of digits) of the operands is not significant.

Zero is considered a unique value, regardless of sign.

Unsigned numeric operands are considered positive.

Regardless of their USAGE, comparison of numeric operands is permitted.

Comparison of Nonnumeric Operands

Comparison of nonnumeric operands, or of one numeric and one nonnumeric operand, is made with respect to the binary collating sequence of the character set in use.

For the EBCDIC character set, the EBCDIC collating sequence is used. For the ASCII character set, the ASCII collating sequence is used. (Appendix H gives both complete collating sequences.) When the PROGRAM COLLATING SEQUENCE clause is specified in the OBJECT-COMPUTER paragraph, the collating sequence associated with the alphabet-name clause in the SPECIAL-NAMES paragraph is used.

When a nonnumeric and a numeric operand are compared, the following rules apply:

- If the nonnumeric operand is a literal or an elementary data item, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size, and the contents of this alphanumeric data item were then compared with the nonnumeric operand.

- 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, and the contents of this group item were then compared with the nonnumeric operand.

(See the rules for the MOVE statement for alphanumeric and group move operations.)

Numeric and nonnumeric operands may be compared only when their USAGE, explicitly or implicitly, is the same. In such comparisons, the numeric operand must be described as an integer literal or data item; noninteger literals and data items must not be compared with nonnumeric operands.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two cases to consider: operands of equal size, and operands of unequal size.

OPERANDS OF EQUAL SIZE: Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered as equal.

If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating sequence. The operand containing the character higher in the sequence is considered the greater operand.

OPERANDS OF UNEQUAL SIZE: If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Note: Valid comparisons for index-names and index data items are given in the chapter on Table Handling.

SIGN CONDITION

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero.

Format

```
                      [ POSITIVE ]
operand IS [NOT]    < NEGATIVE >
                      | ZERO     |
                      L          ]
```

The operand being tested must be defined as a numeric identifier, or it must be defined as an arithmetic expression that contains at least one reference to an identifier.

The 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.

An unsigned operand is POSITIVE or ZERO.

When NOT is specified, one algebraic test is executed for the truth value of the sign condition; for example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

SWITCH-STATUS CONDITION

The switch-status condition determines the on or off status of an UPSI switch.

Format

condition-name

The condition-name must be defined in the SPECIAL-NAMES paragraph as associated with the ON or OFF value of an UPSI switch. (See the description of the SPECIAL-NAMES paragraph in the Environment Division chapter.)

The switch-status condition tests the value associated with the condition-name. (The value associated with the condition-name is considered to be alphanumeric.) The result of the test is true if the UPSI switch is set to the position corresponding to condition-name.

COMPLEX CONDITIONS

A complex condition is formed by combining simple conditions, combined conditions, and/or complex conditions with logical connectives, or negating these conditions with logical negation.

The logical connectives used, and their meanings, are shown in Figure 32.

Logical Connective	Meaning
AND	Logical conjunction; that is, the truth value is true when both conditions are true.
OR	Logical inclusive OR; that is, the truth value is true when either or both conditions are true.
NOT	Logical negation; that is, reversal of truth value (the truth value is true if the condition is false).

Figure 32. Logical Connectives and Their Meanings

Each logical operator must be preceded and followed by a space.

The truth value of a complex condition, whether parenthesized or not, is the truth value that results from the interaction of all the stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically combined or logically negated.

A complex condition can be a negated simple condition or a combined condition (which can be negated).

NEGATED SIMPLE CONDITIONS

A simple condition is negated through the use of the logical operator NOT.

Format

<u>NOT</u> simple-condition

The simple-condition to be negated can be a class condition, a condition-name condition, a relation condition, a sign condition, or a switch-status condition. The simple-condition may not be negated if the condition itself contains a NOT.

The negated simple-condition gives the opposite truth value as the simple condition. That is, if the truth value of the simple-condition is true, then the truth value of that same negated simple-condition is false.

Placing a negated simple-condition within parentheses does not change its truth value. That is, the following two statements are equivalent:

```
NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).
```

COMBINED CONDITIONS

Two or more conditions can be logically connected to form a combined condition.

Format

```
          ┌ AND ┐
condition <     >  condition  ...
          | OR  |
          └     ┘
```

The condition to be combined may be any of the following:

- A simple-condition

- A negated simple-condition

- A combined condition

- A negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses)

- Combinations of the preceding conditions, specified according to the rules given in Figure 33

Combined Condition Element	Permissible Position in Combined Condition			
	Leftmost	When not leftmost may be immediately preceded by:	When not rightmost may be immediately followed by:	Rightmost
simple-condition	yes	OR NOT AND (	OR AND)	yes
OR AND	no	simple-condition)	simple-condition NOT (	no
NOT	yes	OR AND (	simple-condition (	no
(	yes	OR NOT AND (	simple-condition NOT (	no
)	no	simple-condition)	OR AND)	yes

Figure 33. Combined Conditions—Permissible Element Sequences

Parentheses are never needed when either AND or OR (but not both) is used exclusively in one combined condition. However, parentheses may be needed to find a final truth value when a combination of AND, OR, and NOT is used. Figure 33 summarizes the way in which conditions and logical operators can be combined and parenthesized.

There must be a one-to-one correspondence between left and right parentheses, with each left parenthesis to the left of its corresponding right parenthesis.

Figure 34 illustrates the relationships between logical operators and conditions C1 and C2 (where C1 and C2 are any conditions as defined above).

Values	Values			NOT	NOT	NOT	NOT
For C1	For C2	C1 AND C2	C1 OR C2	(C1 AND C2)	C1 AND C2	(C1 OR C2)	C1 OR C2
True	True	True	True	False	False	False	True
False	True	False	True	True	True	False	True
True	False	False	True	True	False	False	False
False	False	False	False	True	False	True	True

Figure 34. Logical Operators and Evaluation Results of Combined Conditions

Evaluation Rules—Combined Conditions

If parentheses are used, logical evaluation of combined conditions proceeds in the following order:

1. Conditions within parentheses are evaluated first.
2. Within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition.

If parentheses are not used (or are not at the same level of inclusiveness), the combined condition is evaluated in the following order:

1. Arithmetic expressions
2. Simple-conditions in the following order:
   a. Relation
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign
3. Negated simple-conditions in the same order as 2 above
4. Combined conditions, in the following order:
   a. AND
   b. OR
5. Negated combined conditions in the same order as 4 above
6. Consecutive operands at the same evaluation-order level are evaluated from left to right

For example:

NOT A IS GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE

is evaluated as if parenthesized as follows:

(NOT (A IS GREATER THAN B)) OR (((A+B) IS EQUAL TO C) AND (D IS POSITIVE))

The order of evaluation is as follows:

1. (A+B) is evaluated, giving some intermediate result, <u>x</u>.
2. (<u>x</u> IS EQUAL TO C) is evaluated, giving some intermediate truth value, <u>t1</u>.
3. (D IS POSITIVE) is evaluated, giving some intermediate truth value, <u>t3</u>.
4. (NOT (A IS GREATER THAN B)) is evaluated, giving some intermediate truth value, <u>t2</u>.
5. <u>t1</u> AND <u>t3</u> is evaluated, giving some intermediate truth value, <u>t4</u>.
6. <u>t2</u> OR <u>t4</u> is evaluated, giving the final truth value, and the result of the combined condition.

ABBREVIATED COMBINED RELATION CONDITIONS

When relation-conditions are written consecutively, any relation-condition after the first can be abbreviated by:

- Omission of the subject, or
- Omission of the subject and relational operator.

Format

```
                     ┌                     ┌              ┐          ┐
                     | ┌     ┐             | GREATER THAN |          |
                     | | AND |             | >            |          |
relation-condition < <       > [NOT]       | LESS THAN    | object   > ...
                     | | OR  |             | <            |          |
                     | └     ┘             | EQUAL TO     |          ┘
                     └                     | =            |
                                           └              ┘
```

In any consecutive sequence of relation-conditions, both forms of abbreviation can be specified. The abbreviated condition is evaluated as if:

1. The last stated subject is the missing subject.
2. The last stated relational operator is the missing relational operator.
3. The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Figure 33.
4. The word NOT is considered part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =.
5. NOT in any other position is considered a logical operator (and consequently results in a negated relation-condition).

Note: The rules for abbreviated combined relation-conditions as previously implemented are given in Appendix A.

Figure 35 shows examples of abbreviated combined relation-conditions and their nonabbreviated equivalents.

Abbreviated Combined Relation-Condition	Nonabbreviated Equivalent
A = B AND NOT LESS THAN C OR D	((A = B) AND (A NOT LESS THAN C)) OR (A NOT LESS THAN D)
A NOT GREATER THAN B OR C	(A NOT GREATER THAN B) OR (A NOT GREATER THAN C)
NOT A = B OR C	(NOT (A = B)) OR (A = C)
NOT (A = B OR LESS THAN C)	NOT ((A = B) OR (A LESS THAN C))
NOT (A NOT = B AND C AND NOT D)	NOT ((((A NOT = B) AND (A NOT = C)) AND (NOT (A NOT = D))))

Figure 35. Abbreviated Combined Relation-Condition Equivalents

CONDITIONAL STATEMENTS

A conditional statement specifies that a truth value of a condition is to be determined, and that the subsequent action of the object program depends on this truth value. Figure 24 on page 106 gives a list of the conditional statements.

Only the IF statement is discussed in this chapter; the other conditional statements are discussed in conjunction with their associated imperative statements, or in the chapters on special features.

IF STATEMENT

The IF statement causes a condition to be evaluated, and provides for alternative actions in the object program, depending on that truth value.

Format

```
             ┌                ┐  ┌                     ┐
             | statement-1    |  | ELSE statement-2    |
IF condition <                >  <                     >
             | NEXT SENTENCE  |  | ELSE NEXT SENTENCE  |
             └                ┘  └                     ┘
```

The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the period that ends the conditional sentence.

The condition may be any simple or complex condition as described in the Conditional Expressions chapter.

Statement-1 or statement-2 can be any one of the following:

- An imperative statement
- A conditional statement
- An imperative statement followed by a conditional statement

If the condition tested is true, one of the following actions takes place:

1. Statement-1, if specified, is executed. If statement-1 contains a procedure branching statement, control is transferred according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.

2. NEXT SENTENCE, if specified, is executed; that is, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.

If the condition tested is false, one of the following actions takes place:

1. ELSE statement-2, if specified, is executed. If statement-2 contains a procedure-branching statement, control is transferred according to the rules for that statement. If statement-2 does not contain a procedure-branching statement, control is passed to the next executable sentence.

2. ELSE NEXT SENTENCE, if specified, is executed; that is, statement-1, if specified, is ignored, and control passes to the next executable sentence.

3. If ELSE NEXT SENTENCE is omitted, control passes to the next executable sentence.

Note: When ELSE NEXT SENTENCE is omitted, all statements following the condition and preceding the period for the sentence are considered to be part of statement-1.

Nested IF Statements

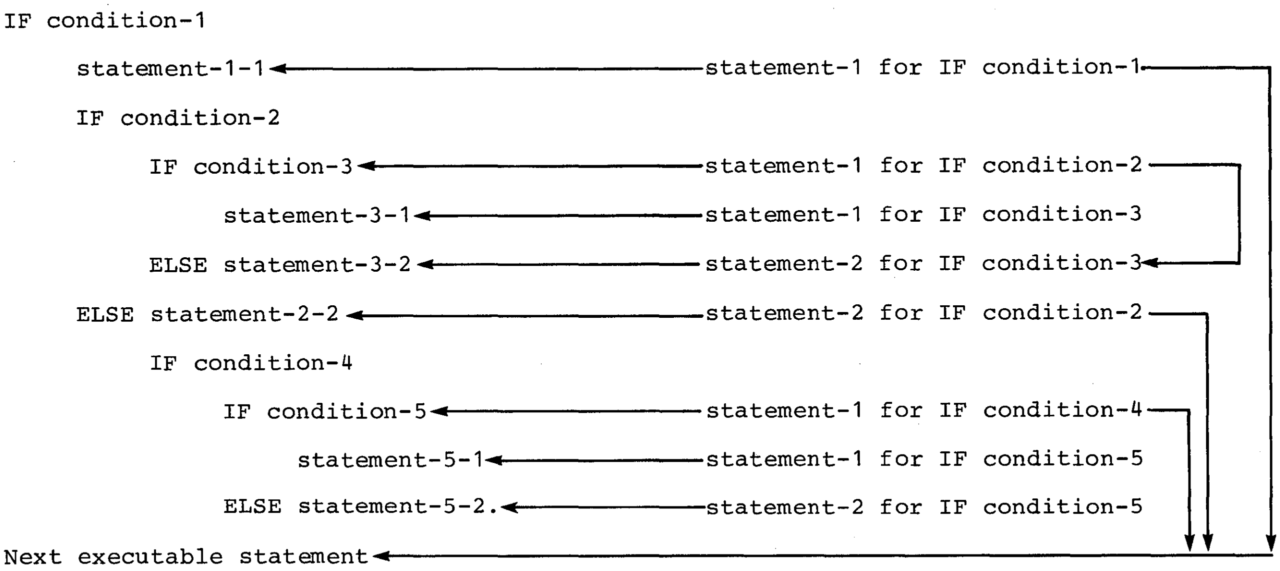
The presence of one or more IF statements within the initial IF statement constitutes a "nested IF statement."

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

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Therefore, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already paired with an ELSE.

Figure 36 shows the logical evaluation for the following nested IF statement:

```
IF condition-1 statement-1-1

  IF condition-2

    IF condition-3 statement-3-1

    ELSE statement-3-2

  ELSE statement-2-2

      IF condition-4

      IF condition-5  statement-5-1

        ELSE statement-5-2.
```



```
IF condition-1
     statement-1-1
     IF condition-2
          IF condition-3
               statement-3-1
          ELSE statement-3-2
     ELSE statement-2-2
          IF condition-4
               IF condition-5
                    statement-5-1
               ELSE statement-5-2.
Next executable statement
```

statement-1 for IF condition-1
statement-1 for IF condition-2
statement-1 for IF condition-3
statement-2 for IF condition-3
statement-2 for IF condition-2
statement-1 for IF condition-4
statement-1 for IF condition-5
statement-2 for IF condition-5

Note: If either condition-1 or condition-4 is false, the next sentence is executed, since neither condition has an ELSE.

Figure 36. Nested IF Statement Evaluation

Figure 37 shows possible true/false combinations for the same example.

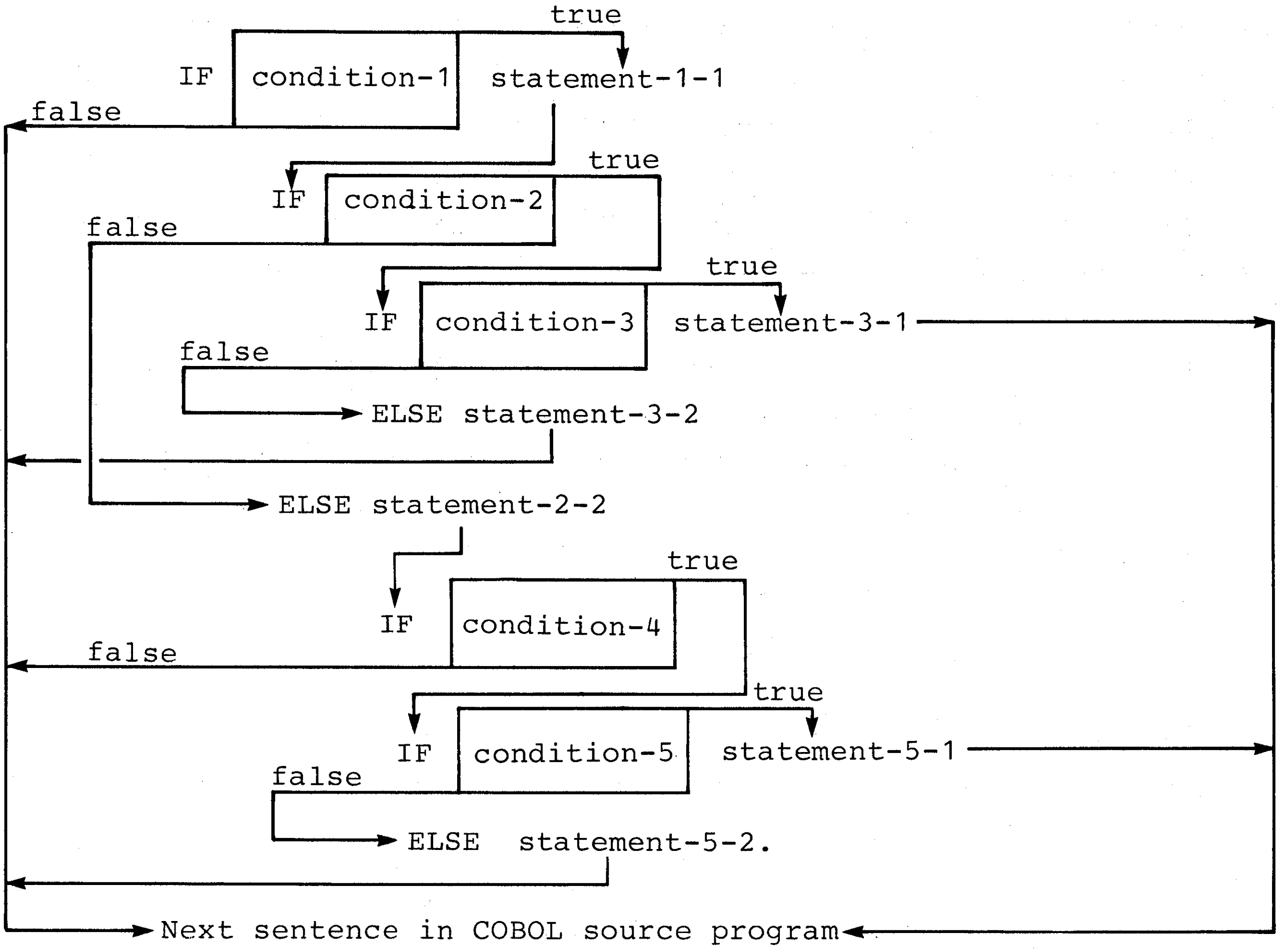


Figure 37. Nested IF Statement—True/False Combinations

Figure 38 is a flowchart for the same example.

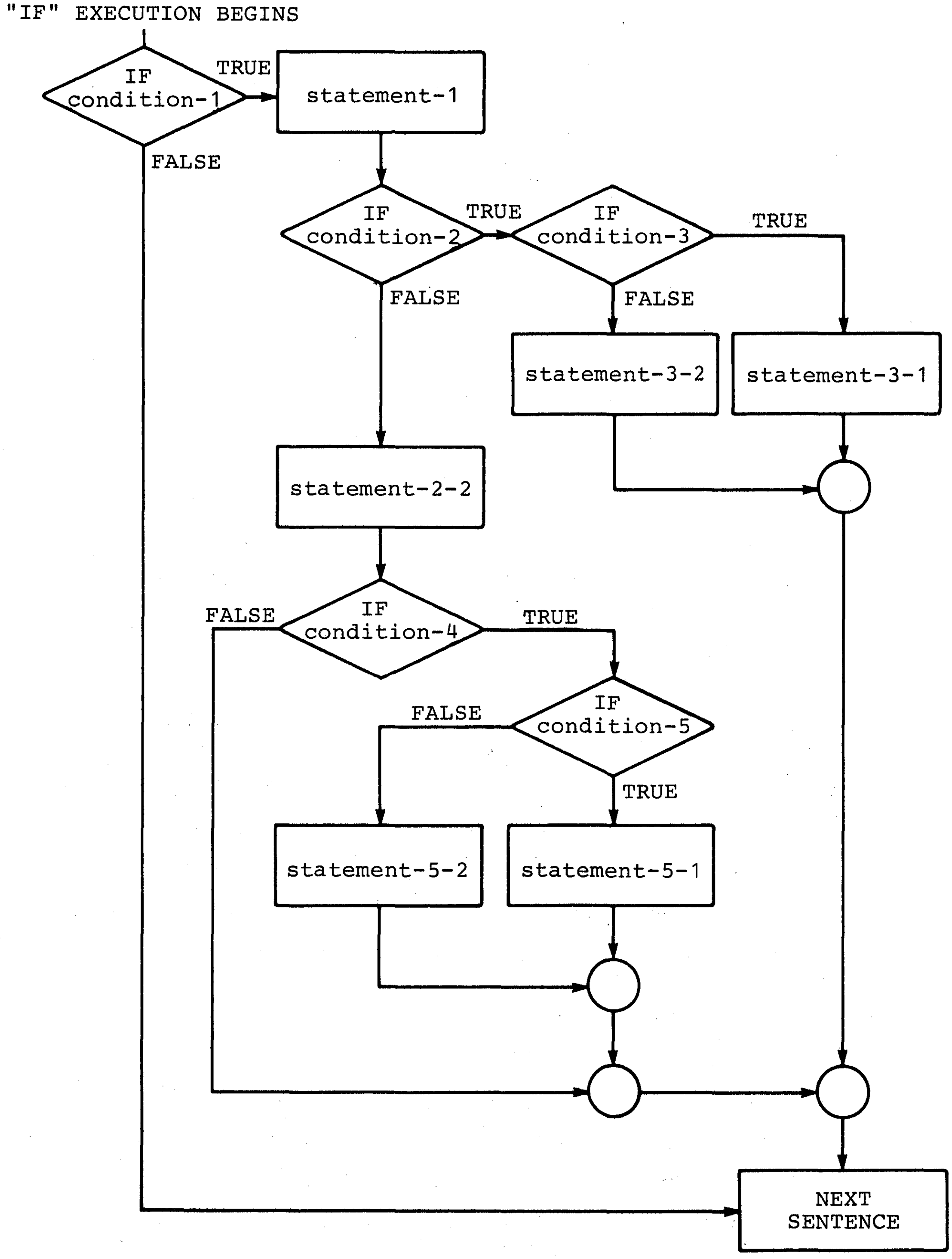


Figure 38. Nested IF Statement—Flowchart

The Declaratives section provides a method of invoking procedures that are executed when an exceptional condition occurs that cannot usually be tested by the COBOL programmer.

Format

```
PROCEDURE DIVISION  [USING identifier-1   [identifier-2] ...].

DECLARATIVES.
{section-name  SECTION  [priority-number].   USE sentence.

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

Declarative procedures are provided for the processing of exceptional input/output conditions and debugging procedures.

IBM Extension

Declarative procedures are also provided for Report Writer report groups.

End of IBM Extension

Declarative procedures are written at the beginning of the Procedure Division in a series of Declarative sections. Each such section starts with a USE sentence that identifies this section's function; the series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative section ends with the occurrence of another section-name followed by a USE sentence, or with the key words END DECLARATIVES.

The entire group of Declarative procedures is preceded by the key word DECLARATIVES, written on the next line after the Procedure Division header; the group is followed by the key words END DECLARATIVES. The key words DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a period. No other text may appear on the same line.

In the Declaratives portion of the Procedure Division, each section header (with an optional priority number) must be followed by a period and a space, and must be followed by a USE sentence followed by a period and a space. No other text may appear on the same line. There are three forms of the USE sentence:

- USE AFTER EXCEPTION/ERROR

IBM Extension

- USE BEFORE REPORTING (see chapter on Report Writer)

End of IBM Extension

- USE FOR DEBUGGING (see chapter on Debugging)

The USE sentence itself is never executed; instead, the USE sentence defines the conditions that will cause execution of the immediately following procedural paragraphs, which specify the actions to be taken. After the procedure is executed, control is returned to the routine that activated it.

Within a Declarative procedure, except for the USE statement itself, there must be no reference to any nondeclarative procedure.

Within a Declarative procedure, no statement may be executed that would cause execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

An exit from a Declarative procedure is effected by executing the last statement in the procedure.

In this chapter, only the USE AFTER EXCEPTION/ERROR procedure is described.

IBM Extension

The USE BEFORE REPORTING Declarative procedure is described in the chapter on the Report Writer feature.

End of IBM Extension

EXCEPTION/ERROR DECLARATIVE

The EXCEPTION/ERROR Declarative specifies procedures for input/output exception or error handling that are to be executed in addition to the standard system procedures.

Format

```
section-name SECTION [priority-number].

                         ┌ EXCEPTION ┐
    USE AFTER STANDARD  <             >  PROCEDURE
                         │ ERROR     │
                         └           ┘

              ┌                                ┐
              │ file-name-1 [file-name-2] ...  │
              │ INPUT                          │
        ON   <  OUTPUT                          >
              │ I-O                            │
              │ EXTEND                         │
              └                                ┘
```

Note: The section-name SECTION and priority-number are not part of the USE sentence itself and are only included in the format for clarity.

The words EXCEPTION and ERROR are synonymous and may be used interchangeably.

File-Name Option

This option is valid for physical sequential and VSAM files. When this option is specified, the procedure is executed only for the file(s) named. No file-name can refer to a sort or merge file. For any given file, only one EXCEPTION/ERROR procedure may be specified; thus, file-name specification must not cause simultaneous requests for execution of more than one EXCEPTION/ERROR procedure. For example, if an input file is specifically named in one EXCEPTION/ERROR procedure, there must not also be an EXCEPTION/ERROR procedure for all INPUT files—if both were specified, then, when an error occurred involving this input file, there would be simultaneous requests for the execution of both procedures.

INPUT Option

This option is valid for physical sequential and VSAM files. When this option is specified, the procedure is executed for all files opened in INPUT mode.

OUTPUT Option

This option is valid for physical sequential and VSAM files. When this option is specified, the procedure is executed for all files opened in OUTPUT mode.

I-O Option

This option is valid for direct access storage physical sequential and VSAM files. When this option is specified, the procedure is executed for all files opened in I-O mode.

EXTEND Option

This option is valid for physical sequential and VSAM sequential files.

IBM Extension

This option is also valid for sequentially accessed VSAM indexed files.

End of IBM Extension

When this option is specified, the procedure is executed for all files opened in EXTEND mode.

General Considerations

The EXCEPTION/ERROR procedure is executed:

- Either after completing the standard system input/output error routine, or
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END option has not been specified in the input/output statement, or
- Upon recognition of an IBM-defined condition that causes status key 1 to be set to 9. (See the description of status keys in the "Common Processing Facilities" section.)

After execution of the EXCEPTION/ERROR procedure, control is returned to the invoking routine.

The EXCEPTION/ERROR procedures are activated when an input/output error occurs during execution of a READ, WRITE, REWRITE, START, or DELETE statement.

If an OPEN statement is issued for a file already in the open status, the EXCEPTION/ERROR procedures are activated; when the execution of an OPEN statement is unsuccessful because of any other reason, the EXCEPTION/ERROR procedures are not activated.

If a file is in the OPEN status, and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedures are activated. If the file is in a closed status and a CLOSE statement is issued, the EXCEPTION/ERROR procedures are not activated.

Within a Declarative procedure, there must be no reference to any nondeclarative procedure. In the nondeclarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR Declarative procedure, except that PERFORM statements may refer to an EXCEPTION/ERROR procedure or to procedures associated with it.

Within an EXCEPTION/ERROR Declarative procedure, no statement may be executed that would cause execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

Programming Notes

EXCEPTION/ERROR procedures can be used to check the status key values whenever an input/output error occurs.

Care should be used in specifying EXCEPTION/ERROR procedures for any file. The following notes apply:

1. At initial OPEN time for any file, the current Declarative has not yet been established by the object program.

2. Therefore, if a file is closed that has never been opened, no Declarative can receive control.

3. However, if this file has been previously opened, the last previously established Declarative procedure receives control.

   For example: An OPEN OUTPUT statement establishes a Declarative procedure for this file, and the file is then closed without error. During later processing, if a logic error occurs, control will go to the Declarative procedure established when the file was opened for OUTPUT.

INPUT/OUTPUT STATEMENTS

COBOL input/output statements transfer data to and from files stored on external media, and also control low-volume data that is obtained from or sent to such input/output devices as the card reader or the console typewriter.

In COBOL, the unit of data made available to the program is a record, and the COBOL programmer need only be concerned with such records. Provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (where feasible), unblocking and blocking, and volume switching procedures.

The description of the file in the Environment Division and Data Division governs which input/output statements are allowed in the Procedure Division. Required and optional entries for each type of file organization are shown in Figure 39 (sequential files), Figure 40 (VSAM indexed files), and Figure 41 (VSAM relative files).

Discussions in this section use the terms volume and reel. The term volume refers to all input/output devices. The term reel applies only to tape devices. Treatment of direct access storage devices in the sequential access mode is logically equivalent to the treatment of tape devices.

There are several common processing facilities that apply to more than one input/output statement. These facilities are discussed before the descriptions of the separate input/output statements.

Device Type	ENVIRONMENT DIVISION		DATA DIVISION		PROCEDURE DIVISION	
	Required Entries	Optional Entries	Required Entries	Optional Entries	Required Entries	Optional Entries
Card Reader	SELECT ASSIGN	ACCESS SEQUENTIAL ORGANIZATION SEQUENTIAL RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS OMITTED	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	READ [INTO] [AT END] USE {EXCEPTION/ERROR FOR DEBUGGING}
Card Punch					OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] USE {EXCEPTION/ERROR FOR DEBUGGING}
Printer				BLOCK CONTAINS RECORD CONTAINS DATA RECORDS LINAGE REPORT	OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] [ADVANCING] [END-OF-PAGE] USE {EXCEPTION/ERROR BEFORE REPORTING FOR DEBUGGING}
Tape		ACCESS SEQUENTIAL ORGANIZATION SEQUENTIAL RESERVE FILE STATUS RERUN {RECORDS REEL/UNIT} SAME [RECORD SORT SORT-MERGE] AREA MULTIPLE FILE TAPE	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS CODE-SET LINAGE	OPEN INPUT CLOSE [REEL/UNIT [REMOVAL NO REWIND]] [NO REWIND] [LOCK]	READ [INTO] [AT END] USE {EXCEPTION/ERROR FOR DEBUGGING}
				BLOCK CONTAINS RECORD CONTAINS DATA RECORDS CODE-SET LINAGE REPORT	OPEN {OUTPUT EXTEND} CLOSE [REEL/UNIT [REMOVAL NO REWIND]] [NO REWIND] [LOCK]	WRITE [FROM] [ADVANCING] [END-OF-PAGE] USE {EXCEPTION/ERROR BEFORE REPORTING FOR DEBUGGING}
Direct Access Storage (Physical Sequential)		ACCESS SEQUENTIAL ORGANIZATION SEQUENTIAL RESERVE FILE STATUS RERUN {RECORDS REEL/UNIT} SAME [RECORD SORT SORT-MERGE] AREA	LABEL RECORDS STANDARD	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [REEL/UNIT [REMOVAL NO REWIND]] [NO REWIND] [LOCK]	READ [INTO] [AT END] USE {EXCEPTION/ERROR FOR DEBUGGING}
				BLOCK CONTAINS RECORD CONTAINS DATA RECORDS LINAGE	OPEN I-O CLOSE [REEL/UNIT [REMOVAL NO REWIND]] [NO REWIND] [LOCK]	READ [INTO] [AT END] REWRITE [FROM] WRITE [FROM] USE {EXCEPTION/ERROR FOR DEBUGGING}
				BLOCK CONTAINS RECORD CONTAINS DATA RECORDS LINAGE REPORT	OPEN {OUTPUT EXTEND} CLOSE [REEL/UNIT [REMOVAL NO REWIND]] [NO REWIND] [LOCK]	WRITE [FROM] [ADVANCING] [END-OF-PAGE] USE {EXCEPTION/ERROR BEFORE REPORTING FOR DEBUGGING}
Direct Access Storage (VSAM)		ACCESS SEQUENTIAL ORGANIZATION SEQUENTIAL RESERVE PASSWORD FILE STATUS RERUN...RECORDS SAME [RECORD SORT SORT-MERGE] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	READ [INTO] [AT END] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	READ [INTO] [AT END] REWRITE [FROM] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN {OUTPUT EXTEND} CLOSE [LOCK]	WRITE [FROM] USE {EXCEPTION/ERROR FOR DEBUGGING}

Figure 39. Sequential Files—Required and Optional Entries

Access	ENVIRONMENT DIVISION		DATA DIVISION		PROCEDURE DIVISION	
	Required Entries	Optional Entries	Required Entries	Optional Entries	Required Entries	Optional Entries
Sequen-tial	SELECT ASSIGN ORGANIZATION INDEXED RECORD KEY [PASSWORD]	ACCESS SEQUENTIAL ALTERNATE RECORD KEY [PASSWORD] [DUPLICATES] RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	READ [NEXT] [INTO] [AT END] START [KEY] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN OUTPUT OPEN EXTEND CLOSE [LOCK]	WRITE [FROM] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	READ [NEXT] [INTO] [AT END] START [KEY] [INVALID KEY] REWRITE [FROM] [INVALID KEY] DELETE USE {EXCEPTION/ERROR FOR DEBUGGING}
Random	SELECT ASSIGN ORGANIZATION INDEXED ACCESS RANDOM RECORD KEY [PASSWORD]	ALTERNATE RECORD KEY [PASSWORD] [DUPLICATES] RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	READ [INTO] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	READ [INTO] [INVALID KEY] WRITE [FROM] [INVALID KEY] REWRITE [FROM] [INVALID KEY] DELETE [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
Dynamic	SELECT ASSIGN ORGANIZATION INDEXED ACCESS DYNAMIC RECORD KEY [PASSWORD]	ALTERNATE RECORD KEY [PASSWORD] [DUPLICATES] RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	USE {EXCEPTION/ERROR FOR DEBUGGING} (sequential request) READ NEXT [INTO] [AT END] START [KEY] [INVALID KEY] (random request) READ [INTO] [INVALID KEY]
					OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	USE {EXCEPTION/ERROR FOR DEBUGGING} (sequential request) READ NEXT [INTO] [AT END] START [KEY] [INVALID KEY] (random request) READ [INTO] [INVALID KEY] WRITE [FROM] [INVALID KEY] REWRITE [FROM] [INVALID KEY] DELETE [INVALID KEY]

Figure 40. VSAM Indexed Direct Access Storage Files—Required and Optional Entries

Access	ENVIRONMENT DIVISION		DATA DIVISION		PROCEDURE DIVISION	
	Required Entries	Optional Entries	Required Entries	Optional Entries	Required Entries	Optional Entries
Sequen- tial	SELECT ASSIGN ORGANIZATION RELATIVE	ACCESS SEQUENTIAL RELATIVE KEY PASSWORD RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	READ [NEXT] [INTO] [AT END] START [KEY] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	READ [NEXT] [INTO] [AT END] REWRITE [FROM] [INVALID KEY] START [KEY] [INVALID KEY] DELETE USE {EXCEPTION/ERROR FOR DEBUGGING}
Random	SELECT ASSIGN ORGANIZATION RELATIVE ACCESS RANDOM RELATIVE KEY	PASSWORD RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	READ [INTO] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	READ [INTO] [INVALID KEY] WRITE [FROM] [INVALID KEY] REWRITE [FROM] [INVALID KEY] DELETE [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
Dynamic	SELECT ASSIGN ORGANIZATION RELATIVE ACCESS DYNAMIC RELATIVE KEY	PASSWORD RESERVE FILE STATUS RERUN...RECORDS SAME [RECORD] AREA	LABEL RECORDS {STANDARD OMITTED}	BLOCK CONTAINS RECORD CONTAINS DATA RECORDS	OPEN INPUT CLOSE [LOCK]	USE {EXCEPTION/ERROR FOR DEBUGGING} (sequential request) READ NEXT [INTO] [AT END] START [KEY] [INVALID KEY] (random request) READ [INTO] [INVALID KEY]
					OPEN OUTPUT CLOSE [LOCK]	WRITE [FROM] [INVALID KEY] USE {EXCEPTION/ERROR FOR DEBUGGING}
					OPEN I-O CLOSE [LOCK]	USE {EXCEPTION/ERROR FOR DEBUGGING} (sequential request) READ NEXT [INTO] [AT END] START [KEY] [INVALID KEY] (random request) READ [INTO] [INVALID KEY] WRITE [FROM] [INVALID KEY] REWRITE [FROM] [INVALID KEY] DELETE [INVALID KEY]

Figure 41. VSAM Relative Direct Access Storage Files—Required and Optional Entries

COMMON PROCESSING FACILITIES

The common processing facilities provided are: status key, invalid key condition, INTO/FROM identifier option, and current record pointer. All are discussed in following sections.

Status Key

If the FILE STATUS clause is specified in the File-Control entry, a value is placed in the specified Status Key (the 2-character data item named in the FILE STATUS clause) during execution of any request on that file; the value indicates the status of that request. The value is placed in the Status Key before execution of any EXCEPTION/ERROR Declarative or INVALID KEY/AT END option associated with the request.

The first character of the Status Key is known as Status Key 1; the second character is known as Status Key 2. For VSAM files, combinations of possible values and their meanings are shown in Figure 42; for physical sequential files, these combinations are shown in Figure 43.

Status Key 1 Value	Meaning	Status Key 2 Value	Meaning
0	Successful completion	0	No further information
		2	Duplicate key, and DUPLICATES specified
1	At End (no next logical record, or an OPTIONAL file not available at OPEN time)	0	No further information
2	Invalid Key	0	No further information
		1	Sequence error
		2	Duplicate key, and duplicate keys not allowed
		3	No record found
		4	Boundary violation (indexed or relative file)
3	Permanent error (data check, parity check, transmission error)	0	No further information
		4	Boundary violation (sequential file)
9	Other Errors	0	No further information
		1	Password failure
		2	Logic error
		3	Resource not available
		4	No current record pointer for sequential request
		5	Invalid or incomplete file information
		6	No file identification (see System Dependencies chapter)

Figure 42. Status Key Values and Meanings—VSAM Files

Status Key 1 Value	Meaning	Status Key 2 Value	Meaning
0	Successful Completion	0	No further information
1	At End (no next logical record, or an OPTIONAL file not available at OPEN time)	0	No further information
3	Permanent Error (data check, parity check, transmission error)	0	No further information
		4	Boundary violation
9	Other Errors	0	No further information
		2	Logic error
		6	No file identification (see System Dependencies chapter)

Figure 43. Status Key Values and Meanings—Physical Sequential Files

Invalid Key Condition

The INVALID KEY condition can occur during execution of a START, READ, WRITE, REWRITE, or DELETE statement. (For details of the causes for the condition, see the discussions of those statements.) When the INVALID KEY condition is recognized, the following actions are taken in the following order:

1. If the FILE-STATUS clause is specified in the File-Control entry, a value is placed into the Status Key to indicate an INVALID KEY condition (see Figure 41 and Figure 42).
2. If the INVALID KEY option is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not executed.
3. If the INVALID KEY option is not specified, but an EXCEPTION/ERROR declarative procedure is specified for the file, the EXCEPTION/ERROR procedure is executed.

When an INVALID KEY condition occurs, the input/output statement which caused the condition is unsuccessful.

If the INVALID KEY option is not specified for a file, an EXCEPTION/ERROR procedure must be specified.

IBM Extension

However, this implementation allows both the INVALID KEY option and the EXCEPTION/ERROR procedure to be omitted.

End of IBM Extension

INTO/FROM Identifier Option

This option is valid for READ, REWRITE, and WRITE statements. The specified identifier must be the name of an entry in the Working-Storage Section, the Linkage Section, or of a record description for another previously opened file. Record-name/file-name and identifier must not refer to the same storage area. The statements take the following form:

```
┌                                ┐
│   READ file-name RECORD INTO   │
│                                │
│ ┌         ┐                    │
< │ WRITE   │                    > identifier
│ <         > record-name FROM   │
│ │ REWRITE │                    │
└ └         ┘                    ┘
```

The INTO identifier option makes a READ statement equivalent to

```
    READ file-name

    MOVE record-name TO identifier
```

After successful execution of the READ statement, the current record becomes available in both the record-name and the identifier.

The FROM identifier option makes a REWRITE or WRITE statement equivalent to

```
  MOVE identifier TO record-name
┌         ┐
│ REWRITE │
<         > record-name
│ WRITE   │
└         ┘
```

After successful execution of the WRITE or REWRITE statement, the current record may no longer be available in record-name, but is still available in identifier.

In both options, the implicit move is executed according to MOVE statement rules without the CORRESPONDING option.

Current Record Pointer

Conceptually, the current record pointer identifies a particular record accessed by a sequential input request; the record identified depends on the statement being executed:

- The OPEN statement positions the current record pointer at the first record in the file.
- For the READ statement, the following considerations apply:
  - If the OPEN statement positioned the current record pointer, the record identified by the current record pointer is made available.
  - If a previous READ statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file; that record is then made available.
- The START statement positions the current record pointer at the first record in the file that satisfies the implicit or explicit comparison specified in the START statement.

The setting of the Current Record Pointer is affected only by the OPEN, START, and READ statements. The concept of the Current Record Pointer has no meaning for random access or for output files.

OPEN STATEMENT

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels, and other input/output operations.

Format 1—Sequential Files

```
        ┌                                                      ┐
        │                         ┌                  ┐         │
        │                         │ REVERSED         │         │
        │ INPUT file-name-1       │                  │         │
        │                         │ WITH NO REWIND   │         │
        │                         └                  ┘         │
        │                                                      │
        │                         ┌                  ┐         │
        │                         │ REVERSED         │         │
        │       [file-name-2      │                  │ ] ...   > ...
OPEN   <                          │ 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] ...                 │
        └                                                      ┘
```

Format 2—VSAM Indexed Files

```
       ┌                                        ┐
       │ INPUT file-name-1 [file-name-2] ...    │
       │                                        │
       │ OUTPUT file-name-3 [file-name-4] ...   │
       │                                        │
OPEN  <  I-O file-name-5 [file-name-6] ...      >
       │ ┌──────────────────────────────────────┐│
       │ │ EXTEND file-name-7 [file-name-8] ... ││
       │ └──────────────────────────────────────┘│
       └                                        ┘
```

Format 3—VSAM Relative Files

```
       ┌                                        ┐
       │ INPUT file-name-1 [file-name-2] ...    │
OPEN   │ OUTPUT file-name-3 [file-name-4] ...   │   ...
       │ I-O file-name-5 [file-name-6] ...      │
       └                                        ┘
```

Each file-name designates a file upon which the OPEN statement is to operate. The specified files need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be equivalent to the information supplied when the file was defined.

The successful execution of an OPEN statement determines the availability of the file and results in that file being in open mode. Before successful execution of the OPEN statement for a given file, no statement—except for a SORT or MERGE statement with the USING or GIVING option—can be executed which refers explicitly or implicitly to that file. The successful execution of the OPEN statement makes the associated record area available to the program; it does not obtain or release the first data record.

At least one of the options INPUT, OUTPUT, I-O, or EXTEND must be specified; there may not be more than one specification of each option, although more than one file-name may be specified in each option. The INPUT, OUTPUT, I-O, or EXTEND options may appear in any order.

The INPUT option permits opening of the file for input operations.

The I-O option permits opening of the file for both input and output operations. The I-O option may be specified only for files assigned to direct access storage devices.

The INPUT and I-O options must not be specified when the file has not already been created. See the System Dependencies chapter.

The OUTPUT option permits opening of the file for output operations. This option can be specified when the file is being created. (The OUTPUT option must not be specified for a file which contains records, or which has contained records that have been deleted.)

The EXTEND option is valid only for sequential files and permits opening the file for output operations. It is discussed in the following section on Sequential Files.

IBM Extension

This IBM implementation also allows the EXTEND option to be specified for VSAM indexed files.

End of IBM Extension

A file may be opened for INPUT, OUTPUT, I-O, or EXTEND in the same program. After the first OPEN statement execution for a given file, each subsequent OPEN statement execution must be preceded by a successful CLOSE file statement execution without the LOCK option.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the OPEN statement is executed.

If an OPEN statement is issued for a file already in the open status, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

When opening a file, the NO REWIND option has no effect on file positioning. It appears in the format for language consistency. When either NO REWIND or no option is specified, positioning of a file is specified with the LABEL parameter of the DD statement. (NO REWIND on the OPEN statement specifies no rewinding at close time.)

Format 1—Sequential Files

The EXTEND option permits opening a sequentially accessed indexed file for output operations. When an OPEN EXTEND statement is executed, the file is prepared for the addition of records immediately following the last record in the file. (The record with the highest prime record key value is considered the last record.) Subsequent WRITE statements add records as if the file were opened for OUTPUT. The EXTEND option can be specified when a file is being created; it can also be specified for a file that contains records, or that has contained records which have been deleted.

Execution of an OPEN INPUT or OPEN I-O statement sets the Current Record Pointer to the first record existing in the file. If no records exist in the file, the Current Record Pointer is set so that execution of the first READ statement results in an AT END condition.

For an INPUT file, if SELECT OPTIONAL is specified in the File-Control entry, OPEN statement execution causes the object program to check for the presence or absence of this file. If the file is absent, the first READ statement causes the AT END condition to occur. (See the chapter on System Dependencies.)

PHYSICAL SEQUENTIAL FILES: The REVERSED and NO REWIND options are valid for this type of file; these options are valid only for single reel files. (See the System Dependencies chapter.)

The following rules apply:

- When the REVERSED, NO REWIND, or EXTEND options are not specified, OPEN statement execution positions the file at its beginning.
- When NO REWIND is specified, OPEN statement execution does not reposition the file; prior to OPEN statement execution, the file must be positioned at its beginning.
- When REVERSED is specified, OPEN statement execution positions the file at its end. Subsequent READ statements make the data records available in the reverse order, starting with the last record.

If the concept of reels has no meaning for the storage medium (for example, a direct access storage device), the REVERSED and NO REWIND options do not apply.

If label records are specified for the file, when the OPEN statement is executed, the labels are processed according to the standard label conventions, as follows:

- INPUT files: the beginning labels are checked
- OUTPUT files: the beginning labels are written
- I-O files: the labels are checked; new labels are then written.
- EXTEND files: the following procedures are executed:
    1. Beginning file labels are processed only if this is a single-volume file.
    2. Beginning volume labels of the last existing volume are checked.
    3. Existing ending file labels are checked; they are then deleted.
    4. Processing continues as if the file were opened as an OUTPUT file.

When label records are specified but not present, or when label records are present but not specified, execution of the OPEN statement is unpredictable.

VSAM SEQUENTIAL FILES: The concept of reels has no meaning in VSAM file organization; therefore, the REVERSED and NO REWIND options must not be specified.

Label processing is not performed for VSAM sequential files.

IBM Extension

If the PASSWORD clause is specified in the File-Control entry, the password data item must contain the valid password before the OPEN statement is executed. If the valid password is not present, the OPEN statement execution is unsuccessful.

End of IBM Extension

Format 2—VSAM Indexed Files

Execution of an OPEN INPUT or OPEN I-O statement sets the Current Record Pointer to the first record existing in the file; the record with the lowest prime record key value is considered to be the first record. If no records exist in the file, the

Current Record Pointer is set so that the first Format 1 READ statement executed results in an AT END condition.

IBM Extension

The EXTEND option permits opening a sequentially accessed indexed file for output operations. When an OPEN EXTEND statement is executed, the file is prepared for the addition of records immediately following the last record in the file. (The record with the highest prime record key value is considered the last record.) Subsequent WRITE statements add records as if the file were opened for OUTPUT. The EXTEND option can be specified when a file is being created; it can also be specified for a file that contains records, or that has contained records which have been deleted.

For most types of file processing, every password data item specified for an indexed file must contain a valid password before the OPEN statement can be successfully executed. For details, see the System Dependencies chapter on the PASSWORD clause.

End of IBM Extension

Format 3—VSAM Relative Files

Execution of an OPEN INPUT or OPEN I-O statement sets the Current Record Pointer to the first record existing in the file; the record with the lowest relative record number is considered the first record in the file. If no records exist in the file, the Current Record Pointer is set so that the first Format 1 READ statement executed results in an AT END condition.

IBM Extension

If the PASSWORD clause is specified in the File-Control entry for this file, the password data item must contain the valid password before the OPEN statement is executed. If the valid password is not present, the OPEN statement execution is unsuccessful.

End of IBM Extension

WRITE STATEMENT

The WRITE statement releases a logical record for an output or input/output file.

Format 1—Physical Sequential Files

```
WRITE record-name [FROM identifier]
                              ┌   ┌ identifier-2 ┐  ┌ LINE  ┐  ┐
                              |   <              >  | LINES |  |
                              |   | integer      ┘  └       ┘  |
   ┌ BEFORE ┐                 |   └                            |
 [ <        > ADVANCING <     ┌                  ┐             > ]
   | AFTER  |                 |   | mnemonic-name |            |
   └        ┘                 |   <               >            |
                              |   | PAGE          |            |
                              └   └               ┘            ┘

      ┌ END-OF-PAGE ┐
  [AT <             >  imperative-statement]
      | EOP         |
      └             ┘
```

Format 2—VSAM Sequential Files

WRITE record-name [FROM identifier]

Format 3—VSAM Indexed and Relative Files

WRITE record-name [FROM identifier]

[INVALID KEY imperative-statement]

When the WRITE statement is executed, the associated file must be open in OUTPUT, the I-O, or EXTEND mode.

Record-name must be the name of a logical record in the File Section of the Data Division. Record-name may be qualified. Record-name must not be associated with a sort or merge file.

The maximum record size for the file is established at the time the file is created, and cannot subsequently be changed.

Execution of the WRITE statement releases a logical record to the file associated with record-name.

After the WRITE statement is executed, the logical record is no longer available in record-name, unless:

- The associated file is named in a SAME RECORD AREA clause (in which case the record is also available as a record of the other files named in the SAME RECORD AREA clause), or

- The WRITE statement is unsuccessful because of a boundary violation.

In either of these two cases, the logical record is still available in record-name.

If the FROM identifier option is specified, then after the WRITE statement is executed, the information is still available in identifier, even though it may not be in record-name. (See "INTO/FROM identifier Option" in the preceding Common Processing Facilities Section.)

The Current Record Pointer is not affected by execution of the WRITE statement.

The number of character positions required to store the record in a file may or may not be the same as the number of character positions defined by the logical description of that record in the COBOL program. (See the descriptions of the PICTURE and USAGE clauses in the Data Division chapter.)

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the WRITE statement is executed, whether or not execution is successful.

Format 1—Physical Sequential Files

When an attempt is made to write beyond the externally defined boundaries of the file, WRITE statement execution is unsuccessful and an EXCEPTION/ERROR condition exists. The status key, if specified, is updated, and if an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is executed; if no such procedure is specified, the results are unpredictable.

The ADVANCING and END-OF-PAGE options control the vertical positioning of each line on a printed page.

ADVANCING OPTION: When this option is omitted, automatic line advancing is provided as if the user had written AFTER ADVANCING 1 LINE.

When this option is specified, the following rules apply:

- When BEFORE ADVANCING is specified, the line is printed before the page is advanced.

- When AFTER ADVANCING is specified, the page is advanced before the line is printed.

- When identifier-2 is specified, the page is advanced the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item.

- When integer is specified, the page is advanced the number of lines equal to the value of integer.

- Integer or the value in identifier-2 may be zero.

- When mnemonic-name is specified, a skip to channels 1 through 9, 10 through 12, or space suppression takes place. Mnemonic-name must be equated with function-name-1 in the SPECIAL-NAMES paragraph (valid function-names are listed in the System Dependencies chapter). This option is not valid if a LINAGE clause is specified in the FD entry for this file.

IBM Extension

The mnemonic-name option may also be specified for stacker selection with a card punch file. When using stacker selection, WRITE AFTER ADVANCING must be used.

End of IBM Extension

- When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the option used) the device is positioned to the next logical page. If PAGE has no meaning for the device used, then BEFORE or AFTER (depending on the option specified) ADVANCING 1 LINE is provided.

  If the FD entry contains a LINAGE clause, the repositioning is to the first printable line of the next page, as specified in that clause. If the LINAGE clause is omitted, the repositioning is to channel 1 of the carriage control tape (that is, to line 1 of the next succeeding page).

See the System Dependencies chapter.

LINAGE-COUNTER Rules: If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the execution of the WRITE statement, according to the following rules:

- If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1.

- If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is incremented by the value in identifier-2 or integer.

- If the ADVANCING option is omitted, LINAGE-COUNTER is incremented by 1.

- When the device is repositioned to the first writable line of a new page, LINAGE-COUNTER is reset to 1.

END-OF-PAGE OPTION: The key words END-OF-PAGE and EOP are equivalent.

When the END-OF-PAGE option is specified, the FD entry for this file must contain a LINAGE clause. When END-OF-PAGE is specified, and the logical end of the printed page is reached during execution of the WRITE statement, the END-OF-PAGE imperative-statement is executed.

The logical end of the printed page is specified in the associated LINAGE clause.

An END-OF-PAGE condition is reached when execution of a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when execution of such a WRITE statement causes the value in the LINAGE-COUNTER special register to equal or exceed the value specified in the WITH FOOTING option of the LINAGE clause. The WRITE statement is executed, and then the END-OF-PAGE imperative-statement is executed.

An automatic page overflow condition is reached when the execution of any given WRITE statement (with or without the END-OF-PAGE option) cannot be completely executed within the current page body. This occurs when a WRITE statement, if executed, would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed BEFORE or AFTER (depending on the option specified) the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause. If the END-OF-PAGE option is specified, the END-OF-PAGE imperative-statement is then executed.

If the WITH FOOTING option of the LINAGE clause is not specified, the END-OF-PAGE condition and the automatic page overflow condition occur simultaneously, since no end-of-page condition as distinct from the page overflow condition can be detected.

If the WITH FOOTING option is specified, but the execution of a given WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause, then both the end-of-page condition and the automatic page overflow condition occur simultaneously.

MULTIVOLUME FILES: The following discussion applies to all multivolume tape files and multivolume physical sequential direct access storage files.

When end-of-volume is recognized for a multivolume OUTPUT file, the WRITE statement performs the following operations:

- The standard ending volume label procedure
- A volume switch
- The standard beginning volume label procedure

Format 2—VSAM Sequential Files

The INVALID KEY option must not be specified.

When an attempt is made to write beyond the externally defined boundaries of the file, the execution of the WRITE statement is unsuccessful, and an EXCEPTION/ERROR condition exists. The contents of record-name are unaffected. If specified, the status key is updated, and, if an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is executed; if no such procedure is specified, the results are unpredictable.

Format 3—VSAM Indexed and Relative files

This format is valid only for VSAM indexed and relative files.

VSAM INDEXED FILES: When the WRITE statement is executed, the system releases the record, using the record keys specified so that later access to the record can be based on any of the specified keys. Before the WRITE statement is executed, the user must set the prime record key (the RECORD KEY data item, as defined in the File-Control entry) to the desired value. (Note that RECORD KEY values must be unique within a file.)

If the ALTERNATE RECORD KEY clause is also specified in the File-Control entry, each alternate record key must be unique, unless the DUPLICATES option is specified. If the DUPLICATES option is specified, alternate record key values may be nonunique. In this case, the system stores the records so that later sequential access to the records allows retrieval in the same order they were stored.

When ACCESS IS SEQUENTIAL is specified in the File-Control entry, records must be released in ascending order of RECORD KEY values.

When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified in the File-Control entry, records may be released in any programmer-specified order.

INVALID KEY Option: This option must be specified if an explicit or implicit EXCEPTION/ERROR procedure is not specified for this file.

IBM Extension

However, this IBM implementation allows both the INVALID KEY option and the EXCEPTION/ERROR procedure to be omitted.

End of IBM Extension

When the INVALID KEY condition is recognized, WRITE statement execution is unsuccessful, and the contents of the record are unaffected. Program execution proceeds according to the rules stated in the preceding INVALID KEY Condition section. An INVALID KEY condition is caused by any of the following:

- When ACCESS SEQUENTIAL is specified, and the file is opened OUTPUT, and the value of the prime record key is not greater than that for the previous record.

- When the file is opened I-O, and the value of the prime record key equals that of an already existing record.

- When the file is opened OUTPUT or I-O, and the value of an ALTERNATE RECORD KEY for which duplicates are not allowed equals that of an already existing record.

- When an attempt is made to write beyond the externally-defined boundaries of the file.

IBM Extension

- The INVALID KEY conditions that apply to an indexed file in OPEN OUTPUT mode also apply to one in OPEN EXTEND mode.

End of IBM Extension

VSAM RELATIVE FILES: The WRITE statement is valid for both OUTPUT and I-O files.

For OUTPUT files, the WRITE statement causes the following actions:

- If ACCESS IS SEQUENTIAL is specified, the first record released has relative record number 1, the second number 2, the third number 3, and so forth. If the RELATIVE KEY is specified in the File-Control entry, the relative record number of the record just released is placed in the RELATIVE KEY during execution of the WRITE statement.

- If ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

For I-O files, either ACCESS IS RANDOM or ACCESS IS DYNAMIC must be specified; the WRITE statement inserts new records into the file. The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is executed, this record is placed at the specified relative record number position in the file.

INVALID KEY Option: This option must be specified if an explicit or implicit EXCEPTION/ERROR procedure is not specified for this file.

IBM Extension

However, this IBM implementation allows both the INVALID KEY option and the EXCEPTION/ERROR procedure to be omitted.

End of IBM Extension

When the INVALID KEY condition is recognized, WRITE statement execution is unsuccessful, and the contents of the record area are unaffected. Program execution proceeds according to the rules stated in the preceding INVALID KEY Condition section. An INVALID KEY condition is caused by either of the following:

- When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, and the RELATIVE KEY specifies a record that already exists in the file

- When an attempt is made to write beyond the externally-defined boundaries of the file

START STATEMENT

The START statement provides a means of positioning within a VSAM indexed or relative file for subsequent sequential record retrieval.

Format

```
                           ┌                ┐
                           | EQUAL TO       |
                           | =              |
                           | GREATER THAN   |
START file-name  [KEY IS < >               > data-name]
                           | NOT LESS THAN  |
                           | NOT <          |
                           └                ┘

  [INVALID KEY imperative-statement]
```

When the START statement is executed, the associated VSAM indexed or relative file must be open in INPUT or I-O mode.

File-name must name a file with sequential or dynamic access. File-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file.

When the KEY option is not specified, the EQUAL TO relational operator is implied.

When the KEY option is specified, the comparison specified in the KEY relational operator is made between data-name and the corresponding key field associated with the file's records. Data-name may be qualified; it may not be subscripted or indexed.

When the START statement is executed, a comparison is made between the current value in the key data-name and the corresponding key field in the file's records. The Current Record Pointer is positioned to the logical record in the file whose key field satisfies the comparison.

INVALID KEY Option

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists; the position of the Current Record Pointer is undefined, and (if specified) the INVALID KEY imperative-statement is executed. (See the description of the INVALID KEY Condition in the preceding Common Processing Facilities section.)

The INVALID KEY option must be specified if no EXCEPTION/ERROR procedure is explicitly or implicitly specified for this file.

IBM Extension

However, this IBM implementation allows both the INVALID KEY option and the EXCEPTION/ERROR procedure to be omitted.

End of IBM Extension

Status Key Considerations

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the START statement is executed. (See the Status Key description in the preceding Common Processing Facilities section.)

VSAM Indexed Files

When the KEY option is not specified, the key data item used for the EQUAL TO comparison is the prime RECORD KEY. When START statement execution is successfully completed, the RECORD KEY becomes the key of reference for subsequent READ statements.

When the KEY option is specified, the key data item used for the comparison is data-name, which can be:

- The prime RECORD KEY, or
- Any ALTERNATE RECORD KEY, or
- An alphanumeric data item subordinate to a record key whose leftmost character position corresponds to the leftmost character position of that record key. This data item may be qualified.

IBM Extension

This data item may also be defined as an external decimal item.

End of IBM Extension

The Current Record Pointer is positioned to the first record in the file whose key field satisfies the comparison. If the operands in the comparison are of unequal length, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply, except that the PROGRAM COLLATING SEQUENCE clause, if specified, has no effect.

When START statement execution is successful, the RECORD KEY or ALTERNATE RECORD KEY with which data-name is associated becomes the key of reference for subsequent READ statements.

When START statement execution is unsuccessful, the key of reference is undefined.

VSAM Relative Files

When the KEY option is specified, data-name must specify the RELATIVE KEY.

Whether or not the KEY option is specified, the key data item used in the comparison is the RELATIVE KEY data item. The Current Record Pointer is positioned to the logical record in the file whose key satisfies the comparison.

READ STATEMENT

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a direct access storage file.

Format 1—Sequential Retrieval

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

Format 2—Random Retrieval

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

The READ statement makes a record available to the object program before execution of any statement following the READ statement.

When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must be defined in a Data Division FD entry, and must not name a sort or merge file.

If more than one record description entry is associated with file-name, these records automatically share the same storage area; that is, they are implicitly redefined. After a READ statement is executed, only those data items within the range of the current record are replaced; data items stored beyond that range are undefined. Figure 44 illustrates this concept.

The FD entry is:

```
FD  INPUT-FILE LABEL RECORDS OMITTED.

01  RECORD-1  PICTURE X(30).

01  RECORD-2  PICTURE X(20).
```

Contents of input area when READ statement is executed:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
```

Contents of record being read in (RECORD-2):

```
01234567890123456789
```

Contents of input area after READ is executed:

```
01234567890123456789??????????
                    |________|
                   (These characters in input area are
                    undefined)
```

Figure 44. READ Statement with Multiple Record Description

When the INTO identifier option is specified, the current record is moved from the input area to the identifier area according to the rules for the MOVE statement without the CORRESPONDING option. Any subscripting or indexing associated with the identifier is evaluated after the record has been read and immediately before it is transferred to identifier. (See INTO/FROM Identifier Option in the preceding Common Processing Facilities section.)

The INTO identifier option must not be specified when the file contains records of various sizes, as indicated by their record descriptions.

IBM Extension

However, this IBM implementation allows the use of records of various sizes with READ INTO.

End of IBM Extension

The AT END or INVALID KEY option must be specified if no implicit or explicit EXCEPTION/ERROR procedure is specified for this file.

IBM Extension

However, this IBM implementation allows both the AT END or INVALID KEY option and the EXCEPTION/ERROR procedure to be omitted.

End of IBM Extension

If the FILE-STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the READ statement is executed.

Following unsuccessful READ statement execution, the contents of the associated record area and the position of the Current Record Pointer are undefined.

Sequential Access

Format 1 must be used for all files in sequential access mode.

Execution of a Format 1 READ statement makes available the next logical record from the file. Which record is considered next depends upon the file organization.

FILES WITH SEQUENTIAL ORGANIZATION: The next record is the succeeding record in logical sequence. The NEXT option need not be specified; it has no effect on READ statement execution.

If SELECT OPTIONAL is specified in the File-Control entry for this file, and the file is absent during this execution of the object program, execution of the first READ statement causes an AT END condition; however, since no file is present, the standard end-of-file processing is not performed.

AT END Condition: If no next logical record exists in the file when the READ statement is executed, an AT END condition occurs, and READ statement execution is unsuccessful. The following actions are taken, in the following order:

1. If the FILE STATUS clause is specified, the Status Key is updated to indicate an AT END condition.

2. If the AT END option is specified, control is transferred to the AT END imperative-statement. Any EXCEPTION/ERROR procedure for this file is not executed.

3. If the AT END option is not specified, then any EXCEPTION/ERROR procedure for this file is executed.

When the AT END condition is recognized, a READ statement for this file must not be executed without first executing a successful CLOSE statement followed by a successful OPEN statement for this file.

Multivolume Physical Sequential Files: If end-of-volume is recognized during execution of a READ statement, and logical end-of-file has not been reached, the following are accomplished:

- The standard ending volume label procedure.

- A volume switch.

- The standard beginning volume procedure.

- The first data record of the next volume is made available.

FILES WITH VSAM INDEXED OR VSAM RELATIVE ORGANIZATION: The next record is the succeeding logical record in key sequence. For VSAM indexed files, the key sequence is the ascending values of the current key of reference. For VSAM relative files, the key sequence is the ascending values of relative record numbers for records that exist in the file.

Before the READ statement is executed, the Current Record Pointer must be set by a successful OPEN, START, or READ statement. When the READ statement is executed, the record indicated by the Current Record Pointer is made available, if it is still accessible through the path indicated by the Current Record Pointer.

If the record is no longer accessible (due, for example, to deletion of the record), the Current Record Pointer is updated to indicate the next existing record in the file, and that record is made available.

For files in sequential access mode, the NEXT option need not be specified; it may be specified.

For files in dynamic access mode, the NEXT option must be specified for sequential record retrieval.

AT END Condition: If no more logical records exist in the file when the READ statement is executed, an AT END condition occurs. The same actions are taken as for files with sequential organization.

When the AT END condition is recognized, a sequential access READ statement for this file must not be executed without first executing one of the following:

- A successful CLOSE statement followed by a successful OPEN statement
- A successful START statement for this file
- A successful random access READ statement for this file

Sequentially Accessed VSAM Indexed Files: When an ALTERNATE RECORD KEY with DUPLICATES is the key of reference, file records with duplicate key values are made available in the order they were placed in the file.

Sequentially Accessed VSAM Relative Files: If the RELATIVE KEY clause is specified for this file, READ statement execution updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

Random Access

Format 2 must be specified for VSAM indexed and relative files in random access mode, and also for files in the dynamic access mode when record retrieval is random.

Execution of the READ statement depends on the file organization, as explained in the following sections.

FILES WITH VSAM INDEXED ORGANIZATION: Execution of a Format 2 READ statement causes the value of the key of reference to be compared with the value of the corresponding key data item in the file records, 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, an INVALID KEY condition exists, and READ statement execution is unsuccessful. (See INVALID KEY condition in the preceding Common Processing Facilities section.)

If the KEY option is not specified, the prime RECORD KEY becomes the key of reference for this request. When dynamic access is specified, the prime RECORD KEY is also used as the key of reference for subsequent executions of sequential READ statements until a different key of reference is established.

KEY Option: The KEY option may be specified only for indexed files. Data-name must identify a record key associated with file-name—either the prime record key or any alternate record key. Data-name may be qualified; it may not be subscripted or indexed.

When the KEY option is specified, data-name becomes the key of reference for this request. When dynamic access is specified, this key of reference is used for subsequent executions of sequential READ statements until a different key of reference is established.

FILES WITH VSAM RELATIVE ORGANIZATION: Execution of a Format 2 READ statement sets the Current Record Pointer to the record whose relative record number is contained in the RELATIVE KEY data item, and makes that record available.

If the file does not contain such a record, the INVALID KEY condition exists, and READ statement execution is unsuccessful. (See INVALID KEY Condition in the preceding Common Processing Facilities section.)

The KEY option must not be specified for relative files.

Dynamic Access

For files with VSAM indexed or relative organization, dynamic access mode may be specified in the File-Control entry. In dynamic access mode, either sequential or random record retrieval can be specified, depending on the format used.

Format 1 with the NEXT option must be specified for sequential retrieval. All other rules for sequential access apply.

Format 2 must be specified for random retrieval. All other rules for random access apply.

REWRITE STATEMENT

The REWRITE statement logically replaces an existing record in a mass storage file.

Format

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

When the REWRITE statement is executed, the associated direct access storage file must be open in I-O mode.

Record-name must be the name of a logical record in the File Section of the Data Division. Record-name must not be associated with a sort or merge file. Record-name may be qualified; it may not be subscripted or indexed.

REWRITE statement execution replaces an existing record in the file with the information contained in record-name.

After successful execution of a REWRITE statement, the logical record is no longer available in record-name unless the associated file is named in a SAME RECORD AREA clause (in which case the record is also available as a record of the other files named in the SAME RECORD AREA clause).

The Current Record Pointer is not affected by execution of the REWRITE statement.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the REWRITE statement is executed.

For files in the sequential access mode, the last prior input/output statement executed for this file must be a successfully executed READ statement. When the REWRITE statement is executed, the record retrieved by that READ statement is logically replaced.

When the FROM identifier option is specified, the logical record is still available in identifier after successful REWRITE statement execution. (See the INTO/FROM Identifier Option in the preceding Common Processing Facilities section.)

Sequential Files

The number of character positions in record-name must equal the number of character positions in the record being replaced.

The INVALID KEY option must not be specified for a file with sequential organization. An EXCEPTION/ERROR procedure may be specified.

VSAM Indexed Files

The number of character positions in record-name must equal the number of character positions in the record being replaced.

IBM Extension

However, this IBM implementation allows the number of character positions in record-name to be different from the number of character positions in the record being replaced.

End of IBM Extension

When the access mode is sequential, the record to be replaced is specified by the value contained in the prime RECORD KEY. When the REWRITE statement is executed, this value must equal the value of the prime record key data item in the last record read from this file.

When the access mode is random or dynamic, the record to be replaced is specified by the value contained in the prime RECORD KEY.

Values of ALTERNATE RECORD KEY data items in the rewritten record may differ from those in the record being replaced. The system ensures that later access to the record can be based upon any of the record keys.

INVALID KEY OPTION: An INVALID KEY condition exists when:

- The access mode is sequential, and the value contained in the prime RECORD KEY of the record to be replaced does not equal the prime RECORD KEY data item of the last-retrieved record from the file, or
- The value contained in the prime RECORD KEY does not equal that of any record in the file, or
- The value of an ALTERNATE RECORD KEY data item for which DUPLICATES is not specified is equal to that of a record already in the file.

If any one of these conditions exists, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the data in record-name is unaffected. (See "INVALID KEY Condition" in the preceding Common Processing Facilities section.)

VSAM Relative Files

The number of character positions in record-name must equal the number of character positions in the record being replaced.

IBM Extension

However, this IBM implementation allows the number of character positions in record-name to be different from the number of character positions in the record being replaced.

End of IBM Extension

For VSAM relative files in the sequential access mode, the INVALID KEY option must not be specified. An EXCEPTION/ERROR procedure may be specified.

When the access mode is random or dynamic, the record to be replaced is specified in the RELATIVE KEY data item. If the file does not contain the specified record, an INVALID KEY condition exists, and, if specified, the INVALID KEY imperative-statement is executed. (See INVALID KEY condition in the preceding Common Processing Facilities section.) The updating operation does not take place, and the data in record-name is unaffected.

DELETE STATEMENT

The DELETE statement logically removes a record from an indexed or a relative file.

```
Format DELETE file-name  RECORD
    [INVALID KEY imperative-statement]
```

When the DELETE statement is executed, the associated file must be open in I-O mode.

File-name must be defined in an FD entry in the Data Division and must be the name of an indexed or relative VSAM file.

After successful execution of a DELETE statement, the record is logically removed from the file and can no longer be accessed. Execution of the DELETE statement does not affect the contents of the record area associated with file-name.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the DELETE statement is executed.

Sequential Access Mode

For a file in sequential access mode, the last prior input/output statement must be a successfully executed READ statement. When the DELETE statement is executed, the system logically removes the record retrieved by that READ statement. The current record pointer is not affected by execution of the DELETE statement.

For a file in sequential access mode, the INVALID KEY option must not be specified. An EXCEPTION/ERROR procedure may be specified.

Random or Dynamic Access Mode

In random or dynamic access mode, DELETE statement execution results depend on the file organization: VSAM indexed or relative.

VSAM INDEXED FILES: In random or dynamic access mode, when the DELETE statement is executed, the system logically removes the record identified by the contents of the prime RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists. (See the INVALID KEY Condition in the preceding Common Processing Facilities section.)

VSAM RELATIVE FILES: In random or dynamic access mode, when the DELETE statement is executed, the system logically removes the record identified by the contents of the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists. (See the INVALID KEY Condition in the preceding Common Processing Facilities section.)

Programming Notes

The DELETE statement logically removes the record from the file. For indexed files, the space is then immediately available for record additions. For relative files, the space is then available for a new record with the same RELATIVE KEY value.

CLOSE STATEMENT

The CLOSE statement terminates the processing of volumes and files, with optional rewind and/or lock or removal, where applicable.

Format 1—Physical Sequential Files

```
                     ┌ ┌      ┐   ┌                ┐ ┐
                     | | REEL |   | WITH NO REWIND | |
                     | < UNIT >   | FOR REMOVAL    | |
                     | |      |   |                | |
                     | └      ┘   └                ┘ |
                     |                               |
CLOSE file-name-1    |                               |
                     |                               |
                     |            ┌           ┐      |
                     |            | NO REWIND |      |
                     |   WITH     <           >      |
                     |            | LOCK      |      |
                     └            └           ┘      ┘

                     ┌ ┌      ┐   ┌                ┐ ┐
                     | | REEL |   | WITH NO REWIND | |
                     | < UNIT >   | FOR REMOVAL    | |
                     | |      |   |                | |
                     | └      ┘   └                ┘ |
                     |                               |
     [file-name-2    |                               | ] ...
                     |                               |
                     |            ┌           ┐      |
                     |            | NO REWIND |      |
                     |   WITH     <           >      |
                     |            | LOCK      |      |
                     └            └           ┘      ┘
```

Format 2—VSAM Files

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

Each file-name designates a file upon which the CLOSE statement is to operate. The files need not have the same organization or access, and must not be sort or merge files.

A CLOSE statement may be executed only for a file in an open mode. After successful execution of a CLOSE statement without the REEL/UNIT option, the record area associated with the file-name is no longer available. Unsuccessful execution of a CLOSE statement leaves availability of the record data undefined.

After a CLOSE statement without the REEL/UNIT option is successfully executed for the file, an OPEN statement for the file must be executed before any other input/output statement (except a SORT/MERGE statement with the USING or GIVING option) can refer explicitly or implicitly to the file.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the CLOSE statement is executed.

If the file is in an open status and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

If a CLOSE statement is not executed for an open file before a STOP RUN or CANCEL statement for this program is executed, results are unpredictable. (See the System Dependencies chapter.)

Format 1—Physical Sequential Files

The REEL/UNIT options may be specified only for physical sequential multivolume files. The terms REEL and UNIT are interchangeable.

The WITH NO REWIND and FOR REMOVAL options apply only to tape files. If they are specified for storage devices to which they do not apply, they are ignored.

If the SELECT OPTIONAL clause is specified in the File-Control entry for this file, and the file is not present at object time, standard end-of-file processing is not performed. (See the chapter on System Dependencies.)

Figure 45 summarizes the permissible combinations of CLOSE statement options for each file type. Figure 46 gives the meaning of each key used in Figure 45 . Physical sequential files are divided into the following types:

- Unit Record: A file whose input or output medium is such that rewinding, units, and reels have no meaning.

- Sequential Single Volume: A sequential file entirely contained on one volume. More than one file may be contained on this volume.

- Sequential Multivolume: A sequential file contained on more than one volume.

	Physical Sequential File Type		
CLOSE Statement Format	**Unit Record**	**Sequential Single Volume**	**Sequential Multi-Volume**
CLOSE	C	C,G	C,G,A
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A
CLOSE WITH NO REWIND	X	C,B	C,B,A
CLOSE REEL/UNIT	X	X	F,G
CLOSE REEL/UNIT FOR REMOVAL	X	X	F,D,G
CLOSE REEL/UNIT WITH NO REWIND	X	X	F,B

Figure 45. Physical Sequential File Types and CLOSE Statement Options

Key	Actions Taken
A	*Previous Volumes Unaffected* *Input and Input/Output Files*—Standard volume-switch processing is performed for all prior volumes (except those controlled by a prior CLOSE REEL/UNIT statement). Any subsequent volumes are not processed. *Output Files*—Standard volume-switch processing is performed for all prior volumes (except those controlled by a prior CLOSE REEL/UNIT statement).
B	*No Rewind of Current Reel*—the current volume is left in its current position.
C	*Close File* *Input and Input/Output Files*—If the file is at its end, and label records are specified, the standard ending label procedure is performed. Standard system closing procedures are then performed. If the file is at its end, and label records are omitted, standard system closing procedures are performed. If the file is not at its end, standard system closing procedures are performed. Even if label records are specified, no label processing is performed. *Output Files*—If label records are specified, standard ending label procedures are performed. Standard system closing procedures are then performed. If label records are not specified, standard system closing procedures are performed.
D	*Volume Removal*—When applicable, the current volume is rewound; the system is notified that the volume is logically removed from this run unit. However, the volume may be accessed again, after execution of a CLOSE statement without the REEL/UNIT option and an OPEN statement for this file.
E	*File Lock*—The compiler ensures that this file cannot be opened again during this execution of the object program.
F	*Close Volume* *Input Files*—the following operations are performed: a volume switch, the standard beginning label procedure, and the first record on the new volume is made available for reading. *Output and Input/Output Files*—the following operations are performed: the standard ending volume label procedure (for output files only), a volume switch, the standard beginning volume label procedure. *For Input/Output Files,* the next-executed READ statement makes the next logical record on the next direct access storage volume available. *For Output Files,* the next-executed WRITE statement places the next logical record on the next direct access storage volume available.
G	*Rewind*—The current volume is positioned at its beginning.
X	*Illegal*—This combination of CLOSE statement option and file type is illegal. Results at object time may be unpredictable.

Figure 46. Meanings of Keys Used in Figure 45

Format 2—VSAM Files

Whether a VSAM file is contained on a single volume or on multiple volumes is of no concern to the COBOL programmer; thus the concept of volumes has no meaning for these types of files.

If an input VSAM sequential file is described as SELECT OPTIONAL in the File-Control entry and the file is not present during this execution of the object program, standard end-of-file processing is not performed.

Figure 47 summarizes the permissible combinations of CLOSE statement options for VSAM files. Figure 48 gives the meaning of each key used in Figure 47.

CLOSE Statement Format	VSAM Files (Sequential, Indexed, Relative)
CLOSE	A
CLOSE WITH LOCK	A,B

Figure 47. VSAM Files and CLOSE Statement Options

Key	Actions Taken
A	**Close File** Standard system closing procedures are performed.
B	**File Lock** The compiler ensures that this file cannot be opened again during this execution of the object program.

Figure 48. Meanings of Keys Used in Figure 47

ACCEPT STATEMENT

The ACCEPT statement causes low-volume data to be made available in the specified identifier.

Format 1—Data Transfer

```
                          ┌                     ┐
                          │   mnemonic-name     │
ACCEPT identifier [FROM  <  ┌───────────────┐   > ]
                          │ │ function-name │   │
                          │ └───────────────┘   │
                          └                     ┘
```

Format 2—System Information Transfer

```
                        ┌      ┐
                        │ DATE │
ACCEPT identifier FROM <  DAY   >
                        │ TIME │
                        └      ┘
```

ACCEPT statement execution causes the transfer of data into the specified identifier. There is no editing or error checking of the incoming data.

Format 1—Data Transfer

This format is used to transfer data from an input/output device into the identifier. When the FROM option is omitted, the system input device is assumed.

Identifier may be any fixed-length group item, or an elementary alphabetic, alphanumeric, or external decimal item.

When the FROM option is specified, mnemonic-name must be associated in the SPECIAL-NAMES paragraph with an input/output device—either the system input device or the console typewriter. If the device is the same as that used for READ statements, results are unpredictable.

When the device is the system input device, the following rules apply:

- An input record size of 80 characters is assumed.
- If the identifier is less than 80 characters long, the input data must appear as the first characters within the input record; any characters beyond the length of identifier are truncated.
- If the identifier is longer than 80 characters, succeeding input records are read until the storage area of identifier is filled. If identifier is not an exact multiple of 80 characters, that part of the last input record which will not fit into the identifier is truncated.

When mnemonic-name is associated with the console typewriter, the maximum length of an input message is 255 characters. The following rules apply:

- A system-generated message code is automatically displayed, followed by the literal "AWAITING REPLY."
- Execution is suspended.
- After the message code (the same code as in point 1) followed by a message is keyed in from the console typewriter and recognized by the system, ACCEPT statement execution is resumed; the message is moved to identifier and left-justified, regardless of its PICTURE.

  If the identifier is longer than the incoming message, the rightmost character positions may contain invalid data.

  If the incoming message is longer than identifier, the character positions beyond the length of identifier are truncated.

IBM Extension

When the FROM option is specified, this compiler allows a valid function-name to be specified in place of mnemonic-name. The list of valid function-names is in the System Dependencies chapter.

End of IBM Extension

Format 2—System Information Transfer

This format is used to transfer the system information contained in the specified Special Register (DATE, DAY, or TIME) into identifier.

Identifier may be a fixed-length group item, or an elementary alphanumeric, alphanumeric edited, external decimal, binary, or numeric edited item.

IBM Extension

Identifier may also be an internal decimal item.

End of IBM Extension

The system information is moved from the specified Special Register into identifier, following the rules for the MOVE statement without the CORRESPONDING option.

The special registers DATE, DAY, and TIME implicitly have USAGE DISPLAY.

DATE has the implicit PICTURE 9(6). The sequence of data elements (from left to right) is: 2 digits for year of century, 2 digits for month of year, 2 digits for day of month. Thus July 4, 1981 is expressed as 810704.

DAY has the implicit PICTURE 9(5). The sequence of data elements (from left to right) is: 2 digits for year of century, 3 digits for day of year. Thus July 4, 1981 is expressed as 81186.

TIME has the implicit PICTURE 9(8). The sequence of data elements (from left to right) is: 2 digits for hour of day, 2 digits for minute of hour, 2 digits for second of minute, 2 digits for hundredths of second. Thus 2:41 PM is expressed as 14410000.

Programming Notes

The Format 1 ACCEPT statement is useful for exceptional situations in a program when operator intervention (to supply a given message, code, or exception indicator) is required. The operator must, of course, be supplied with the appropriate messages with which to reply.

The Format 2 ACCEPT statement allows the programmer access to the current date (in two formats) and time of day, as carried by the system. This can be useful in identifying when a particular run of an object program was executed. It can also be used to supply the date in headings, footings, and so forth.

IBM Extension

An example of this use of the Format 2 ACCEPT statement is in the Report Writer sample program.

End of IBM Extension

DISPLAY STATEMENT

The DISPLAY statement transfers low-volume data to an input/output device.

Format

```
           ┌ identifier-1 ┐    ┌ identifier-2 ┐
DISPLAY   <                >   │              │ ...
           │ literal-1    │    │ literal-2    │
           └              ┘    └              ┘

        ┌    mnemonic-name     ┐
   [UPON <  ┌───────────────┐   > ]
        │   │ function-name │  │
        │   └───────────────┘  │
        └                      ┘
```

The DISPLAY statement causes the contents of each operand to be transferred, in the order, left to right, in which the operands are listed, to the input/output device.

Numeric identifiers not described as external decimal are converted automatically to external format, as follows:

- Binary items are converted to external decimal. Negative signed values cause a low-order sign overpunch.

IBM Extension

- Internal Decimal Items are converted to external decimal. Negative signed values cause a low-order sign overpunch.

End of IBM Extension

- No other identifiers require conversion.

For example, if three binary items have values of -34, +34, and 34, they are displayed as 3M, 34, and 34, respectively.

Each numeric literal must be an unsigned integer.

IBM Extension

However, this IBM implementation allows signed noninteger numeric literals.

End of IBM Extension

Each literal may be any figurative constant except ALL literal. When a figurative constant is specified, only a single occurrence of that figurative constant is displayed.

When the UPON option is omitted, the system logical output device is assumed.

When the UPON option is specified, mnemonic-name must be associated in the SPECIAL-NAMES paragraph with an input/output device—the system logical output device, the system punch device, or the console typewriter. A maximum logical record size is assumed for each device, as follows:

System logical output device	120 characters
System logical punch device	72 characters[1]
Console typewriter	100 characters

[1] Characters 73 through 80 are used for PROGRAM-ID name.

When the DISPLAY statement is executed, the data contained in the sending field is transferred to the input/output device. The size of the sending field is the total character count of all listed operands.

If the total character count is less than the device maximum character count, the remaining rightmost characters are padded with spaces.

If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being printed when the end of a record is reached is continued in the next record.

IBM Extension

When the UPON option is specified, this IBM implementation allows a valid function-name to be specified in place of mnemonic-name. The list of valid function-names is in the System Dependencies chapter.

End of IBM Extension

Note: DISPLAY, WRITE AFTER ADVANCING, and a simple WRITE statement all cause the printer to space before printing. A WRITE BEFORE ADVANCING statement, however, causes the printer to space after printing. Thus, specifying the WRITE BEFORE ADVANCING statement and any other of these statements for the same device may cause overprinting.

Programming Notes

The DISPLAY statement is useful for identifying data records that have caused a SIZE ERROR, INVALID KEY, or ON OVERFLOW condition, as well as those which caused a status key return other than 0. Such records can be printed, with an identifying message on some other medium than that used for valid output. Thus all records for one execution that need special handling are separately printed.

ARITHMETIC STATEMENTS

The arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. These operations can be combined symbolically in a formula, using the COMPUTE statement.

COMMON OPTIONS

There are several options common to the arithmetic statements; their description precedes the descriptions of the individual statements.

CORRESPONDING Option

The CORRESPONDING option is valid in the ADD, SUBTRACT, and MOVE statements. The key word CORRESPONDING is equivalent to its abbreviation, CORR.

The CORRESPONDING option allows operations to be performed on elementary items of the same name simply by specifying the group items to which they belong.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion these identifiers are referred to as d1 and d2.

A pair of data items, one from d1 and one from d2, correspond if the following conditions are true:

- In an ADD or SUBTRACT statement, both of the subordinate items are elementary.

- In a MOVE statement, at least one of the subordinate items is elementary.

- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.

- The subordinate items are not identified by the key word FILLER.

- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored.

  However, d1 and d2 themselves may contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, if two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6 INDEXED BY X.
    10  ITEM-B ...
    10  ITEM-C REDEFINES ITEM-B ...
05  ITEM-2
    10  ITEM-A ...
    10  ITEM-B ...
    10  ITEM-C ...
```

  Then, if ADD CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding and are added together; ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. Note that ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level 66, 77, or 88 item, nor is either item described as a FILLER or USAGE IS INDEX item.

IBM Extension

- However, d1 and/or d2 may be subordinate to a FILLER item.

End of IBM Extension

GIVING Option

If the GIVING option is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it may be a numeric edited item.

ROUNDED Option

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

Unless ROUNDED is specified, truncation occurs when the size of the fractional result exceeds the number of places provided for its storage. When ROUNDED is specified, the least significant digit of the resultant identifier has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps and when the number of places in the calculated result exceeds the number of specified integer positions, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

IBM Extension

If the resultant field is floating point, the ROUNDED option has no meaning. However, if at least one of the operands of an arithmetic operation is floating-point data and the resultant field is fixed-point data, rounding always takes place, whether or not ROUNDED is specified.

End of IBM Extension

SIZE ERROR Option

After decimal point alignment, if the value of a result exceeds the largest value that can be contained in the resultant field, a size error condition exists. Division by zero always causes a size error condition.

In the ADD, SUBTRACT, and COMPUTE statements, the size error condition applies only to final results. In the MULTIPLY and DIVIDE statements, the size error condition applies both to final results and intermediate results.

If the ROUNDED option is specified, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR option is specified.

If the SIZE ERROR option is not specified, and a size error condition occurs, the value of the affected resultant identifier is unpredictable. During execution of this operation, values for resultant identifiers for which no size error occurred are not affected by those for which one did occur.

If the SIZE ERROR option is specified and a size error condition occurs, the value of the resultant identifier affected by the size error is not altered—that is, the error results are not placed in the receiving identifier. After completion of the execution of·the arithmetic operation, the imperative statement in the SIZE ERROR option is executed.

For ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, if an individual arithmetic operation causes a size error condition, the SIZE ERROR imperative-statement is not executed until all of the individual additions or subtractions have been completed.

ARITHMETIC STATEMENT OPERANDS

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler supplies any necessary data conversion and decimal point alignment.

Size of Operands

The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superimposition of the operands aligned by decimal point) must not contain more than 18 decimal digits.

For the ADD and SUBTRACT statements, the composite of operands is determined by superimposing all operands in a given statement (except those following the word GIVING).

For the MULTIPLY statement, the composite of operands is determined by superimposing all receiving data items.

For the DIVIDE statement, the composite of operands is determined by superimposing all receiving data items, except the REMAINDER data item.

For the COMPUTE statement, the restriction on composite of operands does not apply.

For example, if the statement ADD A B TO C is executed, and each item is defined as follows in the Data Division:

```
A  PICTURE 9(7)V9(5).

B  PICTURE 9(11)V99.

C  PICTURE 9(12)V9(3).
```

then the composite of operands for this statement consists of 17 decimal digits; it has the following implicit description:

```
Composite-of-Operands PICTURE 9(12)V9(5).
```

IBM Extension

However, this IBM implementation allows all operands in an arithmetic statement to be up to 18 digits in length.

End of IBM Extension

Overlapping Operands

When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

Multiple Results

When an arithmetic statement has multiple results, execution conceptually proceeds as follows:

- The statement performs all arithmetic operations to find the result to be placed in the receiving items, and stores that result in a temporary location.
- A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order that the multiple results are listed.

For example, executing the following statement:

```
ADD A, B, C, TO C, D (C), E.
```

is equivalent to executing the following series of statements:

```
ADD A, B, C GIVING TEMP.

ADD TEMP TO C.

ADD TEMP TO D(C).

ADD TEMP TO E.
```

where TEMP is a compiler-supplied temporary result field. Note that, when the addition operation for D(C) is performed, the subscript C contains the new value of C.

Programming Notes

In all arithmetic statements, it is the user's responsibility to define data with enough digits and decimal places to ensure the desired accuracy in the final result. (See Appendix D on Intermediate Results.)

ADD STATEMENT

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

In Formats 1 and 2, each identifier, except those following the key word GIVING, must name an elementary numeric item.

In Format 2, each identifier following the key word GIVING must name an elementary numeric or numeric edited item.

In Format 3, each identifier must name a group item.

Each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the key word TO are added together, and this sum is added to and stored immediately in identifier-m, and then, if specified, added to and stored immediately in identifier-n, and so forth.

In Format 2, there must be at least two operands preceding the key word GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m, and (if specified) identifier-n, and so forth.

In Format 3, elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2. (See the preceding description of the CORRESPONDING Option.)

Format 1

```
    [ identifier-1 ]   [ identifier-2 ]
ADD <              >   <              >  ...
    | literal-1    |   | literal-2    |
    L              J   L              J

     TO identifier-m [ROUNDED]
        [identifier-n [ROUNDED] ] ...

    [ON SIZE ERROR imperative-statement]
```

Format 2

```
    [ identifier-1 ]  [ identifier-2 ]  [ identifier-3 ]
ADD <              >  <              >  <              >
    | literal-1    |  | literal-2    |  | literal-3    |
    L              J  L              J  L              J

      GIVING identifier-m [ROUNDED]
          [identifier-n [ROUNDED] ] ...

     [ON SIZE ERROR imperative-statement]
```

Format 3

```
    [ CORRESPONDING ]
ADD <               >
    | CORR          |
    L               J

     identifier-1 TO identifier-2 [ROUNDED]

    [ON SIZE ERROR imperative-statement]
```

For the ROUNDED and SIZE ERROR options, and for operand considerations, see the preceding section on Common Options.

If the composite of operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost during execution.

SUBTRACT STATEMENT

The SUBTRACT statement causes one, or the sum of two or more numeric items to be subtracted from one or more numeric items, and the result to be stored.

In Formats 1 and 2, each identifier, except those following the key word GIVING, must name an elementary numeric item.

In Format 2, each identifier following the key word GIVING must name a numeric elementary or numeric edited elementary item.

In Format 3, each identifier must name a group item.

Each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the key word FROM are added together and this sum is subtracted from and stored immediately in identifier-m, and then, if specified, subtracted from and stored immediately in identifier-n, and so forth.

In Format 2, all identifiers or literals preceding the key word FROM are added together and this sum is subtracted from identifier-m or literal-m. The result of the subtraction is stored as the new value of identifier-n, and, if specified, identifier-o, and so forth.

In Format 3, elementary data items within identifier-1 are subtracted from and stored in the corresponding elementary data items within identifier-2. (See the preceding description of the CORRESPONDING option.)

Format 1

```
          [ identifier-1 ]   [ identifier-2 ]
SUBTRACT  <              >   |              |  ...
          | literal-1    ]   | literal-2    ]
          L              J   L              J

   FROM identifier-m [ROUNDED]
       [identifier-n [ROUNDED] ] ...

  [ON SIZE ERROR imperative-statement]
```

Format 2

```
          [ identifier-1 ]   [ identifier-2 ]
SUBTRACT  <              >   |              |  ...
          | literial-1   ]   | literal-2    ]
          L              J   L              J

          [ identifier-m ]
    FROM  <              >
          | literal-m    ]
          L              J

    GIVING identifier-n [ROUNDED]
          [identifier-o [ROUNDED] ] ...

    [ON SIZE ERROR imperative-statement]
```

Format 3

```
          [ CORRESPONDING ]
SUBTRACT  <               >
          | CORR          ]
          L               J

   identifier-1 FROM identifier-2 [ROUNDED]

   [ON SIZE ERROR imperative-statement]
```

For the ROUNDED and SIZE ERROR options, and for operand considerations, see the preceding section on Common Options.

If the composite of operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost during execution.

MULTIPLY STATEMENT

The MULTIPLY statement causes numeric items to be multiplied and sets the values of data items equal to the results.

Each identifier, except those following the key word GIVING, must name an elementary numeric item.

Each identifier following the key word GIVING must name an elementary numeric or numeric edited item.

Each literal must be a numeric literal.

In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. If identifier-3 is specified, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-3, etc.

Format 1

```
            ┌ identifier-1 ┐
MULTIPLY    <              >  BY identifier-2 [ROUNDED]
            │ literal-1    │
            └              ┘

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

In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2; the product is then stored in identifier-3, and, if specified, identifier-4, etc.

Format 2

```
            ┌ identifier-1 ┐      ┌ identifier-2 ┐
MULTIPLY    <              >  BY  <              >
            │ literal-1    │      │ literal-2    │
            └              ┘      └              ┘

   GIVING identifier-3 [ROUNDED]
         [identifier-4 [ROUNDED] ] ...

  [ON SIZE ERROR imperative-statement]
```

For the ROUNDED and SIZE ERROR options, and for operand considerations, see the preceding section on Common Options.

DIVIDE STATEMENT

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

Identifier-1 and identifier-2 must name an elementary numeric item.

Identifier-3 and identifier-4, following the key words GIVING and REMAINDER, must name an elementary numeric or numeric edited item.

Each literal must be a numeric literal.

In Format 1, the value of literal-1 or identifier-1 is divided into the value of identifier-2; the quotient then is placed in identifier-2. If identifier-3 is specified, the value of literal-1 or identifier-1 is divided into identifier-3; the quotient is then placed in identifier-3, etc.

Format 1

```
        ┌ identifier-1 ┐
DIVIDE  <              >
        │ literal-1    │
        └              ┘

   INTO  identifier-2 [ROUNDED]
        [identifier-3 [ROUNDED] ] ...

  [ON SIZE ERROR imperative-statement]
```

In Format 2, the value of identifier-1 or literal-1 is divided into/by the value of identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and (if specified) identifier-4, etc.

Format 2

```
        [ identifier-1 ]   [ INTO ]   [ identifier-2 ]
DIVIDE  <              >   <      >   <              >
        | literal-1    |   | BY   |   | literal-2    |
        L              J   L      J   L              J

   GIVING    identifier-3 [ROUNDED]
            [identifier-4 [ROUNDED] ] ...

  [ON SIZE ERROR imperative-statement]
```

In Format 3, the value of identifier-1 or literal-1 is divided into/by identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

Format 3

```
        [ identifier-1 ]   [ INTO ]   [ identifier-2 ]
DIVIDE  <              >   <      >   <              >
        | literal-1    |   | BY   |   | literal-2    |
        L              J   L      J   L              J

   GIVING identifier-3 [ROUNDED]
  [REMAINDER identifier-4]

  [ON SIZE ERROR imperative-statement]
```

The remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If identifier-3, the quotient, is a numeric edited field, the quotient used to calculate the remainder is an intermediate field which contains the unedited quotient.

For the ROUNDED and SIZE ERROR options, and for operand considerations, see the preceding section on Common Options.

For Format 3, the following additional considerations for the ROUNDED and SIZE ERROR options apply:

- When ROUNDED is specified, the quotient used to calculate the remainder is an intermediate field which contains the quotient truncated rather than rounded.

- When ON SIZE ERROR is specified, and the size error condition occurs on the quotient, no remainder calculation is meaningful. Therefore the contents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

- When ON SIZE ERROR is specified, and the size error occurs on the remainder, the contents of the remainder field (identifier-4) are unchanged.

Note: In both of the last two preceding cases, the user must analyze the results to determine which situation has actually occurred.

COMPUTE STATEMENT

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

Format

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

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

The identifiers that appear to the left of the equal sign (=) must name either elementary numeric items or elementary numeric edited items.

When the COMPUTE statement is executed, the value of the arithmetic-expression is calculated, and this value is then stored as the new value of identifier-1, identifier-2, etc. in turn.

The arithmetic-expression may be any arithmetic-expression as defined in the chapter on Arithmetic Expressions.

An arithmetic-expression consisting of a single identifier or literal allows the user to set identifier-1, etc. equal to the value of that identifier or literal.

For the ROUNDED and SIZE ERROR options, and for operand considerations, see the preceding section on Common Options.

Programming Notes

When arithmetic operations must be combined, the COMPUTE statement is more efficient than the separate arithmetic statements written in series.

Note: When computing a value with a fractional exponent, the floating-point feature is used, and rounding will occur.

DATA MANIPULATION STATEMENTS

Movement and inspection of data are the functions of the following COBOL statements: MOVE, INSPECT, STRING, and UNSTRING.

IBM Extension

The TRANSFORM statement is also a data manipulation statement.

End of IBM Extension

OVERLAPPING OPERANDS

When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

MOVE STATEMENT

The MOVE statement transfers data from one area of storage to one or more other areas.

Format 1

```
       ┌ identifier-1 ┐
MOVE   <              > TO identifier-2 [identifier-3] ] ...
       │ literal      │
       └              ┘
```

Format 2

```
       ┌ CORRESPONDING ┐
MOVE   <               >    identifier-1 TO identifier-2
       │ CORR          │
       └               ┘
```

Identifier-1 and literal are the sending areas. Identifier-2, identifier-3, etc. are the receiving areas.

When Format 1 is specified, all identifiers may be either group or elementary items. The data in the sending area is moved into the first receiving area (identifier-2), then into the second receiving area (identifier-3), etc. See the following descriptions of Elementary and Group Moves.

When Format 2 is specified, both identifiers must be group items. CORR is an abbreviation for, and equivalent to, CORRESPONDING. When CORRESPONDING is specified, selected items in identifier-1 are moved to identifier-2 according to the rules for the CORRESPONDING Option in the chapter on Arithmetic Statements. The results are the same as if each pair of CORRESPONDING identifiers had been referred to in a separate MOVE statement.

An index data item cannot be specified in a MOVE statement.

Any subscripting or indexing associated with the sending item is evaluted only once: immediately before the data is moved to the first receiving field.

Any subscripting or indexing associated with the receiving items is evaluated immediately before the data is moved into the receiving item.

For example, the result of the statement:

```
MOVE A (B) TO B, C(B).
```

is equivalent to

```
MOVE A (B) TO TEMP.

MOVE TEMP TO B.

MOVE TEMP TO C(B).
```

where TEMP has been defined as an intermediate result item. Note that the subscript B has changed in value between the time the first move took place, and the final move to C (B) is executed.

After execution of a MOVE statement, the sending field(s) contains the same data as before execution.

Elementary Moves

An elementary move is one in which both the sending and receiving items are elementary items. Each elementary item belongs to one of the following categories:

- Numeric—includes numeric data items, numeric literals, and the figurative constant ZERO/ZEROS/ZEROES.

- Alphabetic—includes alphabetic data items and the figurative constant SPACE/SPACES.

- Alphanumeric—includes alphanumeric data items, nonnumeric literals, and all figurative constants except ZERO and SPACE.

- Alphanumeric Edited—includes alphanumeric edited data items.

- Numeric Edited—includes numeric edited data items.

Figure 49 shows valid and invalid elementary moves for each category. Valid elementary moves are executed according to the following rules:

- Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in the receiving item.

- For an alphanumeric or alphanumeric edited receiving item:

  - Alignment and any necessary space filling take place as described in Standard Alignment Rules.

  - If the size of the sending item is greater than that of the receiving item, excess characters at the right are truncated after the receiving item is filled.

  - If the sending item has an operational sign, the absolute value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be one less than its actual size.

Sending Item Category	Receiving Item Category					
	Alphabetic	Alphanumeric	Alphanumeric Edited	Numeric Integer	Numeric Noninteger	Numeric Edited
Alphabetic and SPACE	YES	YES	YES	NO	NO	NO
Alphanumeric and Fig. Cons. (1)	YES	YES	YES	YES (3)	YES (3)	YES (3)
Alphanumeric Edited	YES	YES	YES	NO	NO	NO
Numeric Integer (2) and ZERO	NO	YES	YES	YES	YES	YES
Numeric Noninteger (2)	NO	NO	NO	YES	YES	YES
Numeric Edited	NO	YES	YES	NO	NO	NO

(1)–Includes nonnumeric literals and all Figurative Constants but SPACE and ZERO.
(2)–Includes numeric literals
(3)–Nonnumeric literals must consist only of numeric characters and will be treated as a numeric integer field
YES = move is valid
NO = move is invalid

Figure 49. Valid and Invalid Elementary Moves

- For a numeric or numeric edited receiving item:
  - Alignment by decimal point and any necessary 0 filling take place as described in Standard Alignment Rules in the Data Descriptions chapter, except where 0's are replaced due to editing requirements.
  - If the receiving item is signed, the sign of the sending item is placed in the receiving item, with any necessary sign conversion. If the sending item is unsigned, a positive operational sign is generated for the receiving item.
  - If the receiving item is unsigned, the absolute value of the sending item is moved, and no operational sign is generated for the receiving item.
  - When the sending item is alphanumeric, the data is moved as if the sending item were described as an unsigned integer.

Note: If the receiving field is alphanumeric or numeric edited, and the sending field is a scaled integer (that is, has a P as the rightmost character in its PICTURE character-string), the scaling positions are treated as trailing 0's when the MOVE statement is executed. The rules for scaled integers as previously implemented are given in Appendix A.

- For an alphabetic receiving field:
  - Alignment and any necessary space filling take place as described in "Standard Alignment Rules", under "Data Description."

- If the size of the sending item is greater than that of the receiving item, excess characters at the right are truncated after the receiving item is filled.

Group Moves

A group move is one in which one or both of the sending and receiving fields is a group item. A group move is treated exactly as though it were an alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In a group move, the receiving area is filled without consideration for the individual elementary items contained within either the sending area or the receiving area. However, see the OCCURS clause description in the Table Handling chapter.

INSPECT STATEMENT

The INSPECT statement specifies that characters in a data item are to be counted, replaced, or counted and replaced.

Format

See "EXAMINE" page 399

```
INSPECT identifier-1
   [TALLYING < identifier-2

                  [ [ ALL     ]   [ identifier-3 ] ]
           FOR  < < < LEADING >   < literal-1    > >
                  [   CHARACTERS                   ]

              [  < BEFORE > INITIAL < identifier-4 > ] >...>...]
                 < AFTER  >         < literal-2    >
  [REPLACING
[                                                             ]
|      CHARACTERS BY  < identifier-6 >                        |
|                     < literal-4    >                        |
|                                                             |
|              [ < BEFORE > INITIAL < identifier-7 > ]        |
|                < AFTER  >         < literal-5    >          |
<                                                             >  ]
| < < ALL     > < < identifier-5 > BY < identifier-6 >        |
|   < LEADING >   < literal-3    >    < literal-4    >        |
|   < FIRST   >                                               |
|                                                             |
|              [ < BEFORE > INITIAL < identifier-7 > ] >...>...|
|                < AFTER  >         < literal-5    >          |
[                                                             ]
```

Either the TALLYING or the REPLACING option must be specified. Both may be specified; when both are specified, all tallying is performed before any replacement is made.

Identifier-1 is the inspected item; it must be an elementary or group item with USAGE DISPLAY.

All other identifiers, except identifier-2 (the count field), must be elementary alphabetic, alphanumeric, or external decimal items. Each is treated according to its data category, as follows:

- ALPHABETIC OR ALPHANUMERIC ITEM: treated as a character-string.

- ALPHANUMERIC EDITED, NUMERIC EDITED, OR UNSIGNED NUMERIC (EXTERNAL DECIMAL) ITEM: treated as though redefined as alphanumeric and the INSPECT statement refers to the alphanumeric item.

- SIGNED NUMERIC (EXTERNAL DECIMAL) ITEM: treated as though moved to an unsigned external decimal item of the same length, and then treated as though redefined as alphanumeric and the INSPECT statement refers to the alphanumeric item. If the sign is a separate character, the byte containing the sign is not examined and therefore not replaced.

Each literal must be nonnumeric and may be any figurative constant except ALL literal.

Except when the BEFORE/AFTER option is specified, inspection begins at the leftmost character position of the inspected item (identifier-1) and proceeds character-by-character to the rightmost.

The operands of the TALLYING option (literal-1 or identifier-3, and so forth) and/or REPLACING option (literal-3 or identifier-5, etc.) are compared in the left-to-right order they are specified in the INSPECT statement. (In the following discussion, the TALLYING/REPLACING operands are the comperands.) In any one INSPECT statement, no more than 15 comperands may be specified.

The following comparison rules apply:

1. When both the TALLYING and REPLACING options are specified, the INSPECT statement is executed as if an INSPECT TALLYING statement were specified, immediately followed by an INSPECT REPLACING statement.

2. The first comperand is compared with an equal number of leftmost contiguous characters in the inspected item. The comperand matches the inspected characters only if both are equal character-for-character.

3. If no match occurs for the first comperand, the comparison is repeated for each successive comperand until either a match is found or all comperands have been acted upon.

4. If a match is found, tallying or replacing takes place as described in the following TALLYING/REPLACING option descriptions. In the inspected item, the first character following the rightmost matching character is now considered the leftmost character position. The process described in 2 and 3 (above) is then repeated.

5. If no match is found, then in the inspected item the first character following the leftmost inspected character is now considered the leftmost character position. The process described in 2 and 3 (above) is then repeated.

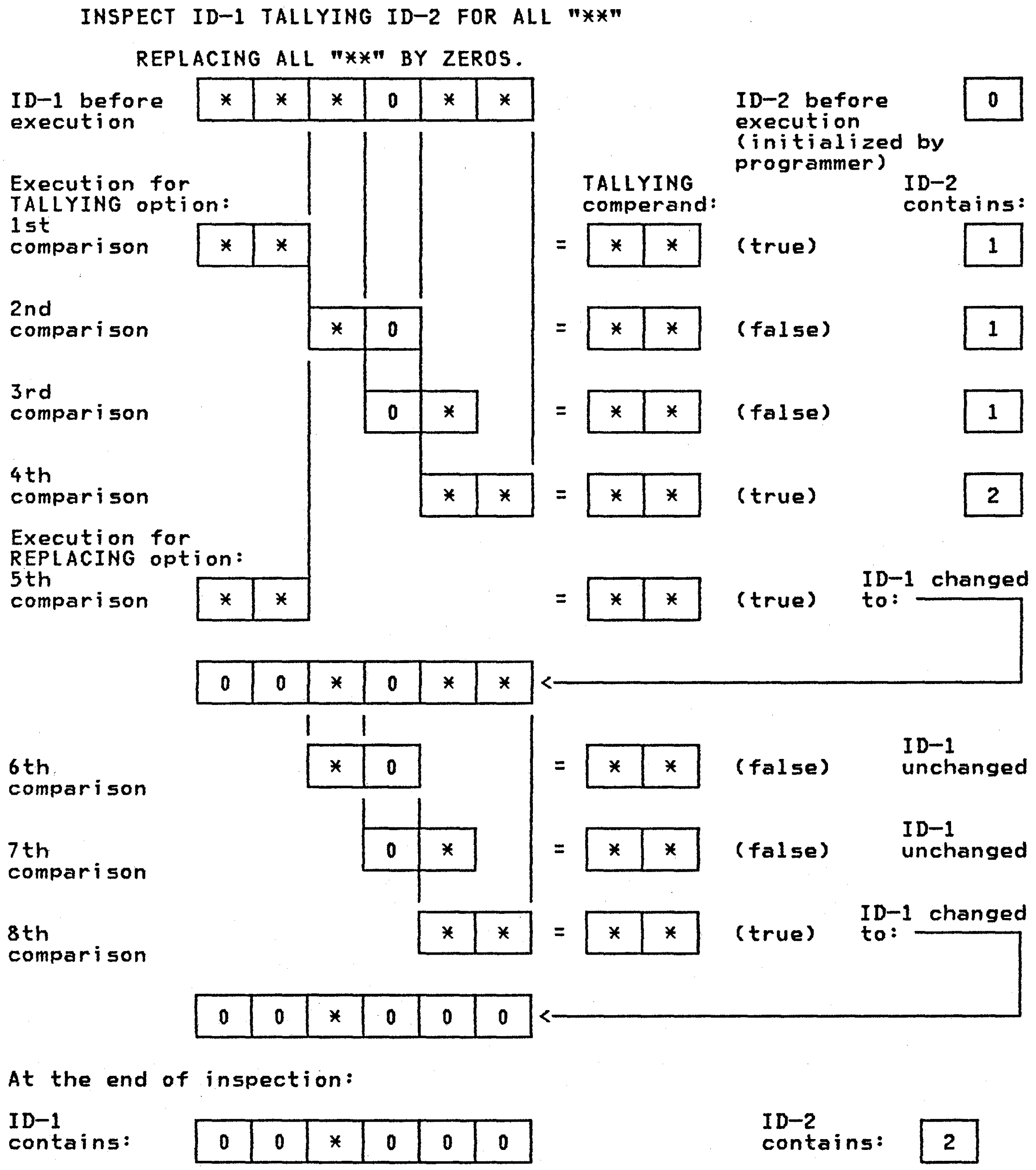
6. If the CHARACTERS option is specified, an implied one-character comperand is used in the process described in 2 and 3 (above). The implied character is considered always to match the inspected character in the inspected item.

7. The actions taken in steps 1 through 6—defined as the comparison cycle—are repeated until the rightmost character

in the inspected item has either been matched or has been considered as the leftmost character position. Inspection then terminates.

When the BEFORE/AFTER option is specified, the preceding rules are modified as described in the following BEFORE/AFTER option description.

Figure 50 illustrates INSPECT statement comparisons.



```
    INSPECT ID-1 TALLYING ID-2 FOR ALL "**"
         REPLACING ALL "**" BY ZEROS.
```

Figure 50. INSPECT Statement Execution Results

TALLYING Option

Identifier-2 is the count field and must be an elementary integer item defined without the symbol P in its PICTURE character-string. It is the programmer's responsibility to initialize identifier-2 before the INSPECT statement is executed.

Identifier-3 or literal-1 is the tallying field. If the tallying field is a figurative constant, it is considered to be a one-character nonnumeric literal.

When the BEFORE/AFTER option is not specified, the following actions take place when the INSPECT TALLYING statement is executed:

- If ALL is specified, the count field is increased by 1 for each nonoverlapping occurrence in the inspected item of this tallying field, beginning at the leftmost character position and continuing to the rightmost.

- If LEADING is specified, the count field is increased by 1 for each contiguous nonoverlapping occurrence of this tallying field in the inspected item, provided that the leftmost such occurrence is at the point at which comparison began in the first comparison cycle for which this tallying field is eligible to participate.

- If CHARACTERS is specified, the count field is increased by 1 for each character (including the space character) in the inspected item. Thus, execution of the INSPECT TALLYING statement increases the value in the count field by the number of characters in the inspected item.

REPLACING Option

Identifier-5 or literal-3 is the subject field.

Identifier-6 or literal-4 is the substitution field.

The subject field and the substitution field must be the same length. The following replacement rules apply:

- If the subject field is a figurative constant, it is considered to be a one-character nonnumeric literal. Each character in the inspected item equivalent to the figurative constant is replaced by the single-character substitution field, which must be one character in length.

- If the substitution field is a figurative constant, the substitution field is considered to be the same length as the subject field. Each nonoverlapping occurrence of the subject field in the inspected item is replaced by the substitution field.

- When the subject and substitution fields are character-strings, each nonoverlapping occurrence of the subject field in the inspected item is replaced by the character-string specified in the substitution field.

- After replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this execution of the INSPECT statement.

- When the BEFORE/AFTER option is not specified, the following actions take place when the INSPECT REPLACING statement is executed:

- If CHARACTERS is specified, the substitution field must be 1-character in length. Each character in the inspected field is replaced by the substitution field, beginning at the leftmost character and continuing to the rightmost.

- If ALL is specified, each nonoverlapping occurrence of the subject field in the inspected item is replaced by the substitution field, beginning at the leftmost character and continuing to the rightmost.
- If LEADING is specified, each contiguous nonoverlapping occurrence of the subject field in the inspected item is replaced by the substitution field, (provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this substitution field is eligible to participate).
- If FIRST is specified, the leftmost occurrence of the subject field in the inspected item is replaced by the substitution field.

BEFORE/AFTER Options

When these options are specified, the preceding rules for counting and replacing are modified.

Identifier-4, identifier-7, literal-2, and literal-5 are delimiters. Counting and/or replacement of the inspected item is bounded by their presence; however, the delimiters themselves are not counted or replaced.

In the TALLYING option, if the delimiter (literal-2) is a figurative constant, it is considered to be 1 character in length.

In the REPLACING option, if the CHARACTERS phrase is specified, the delimiter (literal-5 or identifier-7) must be 1 character in length.

When BEFORE is specified, counting and/or replacement of the inspected item begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, counting and/or replacement continues to the rightmost character.

When AFTER is specified, counting and/or replacement of the inspected item begins with the first character to the right of the delimiter and continues to the rightmost character in the inspected item. If no delimiter is present in the inspected item, no counting or replacement takes place.

INSPECT Statement Examples

The following examples illustrate some uses of the INSPECT statement. Note that in all instances the programmer has initialized the COUNTR field to 0 before the INSPECT statement is executed.

```
INSPECT ID-1 REPLACING CHARACTERS BY ZERO.
```

ID-1 Before	COUNTR After	ID-1 After
1234567	0	0000000
HIJKLMN	0	0000000

```
INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS
REPLACING CHARACTERS BY SPACES.
```

ID-1 Before	COUNTR After	ID-1 After
1234567	7	
HIJKLMN	7	

```
INSPECT ID-1 REPLACING CHARACTERS BY ZEROS
BEFORE INITIAL QUOTE.
```

ID-1 Before	COUNTR After	ID-1 After
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"TWAS BR	0	"TWAS BR

```
INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS
AFTER INITIAL "S"
REPLACING ALL "A" BY "O".
```

ID-1 Before	COUNTR After	ID-1 After
ANSELM	3	ONSELM
SACKET	5	SOCKET
PASSED	3	POSSED

```
INSPECT ID-1 TALLYING COUNTR FOR LEADING "0"
REPLACING FIRST "A" BY "2"
AFTER INITIAL "C".
```

ID-1 Before	COUNTR After	ID-1 After
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

Programming Notes

The INSPECT statement is useful for filling portions or all of a data item with spaces or 0's. It is also useful for counting the number of times a specific character (zero, space, asterisk, and so forth) occurs in a data item. In addition, it can be used to translate characters from one collating sequence to another.

STRING STATEMENT

The STRING statement gives the programmer the ability to put together the partial or complete contents of two or more data items in one single data item.

Format

```
        ┌ identifier-1 ┐   ┌ identifier-2 ┐
STRING  <              >   │              │ ...
        │ literal-1    │   │ literal-2    │
        └              ┘   └              ┘

                           ┌ identifier-3 ┐
              DELIMITED BY < literal-3    >
                           │ SIZE         │
                           └              ┘
     ┌ identifier-4 ┐  ┌ identifier-5 ┐
  [  <              >  │              │  ...
     │ literal-4    │  │ literal-5    │
     └              ┘  └              ┘
                            ┌ identifier-6 ┐
               DELIMITED BY < literal-6    > ] ...
                            │ SIZE         │
                            └              ┘

      INTO identifier-7
     [WITH POINTER identifier-8]
     [ON OVERFLOW imperative-statement]
```

Each literal must be a nonnumeric literal; each may be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered a 1-character nonnumeric literal.

All identifiers, except identifier-8 (the POINTER item), must have USAGE DISPLAY, explicitly or implicitly.

The sending fields are identifier-1, identifier-2, identifier-4, identifier-5, or their corresponding literals.

The receiving field is identifier-7, which must be an elementary alphanumeric item without editing symbols, and without the JUSTIFIED clause in its description.

The delimiters are identifier-3, identifier-6, and their corresponding literals, or the key word SIZE. The delimiters specify the character(s) delimiting the data to be transferred; when SIZE is specified, the complete sending area is transferred.

When the sending field or any of the delimiters are elementary numeric items, they must be described as integers, and their PICTURE character-strings must not contain the symbol P.

The pointer field is identifier-8, which must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus 1. The pointer field must not contain the symbol P in its PICTURE character-string.

STRING Statement Execution

When the STRING statement is executed, data is transferred from the sending fields to the receiving field. The order in which sending fields are processed is the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field according to the rules for alphanumeric to alphanumeric elementary moves, except that no space filling is provided (see the preceding MOVE statement description).

- When DELIMITED BY identifier/literal is specified, the contents of each sending item are transferred character-by-character beginning with the leftmost and continuing until either:

  - A delimiter for this sending field is reached (the delimiter itself is not transferred), or

  - The rightmost character of this sending field has been transferred.

- When DELIMITED BY SIZE is specified, each entire sending field is transferred to the receiving field.

- When the receiving field is filled, or when all of the sending fields have been processed, the operation is ended.

- When the POINTER option is specified, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. The user must set the explicit pointer's initial value, which must not be less than 1 and not more than the character count of the receiving field. (Note that the pointer field must be defined as large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.)

- When the POINTER option is not specified, no pointer is available to the user. However, a conceptual implicit pointer with an initial value of 1 is used by the system.

- Conceptually, when the STRING statement is executed, the initial pointer value (explicit or implicit) is the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned character-by-character from left to right. After each character is positioned, the explicit or implicit pointer is incremented by 1. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates one character beyond the last character transferred into the receiving field.

- If, at any time during or after initiation of STRING statement execution, the pointer value (explicit or implicit) is less than 1 or exceeds a value equal to the length of the receiving field, no more data is transferred into the receiving field, and, if specified, the ON OVERFLOW imperative-statement is executed.

- If the ON OVERFLOW option is not specified, then, when the preceding conditions occur, control passes to the next executable statement.

After STRING statement execution is completed, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before this execution of the STRING statement.

Figure 51 illustrates the rules of execution for the STRING statement.

When the following STRING statement is executed:

```
STRING ID-1 ID-2 DELIMITED BY ID-3
       ID-4 ID-5 DELIMITED BY SIZE
  INTO ID-7 WITH POINTER ID-8.
```

The following results are obtained:

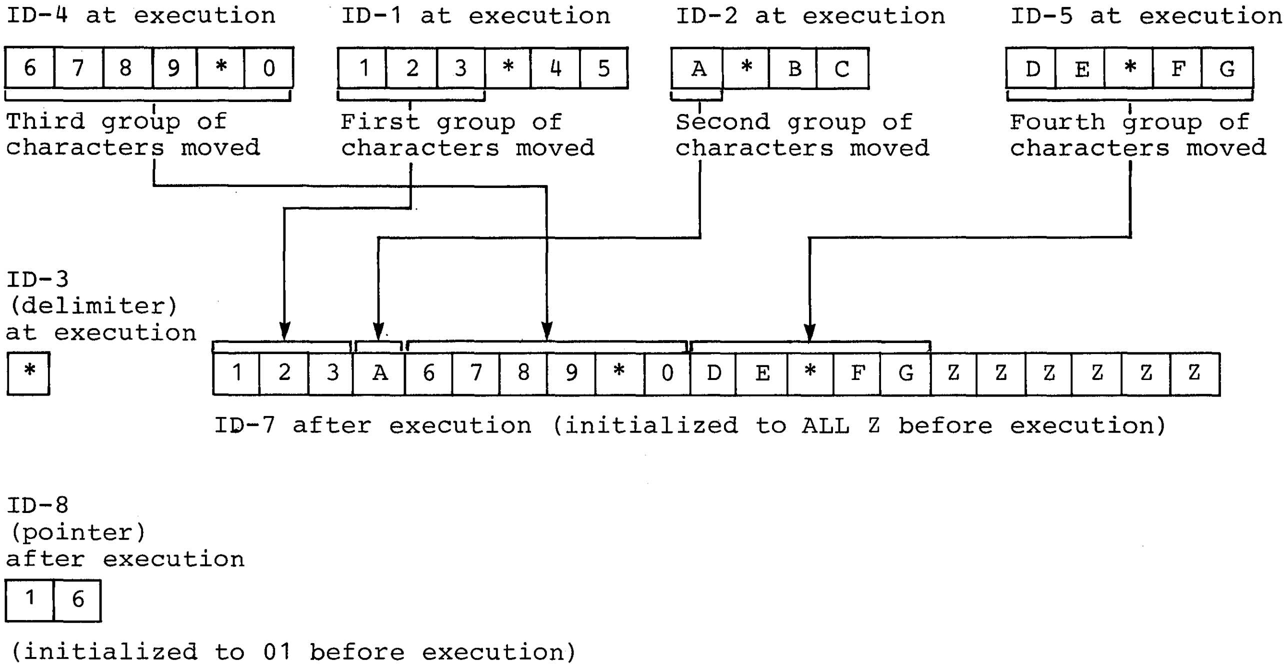

ID-4 at execution
6 7 8 9 * 0
Third group of characters moved
ID-1 at execution
1 2 3 * 4 5
First group of characters moved
ID-2 at execution
A * B C
Second group of characters moved
ID-5 at execution
D E * F G
Fourth group of characters moved
ID-3 (delimiter) at execution
*
1 2 3 A 6 7 8 9 * 0 D E * F G Z Z Z Z Z Z
ID-7 after execution (initialized to ALL Z before execution)
ID-8 (pointer) after execution
1 6
(initialized to 01 before execution)

Figure 51. STRING Statement Execution Results

STRING Statement Example

The following illustrates some of the considerations that apply to the STRING statement.

In the Data Division, the programmer has defined the following fields:

```
77    RPT-LINE    PICTURE  X(120).
77    LINE-POS    PICTURE  99.
77    LINE-NO     PICTURE  9(5) VALUE 1.
77    DEC-POINT   PICTURE  X VALUE ".".
```

In the File Section, the following input record is defined:

```
01    RCD-01.
      05    CUST-INFO.
            10 CUST-NAME     PICTURE X(15).
            10 CUST-ADDR     PICTURE X(35).
      05    BILL-INFO.
            10 INV-NO        PICTURE X(6).
            10 INV-AMT       PICTURE $$,$$$.99.
            10 AMT-PAID      PICTURE $$,$$$.99.
            10 DATE-PAID     PICTURE X(8).
            10 BAL-DUE       PICTURE $$,$$$.99.
            10 DATE-DUE      PICTURE X(8).
```

The programmer wants to construct an output line consisting of portions of the information from RCD-01. The line is to consist of a line number, customer name and address, invoice number, next billing date, and balance due, truncated to the dollar figure shown.

The record as read in contains the following information:

```
J.B.bSMITHbbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

In the Procedure Division, the programmer initializes RPT-LINE to SPACES, and sets LINE-POS (which is to be used as the POINTER field) to 4. This STRING statement is then issued:

```
STRING LINE-NO SPACE CUST-INFO INV-NO SPACE DATE-DUE
          SPACE DELIMITED BY SIZE
    BAL-DUE DELIMITED BY DEC-POINT
    INTO RPT-LINE WITH POINTER LINE-POS.
```

When the statement is executed, the following actions take place:

- The field LINE-NO is moved into positions 4 through 8 of RPT-LINE.

- A space is moved into position 9.

- The group item CUST-INFO is moved into positions 10 through 59.

- INV-NO is moved into positions 60 through 65.

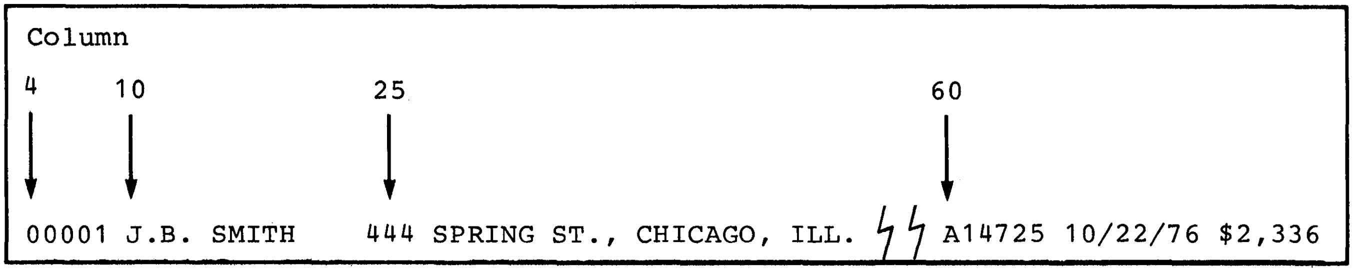
- A space is moved into position 66.

- DATE-DUE is moved into positions 67 through 74.

- A space is moved into position 75.

- The portion of BAL-DUE that precedes the decimal point is moved into positions 76 through 81.

At the end of execution of the STRING statement, RPT-LINE appears as shown in Figure 52.



```
Column
4    10               25                                   60
00001 J.B. SMITH     444 SPRING ST., CHICAGO, ILL.         A14725 10/22/76 $2,336
```

Figure 52. STRING Statement Example—Output

Programming Notes

One STRING statement can be written instead of a series of MOVE statements. In communication applications, this is especially useful in assembling messages to be transmitted.

UNSTRING STATEMENT

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

```
UNSTRING identifier-1

                                 [ identifier-2 ]
   [DELIMITED BY [ALL]          <               >
                                 [ literal-1    ]

                            [ identifier-3 ]
      [OR [ALL]            <               >  ] ... ]
                            [ 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]
```

Each literal must be a nonnumeric literal; each may be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered to be a 1-character nonnumeric literal.

Sending Field

Identifier-1 is the sending field. It must be an alphanumeric data item. Data is transferred from this field to the receiving fields.

DELIMITED BY OPTION: This option specifies delimiters within the data that control the data transfer.

The delimiters are identifier-2, identifier-3, or their corresponding literals. Each identifier or literal specified represents one delimiter. No more than 15 delimiters may be specified. Each must be an alphanumeric data item.

If a delimiter contains two or more characters, it is recognized as a delimiter only if the delimiter characters are contiguous, and in the sequence specified, in the sending field.

When two or more delimiters are specified, an OR condition exists, and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the specified sequence. For example, if DELIMITED BY "AB" OR "BC" is specified, then an occurrence of either AB or BC in the sending field is considered a delimiter; an occurrence of ABC is considered an occurrence of AB.

When DELIMITED BY ALL is not specified, and two or more contiguous occurrences of any delimiter are encountered, the current data receiving field is space filled or zero filled according to the description of the data receiving field.

When DELIMITED BY ALL is specified, one or more contiguous occurrences of any delimiter are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field (if specified). The delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved into the current delimiter receiving field according to the rules of the MOVE statement.

Note: Rules for the DELIMITED BY ALL option as previously implemented are given in Appendix A.

Unless the DELIMITED BY option is specified, the DELIMITER IN and COUNT IN options must not be specified.

Data Receiving Fields

Identifier-4, identifier-7, etc., are the data receiving fields, and must have USAGE DISPLAY. These fields can be defined as:

- Alphabetic (without the symbol B in the PICTURE string)
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string)

These fields must not be defined as alphanumeric edited or numeric edited items. Data is transferred to these fields from the sending field.

DELIMITER IN OPTION: The delimiter receiving fields are identifier-5, identifier-8, etc. These identifiers must be alphanumeric.

COUNT IN OPTION: The data-count fields for each data transfer are identifier-6, identifier-9, and so forth. Each field holds the count of examined characters in the sending field, terminated by the delimiters or the end of the sending field, for the move to this receiving field; the delimiters are not included in this count.

first position = 1
not zero

POINTER OPTION: The pointer field is identifier-10; it contains a value that indicates a relative position in the sending field. When this option is specified, the user must initialize this field before execution of the UNSTRING statement is begun.

TALLYING OPTION: The field-count field is identifier-11; it is increased by the number of data receiving fields acted upon in this execution of the UNSTRING statement. When this option is specified, the user must initialize this field before execution of the UNSTRING statement is begun.

The data-count fields, the pointer field, and the field-count field must each be an integer item without the symbol P in the PICTURE character-string.

UNSTRING Statement Execution

When the UNSTRING statement is initiated, the current data receiving field is identifier-4. Data is transferred from the sending field to the current data receiving field according to the following rules:

- If the POINTER option is not specified, the sending field character-string is examined beginning with the leftmost character. If the POINTER option is specified, the field is examined beginning at the relative character position specified by the value in the pointer field.

- If the DELIMITED BY option is specified, the examination proceeds left to right character-by-character until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field.

- If the DELIMITED BY option is not specified, the number of characters examined is equal to the size of the current data receiving field, which depends on its data category:

  - If the receiving field is alphanumeric or alphabetic, the number of characters examined is equal to the number of characters in the current receiving field.

  - If the receiving field is numeric, the number of characters examined is equal to the number of characters in the integer portion of the current receiving field.

  - If the receiving field is described with the SIGN IS SEPARATE clause, the characters examined are one fewer than the size of the current receiving field.

  - If the receiving field is described as a variable-length data item, the number of characters examined is determined by the current size of the current receiving field.

- The examined characters (excluding any delimiter characters) are treated as an alphanumeric elementary item, and are moved into the current data receiving field according to the rules for the MOVE statement (see the preceding MOVE statement description).

- If the DELIMITER IN option is specified, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.

- If the COUNT IN option is specified, a value equal to the number of examined characters (excluding any delimiters) is moved into the data count field, according to the rules for an elementary move.

- If the DELIMITED BY option is specified, the sending field is further examined, beginning with the first character to the right of the delimiter.

- If the DELIMITED BY option is not specified, the sending field is further examined, beginning with the first character to the right of the last character examined.

- After data is transferred to the first data receiving field (identifier-4) the current data receiving field becomes identifier-7. For each succeeding current data receiving field, the preceding procedure is repeated—either until all of the characters in the sending field have been transferred, or until there are no more unfilled data receiving fields.

- When the POINTER option is specified, the contents of the pointer field act as if increased by 1 for each examined character in the sending field. When this execution of the UNSTRING statement is completed, the pointer field contains a value equal to its initial value plus the number of characters examined in the sending field.

- When the TALLYING option is specified, then, when this execution of the UNSTRING statement is completed, the field-count field contains a value equal to the initial value plus the number of data receiving areas acted upon; this count includes any null fields.

- When an overflow condition exists, the execution of the UNSTRING statement is ended. If the ON OVERFLOW option has been specified, that imperative-statement is executed. If the ON OVERFLOW option has not been specified, control passes to the next executable statement. An overflow condition exists when:

  - An UNSTRING statement is initiated, and the value in the pointer field is less than 1 or greater than the length of the sending field.

  - If, during UNSTRING statement execution, all data receiving fields have been acted upon, and the sending field still contains unexamined characters.

Figure 53 illustrates the rules of execution for the UNSTRING statement.

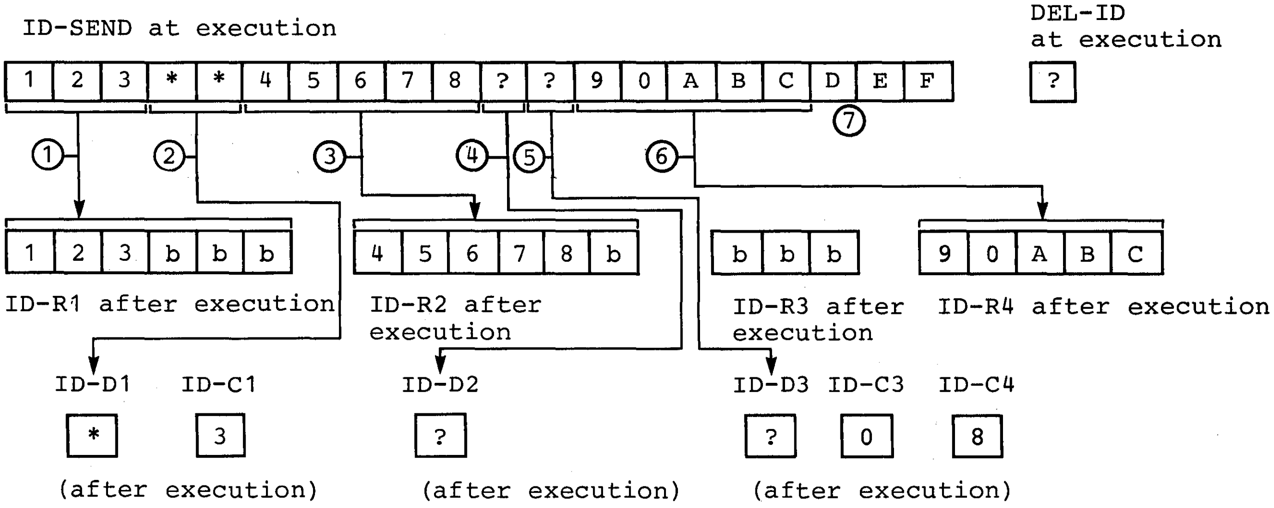
Note: If any of the UNSTRING statement identifiers are subscripted or indexed, the subscripts and indexes are evaluated as follows;

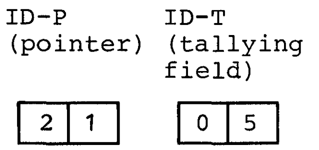
- Any subscripting or indexing associated with the sending field, the pointer field, or the field-count field is evaluated only once—immediately before any data is transferred.

- Any subscripting or indexing associated with the delimiters, the data and delimiter receiving fields, or the data-count fields, is evaluated immediately before the transfer of data into the affected data item.

The following UNSTRING Statement has the execution results shown:

```
UNSTRING ID-SEND DELIMITED BY DEL-ID OR ALL "*"
    INTO ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1   (All the data
         ID-R2 DELIMITER IN ID-D2                  receiving fields
         ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3   are defined as
         ID-R4 COUNT IN ID-C4                      alphanumeric)
    WITH POINTER ID-P
    TALLYING IN ID-T
    ON OVERFLOW GO TO OFLOW-EXIT.
```





(after execution --
both initialized to
01 before execution)

The order of execution is:

① 3 characters are placed in ID-R1.

② Because ALL * is specified, one * is placed in ID-D1.

③ 5 characters are placed in ID-R2.

④ A ? is placed in ID-D2. The current receiving field is now ID-R3.

⑤ A ? is placed in ID-D3; ID-R3 is filled with spaces; no characters are transferred, so 0 is placed in ID-C3

⑥ No delimiter is encountered before 5 characters fill ID-R4; 8 is placed in ID-C4, representing the number of characters examined since the last delimiter.

⑦

ID-P is updated to 21, the total length of the sending field + 1; ID-T is updated to 5, the number of fields acted upon. Since there are no unexamined characters in the ID-SEND, the OVERFLOW EXIT is not taken.

Figure 53. Results of UNSTRING Statement Execution

UNSTRING Statement Example

The following example illustrates some of the considerations that apply to the UNSTRING statement.

In the Data Division, the programmer has defined the following input record to be acted upon by the UNSTRING statement:

```
01    INV-RCD.
    05  CONTROL-CHARS              PIC XX.
    05  ITEM-INDENT                PIC X(20).
    05  FILLER                     PIC X.
    05  INV-CODE                   PIC X(10).
    05  FILLER                     PIC X.
    05  NO-UNITS                   PIC 9(6).
    05  FILLER                     PIC X.
    05  PRICE-PER-M                PIC 99999.
    05  FILLER                     PIC X.
    05  RTL-AMT                    PIC 9(6).99.
```

The next two records are defined as receiving fields for the UNSTRING statement. DISPLAY-REC is to be used for printed output. WORK-REC is to be used for further internal processing.

```
01   DISPLAY-REC.
    05  INV-NO                     PIC X(6).
    05  FILLER                     PIC X VALUE SPACE.
    05  ITEM-NAME                  PIC X(20).
    05  FILLER                     PIC X VALUE SPACE.
    05  DISPLAY-DOLS               PIC 9(6).

01    WORK-REC.
    05  M-UNITS                        PIC 9(6).
    05  FIELD-A                        PIC 9(6).
    05  WK-PRICE REDEFINES FIELD-A     PIC 9999V99.
    05  INV-CLASS                      PIC X(3).
```

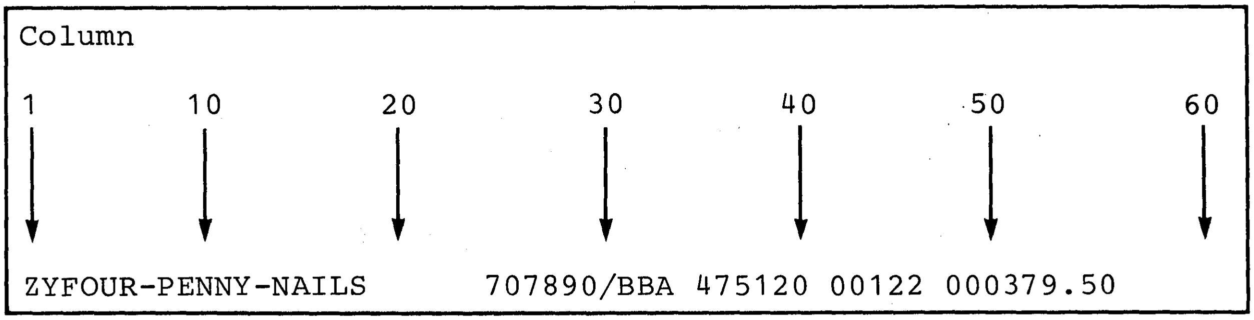
The programmer has also defined the following fields for use as control fields in the UNSTRING statement:

```
77    DBY-1        PIC X.
77    CTR-1        PIC 99.
77    CTR-2        PIC 99.
77    CTR-3        PIC 99.
77    CTR-4        PIC 99.
77    DLTR-1        PIC X.
77    DLTR-2        PIC X.
77    CHAR-CT         PIC 99.
77    FLDS-FILLED     PIC 99.
```

In the Procedure Division, the programmer writes the following UNSTRING statement to move subfields of INV-RCD to the subfields of DISPLAY-REC and WORK-REC:

```
UNSTRING INV-RCD DELIMITED BY ALL SPACES OR '/' OR
    DBY-1 INTO ITEM-NAME COUNT IN CTR-1
        INV-NO DELIMITER IN DLTR-1 COUNT IN CTR-2
        INV-CLASS
        M-UNITS COUNT IN CTR-3
        FIELD-A
        DISPLAY-DOLS DELIMITER IN DLTR-2 COUNT IN CTR-4
        WITH POINTER CHAR-CT
        TALLYING IN FLDS-FILLED
        ON OVERFLOW GO TO UNSTRING-COMPLETE.
```

Before the UNSTRING statement is issued, the programmer places the value 3 in the CHAR-CT (the POINTER item). Because the programmer doesn't want to work with the two control characters in INV-RCD, the value 3 is placed in the CHAR-CT (the POINTER item) before the UNSTRING statement is issued. In DBY-1, a period (.) is placed for use as a delimiter, and, in FLGS-FILLED (the TALLYING item), the value 0 (zero) is placed. The following data is then read into INV-RCD as shown in Figure 54.



```
ZYFOUR-PENNY-NAILS       707890/BBA 475120 00122 000379.50
```

Figure 54. UNSTRING Statement Example—Input Data

When the UNSTRING statement is executed, the following actions take place:

1. Positions 3 through 18 (FOUR-PENNY-NAILS) of INV-RCD are placed in ITEM-NAME, left-justified within the area, and the unused character positions are padded with spaces. The value 16 is placed in CTR-1.

2. Since ALL SPACES is specified as a delimiter, the 5 contiguous SPACE characters are considered to be one occurrence of the delimiter.

3. Positions 24 through 29 (707890) are placed in INV-NO. The delimiter character / is placed in DLTR-1, and the value 6 is placed in CTR-2.

4. Positions 31 through 33 are placed in INV-CLASS. The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is merely bypassed.

5. Positions 35 through 40 (475120) are examined, and are placed in M-UNITS. The delimiter is a SPACE, but because no receiving field has been defined as a receiving area for delimiters, the SPACE is bypassed. The value 6 is placed in CTR-3.

6. Positions 42 through 46 (00122) are placed in FIELD-A, and right-justified within the area. The high-order digit position is filled with a 0 (zero). The delimiter is a SPACE, but, because no field has been defined as a receiving area for delimiters, the SPACE is bypassed.

7. Positions 48 through 53 (000379) are placed in DISPLAY-DOLS. The period (.) delimiter character is placed in DLTR-2, and the value 6 is placed in CTR-4.

8. Because all receiving fields have been acted upon and two characters of data in INV-RCD have not been examined, the ON OVERFLOW exit is taken, and execution of the UNSTRING statement is completed.

At the end of execution of the UNSTRING statement, DISPLAY-REC contains the following data:

```
707890 FOUR-PENNY-NAILS     000379
```

WORK-REC contains the following data:

```
475120000122BBA
```

CHAR-CT (the POINTER field) contains the value 55, and FLDS-FILLED (the TALLYING field) contains the value 6.

Programming Notes

One UNSTRING statement can be written instead of a series of MOVE statements. In communications applications, this is especially useful in breaking apart incoming messages for internal processing.

If the sending field is also the receiving field, part of the sending field may be overlaid by spaces (depending on the definition of the receiving field). The current delimited receiving field is filled with spaces.

IBM Extension

TRANSFORM STATEMENT

The TRANSFORM statement is used to alter characters according to a transformation rule.

Format

```
TRANSFORM identifier-3 CHARACTERS

         [ figurative-constant-1 ]
    FROM < nonnumeric-literal-1  >
         [ identifier-1          ]

         [ figurative-constant-2 ]
      TO < nonnumeric-literal-2  >
         [ identifier-2          ]
```

Identifier-3 represents an elementary alphabetic, alphanumeric, or numeric edited item, or a group item. Identifier-3 may be an external decimal, provided that figurative-constant-1 is space or spaces and figurative-constant-2 is valid. However, whenever identifier-3 is an external decimal, a warning message will be issued.

The combination of the FROM and TO options determines the transformation rule.

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

1. Nonnumeric literals require enclosing quotation marks.

2. Identifier-1 and identifier-2 must be elementary alphabetic, or alphanumeric items, or fixed length group items not over 255 characters in length.

3. A character may not be repeated in nonnumeric-literal-1 or in the area defined by identifier-1. If a character is repeated, the results are unpredictable.

4. The allowable figurative constants are: ZERO, ZEROES, ZEROS, SPACE, SPACES, QUOTE, QUOTES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, and LOW-VALUES.

When either identifier-1 or identifier-2 appears as an operand of the specific transformation, the user can change the transformation rule at object time.

Examples of data transformation are given in Figure 55; combinations of the FROM and TO options are shown in Figure 56.

If any of the operands of a TRANSFORM statement share a part of their storage (that is, if the operands overlap), the result of the execution of such a statement is unpredictable.

Note: No matter which collating sequence is used by the program, the TRANSFORM statement uses the EBCDIC collating sequence for comparisons.

Programming Notes

The TRANSFORM statement can be used to translate from one collating sequence to another. It can also be used to change a space-filled field to a zero-filled field, and vice versa.

see quote-mark explanation on page 406

Identifier-3 (Before)	FROM	TO	Identifier-3 (After)
1b7bbABC	SPACE	QUOTE	1'7''ABC
1b7bbABC	'17CB'	'QRST'	QbRbbATS
1b7bbABC	b17ABC	CBA71b	BCACC71b
1234WXy89	98YXW4321	ABCDEFGHI	IHGFEDCBA

Figure 55. Examples of Data Transformation

Operands	Transformation Rule
FROM figurative-constant-1 TO figurative-constant-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character figurative-constant-2.
FROM figurative-constant-1 TO nonnumeric-literal-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character nonnumeric-literal-2.
FROM figurative-constant-1 TO identifier-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the identifier-2.
FROM nonnumeric-literal-1 TO figurative-constant-2	All characters in the data item represented by identifier-3 that are equal to any character in nonnumeric-literal-1 are replaced by the single character figurative-constant-2.
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 nonnumeric-literals are equal in length, any character in the data item represented by identifier-3 equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of nonnumeric-literal-2. If the length of nonnumeric-literal-2 is one, all characters in the data item represented by identifier-3 that are equal to any character appearing in nonnumeric-literal-1 are replaced by the single character given in nonnumeric-literal-2.

Figure 56 (Part 1 of 2). TRANSFORM Statement—FROM and TO Options

Operands	Transformation Rule
FROM nonnumeric-literal-1 TO identifier-2	Nonnumeric-literal-1 and the data item represented by identifier-2 must be equal in length or identifier-2 must represent a single character item. If nonnumeric-literal-1 and identifier-2 are equal in length, any character represented by identifier-3 equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of the item represented by identifier-2. If the length of the data item represented by identifier-2 is one, all characters represented by identifier-3 that are equal to any character appearing in nonnumeric-literal-1 are replaced by the single character represented by identifier-2.
FROM identifier-1 TO figurative-constant-2	All characters represented by identifier-3 that are equal to any character in the data item represented by identifier-1 are replaced by the single character figurative-constant-2.
FROM identifier-1 TO nonnumeric-literal-2	The data item represented by identifier-1 and nonnumeric-literal-2 must be of equal length or nonnumeric-literal-2 must be one character. If identifier-1 and nonnumeric-literal-2 are equal in length, any character in identifier-3 equal to a character in identifier-1 is replaced by the character in the corresponding position of nonnumeric-literal-2. If the length of nonnumeric-literal-2 is one, all characters represented by identifier-3 that are equal to any character represented by identifier-1 are replaced by the single character given in nonnumeric-literal-2.
FROM identifier-1 TO identifier-2	Any character in the data item represented by identifier-3 equal to a character in the data item represented by identifier-1 is replaced by the character in the corresponding position of the data item represented by identifier-2. Identifier-1 and identifier-2 can be one or more characters, but must be equal in length.

Figure 56 (Part 2 of 2). TRANSFORM Statement—FROM and TO Options

End of IBM Extension

PROCEDURE BRANCHING STATEMENTS

Statements, sentences, and paragraphs in the Procedure Division are ordinarily executed sequentially. The procedure branching statements (GO TO, ALTER, PERFORM, STOP, and EXIT) allow alterations in the sequence.

GO TO STATEMENT

The GO TO statement transfers control from one part of the Procedure Division to another.

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.
```

Each specified procedure-name must name a paragraph or a section in the Procedure Division.

Identifier must name an elementary integer item. Its PICTURE must be of four digits or less.

Format 1—Unconditional GO TO

The GO TO statement causes control to be transferred to the first statement in the paragraph or section named in procedure-name-1, unless the GO TO statement has been modified by an ALTER statement. (See the following description of the ALTER statement.)

A Format 1 GO TO statement, when it appears in a sequence of imperative statements, must be the last statement in the sequence.

When a paragraph is referred to by an ALTER statement, the paragraph may consist only of a paragraph-name followed by a Format 1 or Format 3 GO TO statement.

Format 2—Conditional GO TO

Control is transferred to one of a series of procedures, depending on the value of identifier. When identifier has a value of 1, control is transferred to the first statement in the procedure named by procedure-name-1; if it has a value of 2, control is transferred to the first statement in the procedure named by procedure-name-2; and so forth.

If the value of the identifier is other than a value within the range 1 through n (where n is the number of procedure-names specified in this GO TO statement), no control transfer occurs; instead, control passes to the next statement in the normal sequence of execution.

The number of procedure-names must not exceed 2031.

Format 3—Altered GO TO

An ALTER statement referring to this GO TO statement must have been executed before this GO TO statement is executed. When this GO TO statement is executed, control is transferred to the first statement of the paragraph named in the ALTER statement.

A Format 3 GO TO statement can appear only in a single-statement paragraph.

When a paragraph is referred to by an ALTER statement, the paragraph may consist only of the paragraph-name followed by a Format 1 or Format 3 GO TO statement.

ALTER STATEMENT

The ALTER statement changes the transfer point specified in a GO TO statement.

Format

```
ALTER procedure-name-1
          TO [PROCEED TO] procedure-name-2
      [procedure-name-3
          TO [PROCEED TO] procedure-name-4] ...
```

Procedure-name-1, procedure-name-3, and so forth, must each name a Procedure Division paragraph that contains only one sentence: a GO TO statement without the DEPENDING ON option.

Procedure-name-2, procedure-name-4, etc., must each name a Procedure Division section or paragraph.

ALTER statement execution modifies the GO TO statement in the paragraph named by procedure-name-1, procedure-name-3, and so forth. Subsequent executions of the modified GO TO statement(s) cause control to be transferred to procedure-name-2, and (if specified) procedure-name-4, etc. For example:

```
PARAGRAPH-1.
  GO TO BYPASS-PARAGRAPH.
PARAGRAPH-1A.
    .
    .
BYPASS-PARAGRAPH.
    .
    .
  ALTER PARAGRAPH-1 TO PROCEED TO PARAGRAPH-2.
    .
    .
PARAGRAPH-2.
    .
    .
```

Before the ALTER statement is executed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After execution of the ALTER statement, however, the next time control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Programming Notes

The ALTER statement acts as a program switch, allowing, for example, one sequence of execution during initialization and another sequence during the bulk of file processing. Because altered GO TO statements are difficult to debug, a tally should be kept to indicate when an ALTER statement has been executed.

Segmentation Information

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in an overlayable fixed segment.

Modified GO TO statements in independent segments may, in some cases, be returned to their initial states.

(For further information, see the chapter on Segmentation.)

PERFORM STATEMENT

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next executable statement after execution of the specified procedure(s) is completed.

Format 1

```
                                 [ [ THROUGH ]                      ]
PERFORM procedure-name-1         | <         >   procedure-name-2   |
                                 | | THRU    ]                      ]
                                 L L         J                      J
```

Format 2

```
                                 [ [ THROUGH ]                      ]
PERFORM procedure-name-1         | <         >   procedure-name-2   |
                                 | | THRU    ]                      ]
                                 L L         J                      J

   [ identifier-1 ]
   <              > TIMES
   | integer-1    ]
   L              J
```

Format 3

```
                                 [ [ THROUGH ]                      ]
PERFORM procedure-name-1         | <         >   procedure-name-2   |
                                 | | THRU    ]                      ]
                                 L L         J                      J
   UNTIL condition-1
```

Format 4

```
                                  ┌ ┌ THROUGH ┐                    ┐
PERFORM procedure-name-1 [        │ <          >  procedure-name-2 │
                                  │ │ THRU    ┘                    │
                                  └ └                              ┘

       VARYING  ┌ identifier-1 ┐      ┌ literal-2      ┐
                <              > FROM < identifier-2   >
                │ index-name-1 ┘      │ index-name-2   ┘
                └                     └

            BY ┌ literal-3     ┐
               <               > UNTIL condition-1]
               │ identifier-3  ┘
               └

      ┌ AFTER   ┌ identifier-4 ┐      ┌ literal-5      ┐
      │         <              > FROM < identifier-5   >
      └         │ index-name-4 ┘      │ index-name-5   ┘
                └                     └

            BY ┌ literal-6     ┐
               <               > UNTIL condition-2]
               │ identifier-6  ┘
               └

      ┌ AFTER   ┌ identifier-7 ┐      ┌ literal-8      ┐
      │         <              > FROM < identifier-8   >
      └         │ index-name-7 ┘      │ index-name-8   ┘
                └                     └

            BY ┌ literal-9     ┐
               <               > UNTIL condition-3]
               │ identifier-9  ┘
               └
```

Each procedure-name must name a section or paragraph in the Procedure Division.

When both procedure-name-1 and procedure-name-2 are specified, if either is a procedure-name in a Declarative procedure, then both must be procedure-names in the same Declarative procedure.

Each identifier must name a numeric elementary item.

Each literal must be a numeric literal.

Whenever a PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. Control is always returned to the statement following the PERFORM statement. The point from which this control is returned is determined as follows:

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

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

- If procedure-name-2 is specified and it is a paragraph name, the return is made after the execution of the last statement of that paragraph.

- If procedure-name-2 is specified and it is a section name, the return is made after the execution of the last sentence of the last paragraph in the section.

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is executed beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2. A PERFORM statement must not cause the flow of control to pass through itself; that is, PERFORM A THRU A is not valid.

GO TO and PERFORM statements may be specified within the performed procedure. If there are two or more logical paths to the return point, then procedure-name-2 may name a paragraph that consists only of an EXIT statement; all the paths to the return point must then lead to this paragraph.

When a performed procedure includes another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose execution point begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM statement. In addition, two or more such active PERFORM statements must not have a common exit.

IBM Extension

However, this IBM implementation allows embedded PERFORM and GO TO statements to have a common exit point with the original PERFORM statement. The common exit point must be the name of a paragraph consisting solely of an EXIT statement.

End of IBM Extension

Figure 57 illustrates valid sequences of execution for PERFORM statements.

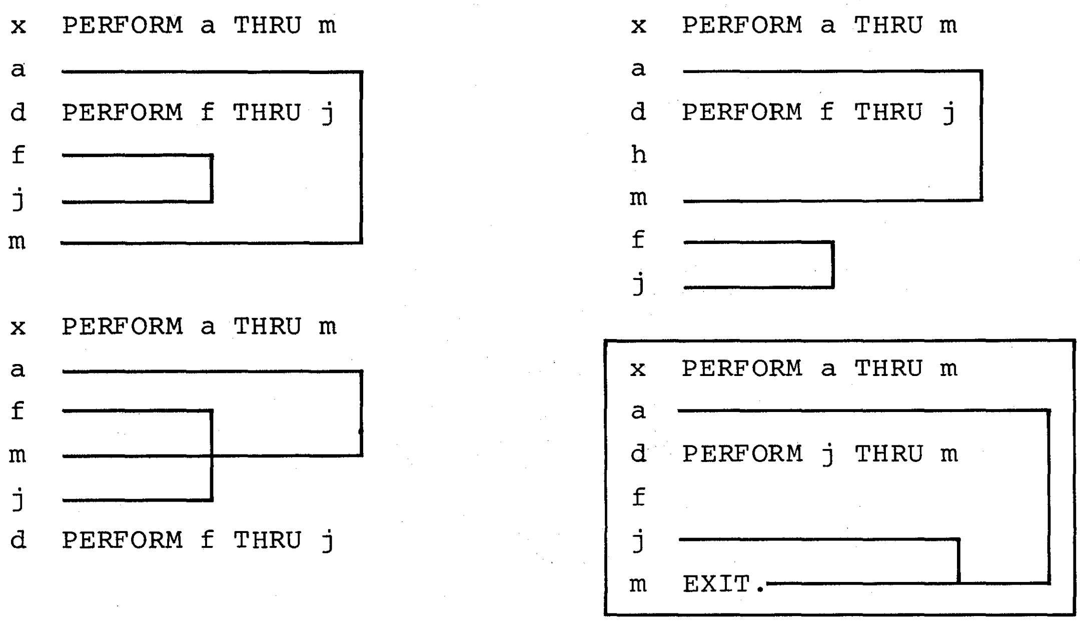


Figure 57. Valid PERFORM Statement Execution Sequences

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement, as if no PERFORM statement referred to these procedures.

The preceding rules refer to all 4 formats of the PERFORM statement. The following sections give rules applying to each individual format.

Format 1—Basic PERFORM

Format 1 is the basic PERFORM statement. The procedure(s) referred to are executed once, and then control passes to the next executable statement following the PERFORM statement.

Format 2—TIMES Option

Format 2 uses the TIMES option. Identifier-1 must name an integer item.

The procedure(s) referred to are executed the number of times specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement. The following rules apply:

1. If integer-1 or identifier-1 is 0 or a negative number at the time the PERFORM statement is initated, control passes to the statement following the PERFORM statement.

2. After the PERFORM statement has been initiated, any reference to identifier-1 has no effect in varying the number of times the procedures are initiated.

Format 3—Conditional PERFORM

Format 3 uses the UNTIL option. The procedure(s) referred to are performed until the condition specified by the UNTIL option is true. Control is then passed to the next executable statement following the PERFORM statement.

If the condition is true at the time the PERFORM statement is encountered, the specified procedure(s) are not executed.

The condition may be any condition described in the chapter on Conditional Expressions.

Format 4—VARYING Option

Format 4 uses the VARYING option. This option increments or decrements one or more identifiers or index-names according to the following rules. Once the condition(s) specified in the UNTIL option are satisfied, control is passed to the next executable statement following the PERFORM statement.

The condition may be any condition described in the chapter on Conditional Expressions.

No matter how many variables are specified, the following rules apply:

1. In the VARYING/AFTER options, when an index-name is specified:

   - The index-name is initialized and incremented or decremented according to the rules for the SET statement (see the chapter on Table Handling).

   - In the associated FROM option, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.

- In the associated BY option, an identifier must be described as an integer; a literal must be a nonzero integer.

2. In the FROM option, when an index-name is specified:

   - In the associated VARYING/AFTER option, an identifier must be described as an integer. It is initialized as described in the SET statement.

   - In the associated BY option, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.

3. In the BY option, identifiers and literals must have nonzero values.

4. Changing the values of identifiers and/or index-names in the VARYING, FROM, and BY options during execution changes the number of times the procedures are executed.

The way in which operands are incremented or decremented depends on the number of specified variables. In the following discussion explaining this, every reference to identifier-n refers equally to index-name-n (except when identifier-n is the object of the BY option).

Varying One Identifier: The following actions take place:

1. Identifier-1 is set equal to its starting value, identifier-2 or literal-2.

2. Condition-1 is evaluated:

   a. If it is false, steps 3 through 5 (below) are executed.

   b. If it is true, control passes directly to the statement following the PERFORM statement.

3. Procedure-1 through procedure-2 (if specified) are executed once.

4. Identifier-1 is augmented by identifier-3 (or literal-3).

5. Steps 2 through 4 are repeated until condition-1 is true.

Figure 58 illustrates the logic of the PERFORM statement when one identifier is varied.

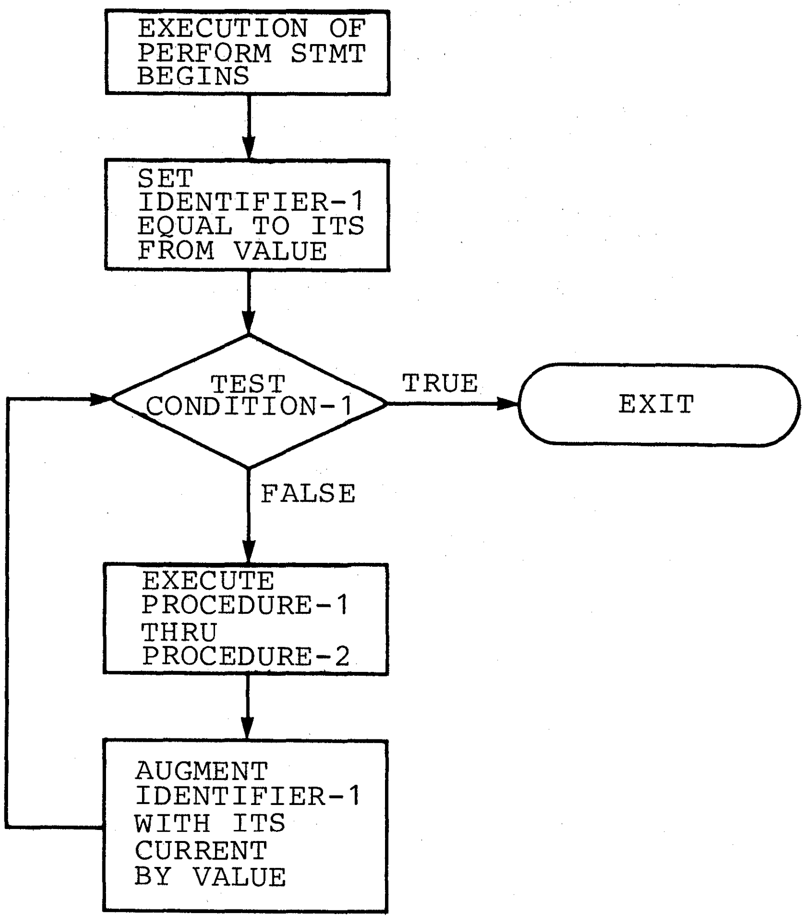

EXECUTION OF PERFORM STMT BEGINS
SET IDENTIFIER-1 EQUAL TO ITS FROM VALUE
TEST CONDITION-1
TRUE
EXIT
FALSE
EXECUTE PROCEDURE-1 THRU PROCEDURE-2
AUGMENT IDENTIFIER-1 WITH ITS CURRENT BY VALUE

Figure 58. Format 4 PERFORM Statement Logic—Varying One Identifier

Varying Two Identifiers: The following actions take place:

1. Identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.

2. Condition-1 is evaluated:

   a. If it is false, steps 3 through 7 (below) are executed.

   b. If it is true, control passes directly to the statement following the PERFORM statement.

3. Condition-2 is evaluated:

   a. If it is false, steps 4 through 6 (below) are executed.

   b. If it is true, identifier-4 is set to the current value of identifier-5, and identifier-1 is augmented by identifier-3 (or literal-3), and step 2 is repeated.

4. Procedure-1 through procedure-2 (if specified) are executed once.

5. Identifier-4 is augmented by identifier-6 (or literal-6).

6. Steps 3 through 5 (above) are repeated until condition-2 is true.

7. Steps 2 through 6 (above) are repeated until condition-1 is true.

At the end of PERFORM statement execution, identifier-4 contains the current value of identifier-5. Identifier-1 has a value that exceeds the last used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case identifier-1 contains the current value of identifier-2).

Figure 59 illustrates the logic of the PERFORM statement when two identifiers are varied.

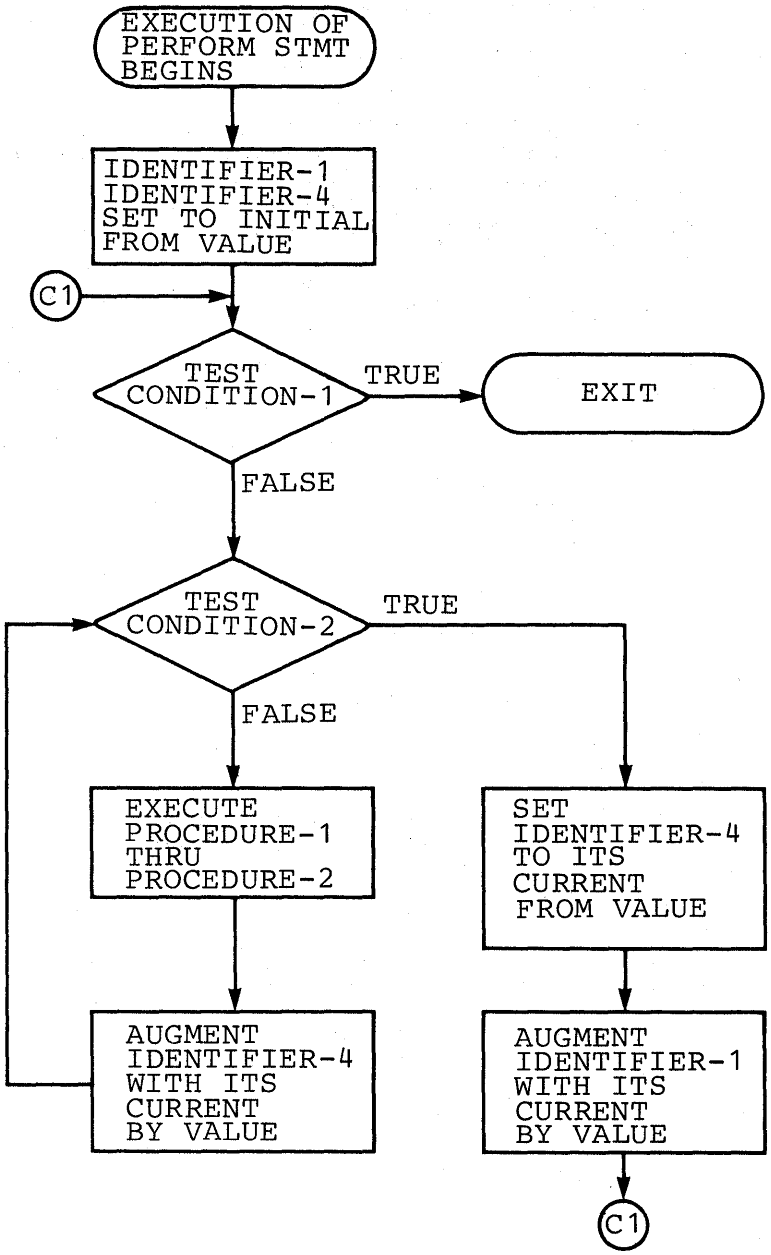

EXECUTION OF PERFORM STMT BEGINS
IDENTIFIER-1 IDENTIFIER-4 SET TO INITIAL FROM VALUE
C1
TEST CONDITION-1
TRUE
EXIT
FALSE
TEST CONDITION-2
TRUE
FALSE
EXECUTE PROCEDURE-1 THRU PROCEDURE-2
SET IDENTIFIER-4 TO ITS CURRENT FROM VALUE
AUGMENT IDENTIFIER-4 WITH ITS CURRENT BY VALUE
AUGMENT IDENTIFIER-1 WITH ITS CURRENT BY VALUE
C1

Figure 59. Format 4 PERFORM Statement Logic—Varying Two Identifiers

Varying Three Identifiers: The actions are the same as for two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

At the end of PERFORM statement execution, identifier-4 and identifier-7 contain the current values of identifier-5 and identifier-8, respectively. Identifier-1 has a value exceeding its last used setting by one increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement execution, in which case identifier-1 contains the current value of identifier-2).

Figure 60 illustrates the logic of the PERFORM statement when three identifiers are varied.

Programming Notes

In effect, the performed procedures are in a closed subroutine that can be entered from many other points in the program.

The Format 4 PERFORM statement is especially useful in Table Handling. One Format 4 PERFORM statement can serially search an entire 3-dimensional table.

Segmentation Information

A PERFORM statement appearing in a permanent segment can have in its range only one of the following:

1. Sections, each of which has a priority number less than 50

2. Sections and/or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have in its range only one of the following:

1. Sections, each of which has a priority number less than 50

2. Sections and/or paragraphs wholly contained within the same independent segment as the PERFORM statement.

IBM Extension

However, this IBM implementation allows a PERFORM statement in any segment to have sections with any priority numbers within its range of procedures.

End of IBM Extension

Control is passed to the performed procedures only once for each execution of the PERFORM statement.

Note: The rules for control transfers in performed procedures as previously implemented are given in Appendix A.

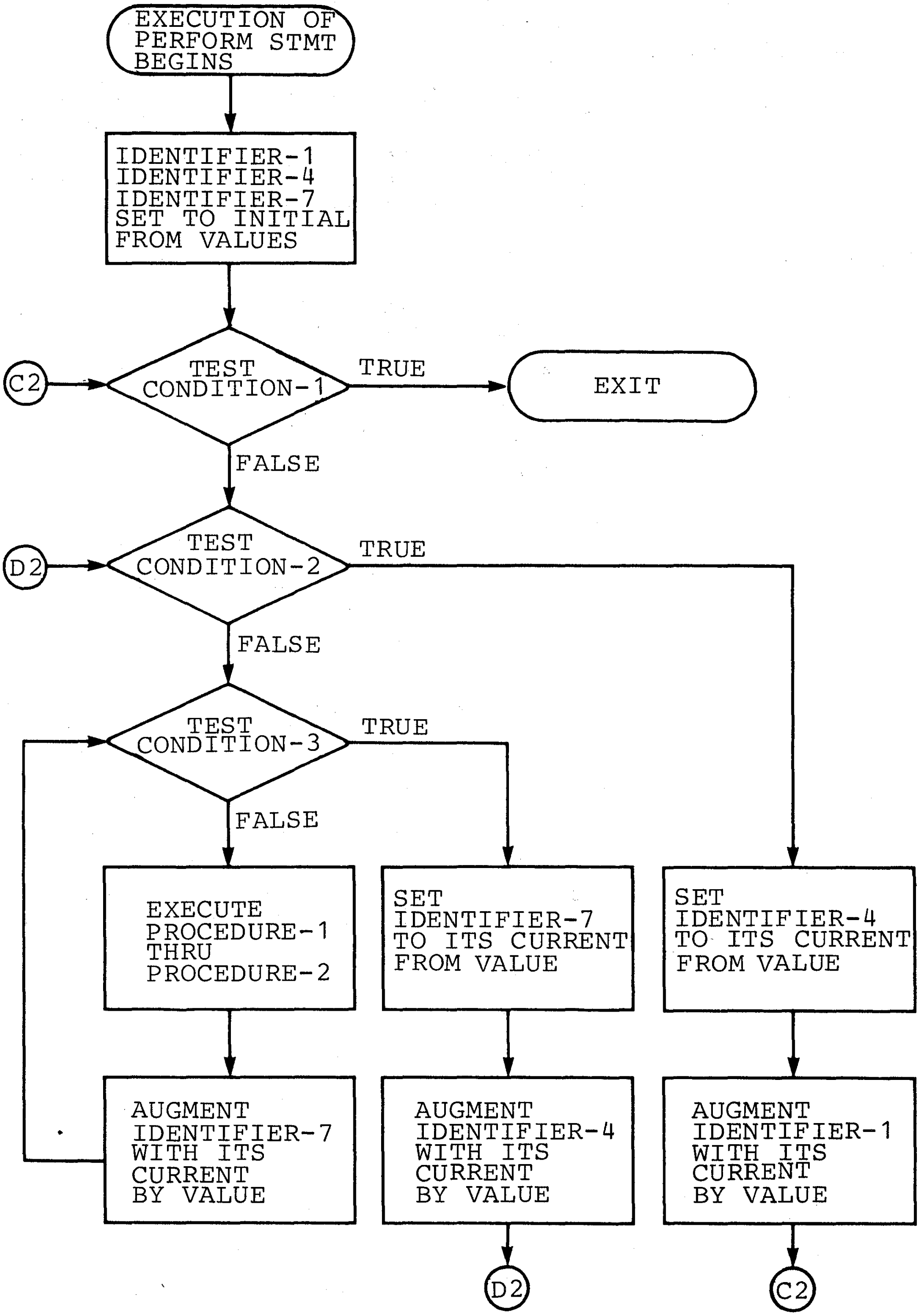

EXECUTION OF PERFORM STMT BEGINS
IDENTIFIER-1 IDENTIFIER-4 IDENTIFIER-7 SET TO INITIAL FROM VALUES
C2
TEST CONDITION-1
TRUE
EXIT
FALSE
D2
TEST CONDITION-2
TRUE
FALSE
TEST CONDITION-3
TRUE
FALSE
EXECUTE PROCEDURE-1 THRU PROCEDURE-2
SET IDENTIFIER-7 TO ITS CURRENT FROM VALUE
SET IDENTIFIER-4 TO ITS CURRENT FROM VALUE
AUGMENT IDENTIFIER-7 WITH ITS CURRENT BY VALUE
AUGMENT IDENTIFIER-4 WITH ITS CURRENT BY VALUE
AUGMENT IDENTIFIER-1 WITH ITS CURRENT BY VALUE
D2
C2

Figure 60. Format 4 PERFORM Statement Logic—Varying Three Identifiers

EXIT STATEMENT

The EXIT statement provides a common end point for a series of procedures.

Format

```
paragraph-name. EXIT [PROGRAM].
```

The EXIT statement must appear in a sentence by itself, and this sentence must be the only sentence in the paragraph.

The EXIT statement enables the user to assign a procedure-name to a given point in a program. The EXIT statement has no other effect on the compilation or execution of the program.

The EXIT PROGRAM statement is discussed in the chapter on Subprogram Linkage statements.

Programming Notes

The EXIT statement is useful for documenting the end point in a series of procedures. If an EXIT paragraph is written as the last paragraph in a Declarative procedure or a series of performed procedures, it identifies the point at which control will be transferred. When control reaches such an exit paragraph and the associated Declarative or PERFORM statement is active, control is transferred to the appropriate part of the Procedure Division. When control reaches such an EXIT paragraph, which is not the end of a range of procedures governed by an active PERFORM or USE statement, control passes through the EXIT statement to the first statement of the next paragraph.

Unless an EXIT statement is written, the end of the sequence is difficult to determine unless the reader knows the logic of the program.

STOP STATEMENT

The STOP statement halts the object program either temporarily or permanently.

Format

```
       ┌ RUN     ┐
STOP  <           >
       | literal |
       └         ┘
```

The literal may be numeric or nonnumeric, and may be any figurative constant except the ALL literal. If the literal is numeric, it must be an unsigned integer.

IBM Extension

However, this IBM implementation also allows signed numeric integer and noninteger literals.

End of IBM Extension

When the STOP literal is specified, the literal is communicated to the operator, and object program execution is suspended. Program execution is resumed only after operator intervention, and continues at the next executable statement in sequence.

When STOP RUN is specified, execution of the object program is terminated, and control is returned to the system. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or only statement in the sequence. All files should be closed before a STOP RUN statement is executed. (See the chapter on System Dependencies.)

For use of the STOP RUN statement in calling and called programs, see the System Dependencies chapter.

Programming Notes

The STOP literal statement is useful for special situations (a special tape or disk must be mounted, a specific daily code must be entered, and so forth). when operator intervention is needed during program execution.

COMPILER-DIRECTING STATEMENTS

Compiler-directing statements provide instructions to the COBOL compiler. The compiler-directing statements are COPY, ENTER, and USE.

Only the ENTER statement is discussed in this chapter. The COPY statement is discussed in the Source Program Library chapter.

IBM Extension

The Extended Source Program Library—the BASIS, INSERT, and DELETE statements—is discussed in the Source Program Library chapter.

End of IBM Extension

The USE statements are discussed in the chapters on Declaratives and the Debugging Features.

IBM Extension

The USE BEFORE REPORTING Declarative is discussed in the chapter on the Report Writer feature.

End of IBM Extension

ENTER STATEMENT

The ENTER statement is intended to provide a means of allowing the use of more than one source language in the same source program. This compiler allows no other source language in the source program.

Format

ENTER language-name [routine-name].

The language-name is a system name that has no defined meaning in this IBM implementation.

The language-name and routine-name must follow the rules for formation of a user-defined word; at least one character must be alphabetic.

The ENTER statement is accepted as documentation.

PART 5. SPECIAL FEATURES

- TABLE HANDLING
- SORT/MERGE

IBM Extension

- REPORT WRITER

End of IBM Extension

- SEGMENTATION
- SOURCE PROGRAM LIBRARY
- SUBPROGRAM LINKAGE
- DEBUGGING

TABLE HANDLING

A table, as used in data processing, is a collection of similar data items that is designed for easy retrieval of the group of related items as one unit, as well as of subgroups within the table.

Every item belonging to the table has the same data description. The items in a table could be defined separately as consecutive entries in the Data Division, but this practice has disadvantages. First, the unity of the items is not apparent from the documentation; second, the Procedure Division becomes a snarl of complicated references, both for the programmer and for the compiler at object time.

The table handling features of the COBOL language make it possible to refer to a large number of related items by one name.

TABLE HANDLING CONCEPTS

In COBOL, a table is defined with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as stated. The item so named is considered a table element, and its name and description apply to each repetition (or occurrence) of the item. Since the occurrences are not given unique data-names, reference to a particular occurrence can be made only by specifying the data-name of the table element, together with the occurrence number of the desired item within the element.

The occurrence number is known as a subscript, and the technique of supplying the occurrence number of individual table elements is called subscripting. A related technique, called indexing, is also available for table references. Both subscripting and indexing are described in the following sections.

TABLE DEFINITION

COBOL allows tables in one, two, or three dimensions.

To define a one-dimensional table, the user writes an OCCURS clause as part of the definition of a table element. However, the OCCURS clause must not appear in the description of a group item that contains the table element. For example:

```
01  TABLE-ONE.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10  ELEMENT-A PIC X(4).
        10  ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE names the table element of a one-dimensional table that occurs 3 times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

To define a two-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table.

For example:

```
01  TABLE-TWO.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10  ELEMENT-TWO OCCURS 3 TIMES.
            15  ELEMENT-A PIC X(4).
            15  ELEMENT-B PIC 9(4).
```

ELEMENT-ONE is an element of a one-dimensional table that occurs 3 times. ELEMENT-TWO is an element of a two-dimensional table that occurs 3 times within each occurrence of ELEMENT-ONE.

To define a <u>three-dimensional table</u>, a one-dimensional table is defined within each occurrence of another one-dimensional table which is itself contained within each occurrence of another one-dimensional table. For example:

```
01  TABLE-THREE.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10  ELEMENT-TWO OCCURS 3 TIMES.
            15 ELEMENT-THREE OCCURS 2 TIMES
               PICTURE X(8).
```

In this example, ELEMENT-ONE is an element of a one-dimensional table that occurs 3 times. ELEMENT-TWO is an element of a two-dimensional table that occurs 3 times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a three-dimensional table that occurs 2 times within each occurrence of ELEMENT-TWO. Figure 61 shows the storage layout for TABLE-THREE.

ELEMENT-ONE OCCURS 3 TIMES	ELEMENT-TWO OCCURS 3 TIMES	ELEMENT-THREE OCCURS 2 TIMES	byte displace-ment
ELEMENT-ONE (1)	ELEMENT-TWO (1 1)	ELEMENT-THREE (1 1 1)	0
		ELEMENT-THREE (1 1 2)	8
	ELEMENT-TWO (1 2)	ELEMENT-THREE (1 2 1)	16
		ELEMENT-THREE (1 2 2)	24
	ELEMENT-TWO (1 3)	ELEMENT-THREE (1 3 1)	32
		ELEMENT-THREE (1 3 2)	40
ELEMENT-ONE (2)	ELEMENT-TWO (2 1)	ELEMENT-THREE (2 1 1)	48
		ELEMENT-THREE (2 1 2)	56
	ELEMENT-TWO (2 2)	ELEMENT-THREE (2 2 1)	64
		ELEMENT-THREE (2 2 2)	72
	ELEMENT-TWO (2 3)	ELEMENT-THREE (2 3 1)	80
		ELEMENT-THREE (2 3 2)	88
ELEMENT-ONE (3)	ELEMENT-TWO (3 1)	ELEMENT-THREE (3 1 1)	96
		ELEMENT-THREE (3 1 2)	104
	ELEMENT-TWO (3 2)	ELEMENT-THREE (3 2 1)	112
		ELEMENT-THREE (3 2 2)	120
	ELEMENT-TWO (3 3)	ELEMENT-THREE (3 3 1)	128
		ELEMENT-THREE (3 3 2)	136
			144

Figure 61. Storage Layout for TABLE-THREE

TABLE REFERENCES

Whenever the user refers to a table element, or to any item associated with a table element, the reference must indicate which occurrence is intended.

For a one-dimensional table, the occurrence number of the desired element gives the complete information.

For tables of more than one dimension, an occurrence number for each dimension must be supplied. In the three-dimensional table defined above, for example, a reference to ELEMENT-THREE must supply the occurrence number for ELEMENT-ONE, ELEMENT-TWO, and ELEMENT-THREE. Either subscripting or indexing, described in the following paragraphs, can be used to supply the necessary references.

Subscripting

(see INDEXing on next page)

Subscripting is a method of providing table references through the use of subscripts. A subscript is an integer value that specifies the occurrence number of a table element.

Format

```
[ data-name-1    ]   [ OF ]
<                > [ <    > data-name-2] ...
| condition-name |   | IN |
L                J   L    J

      (subscript [subscript [subscript]])
```

Subscripts can be used only when reference is made to an individual item within a table element.

Data-name-1 must be the name of a table element, and may be qualified, if necessary. (Note that, when qualification is used, it is data-name-1, not data-name-2, that is subscripted.)

The subscript can be represented either by a literal or a data-name.

A literal subscript must be an integer, and must have a value of 1 or greater. The literal can have a positive sign or be unsigned. Negative subscript values are not permitted. For example, valid literal subscript references to TABLE-THREE are:

ELEMENT-THREE (1, 2, 1)

ELEMENT-THREE (2 2 1)

A data-name subscript must be described as an elementary numeric integer data item. A data-name subscript may be qualified; it may not be subscripted or indexed. For example, valid data-name subscript references to TABLE-THREE—assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM—are:

ELEMENT-THREE (SUB1 SUB2 SUB3)

ELEMENT-THREE IN TABLE-THREE (SUB1 OF SUBSCRIPT-ITEM, SUB2 OF SUBSCRIPT-ITEM,
SUB3 OF SUBSCRIPT-ITEM)

The set of one to three subscripts must be written within a balanced pair of parentheses immediately following data-name-1 or its last qualifier. One or more spaces may optionally precede the opening parenthesis.

When more than one subscript is specified, each subscript must be separated from the next by either a space or a comma followed by a space. (The comma is not required.)

When more than one subscript is required, the subscripts are written in the order of successively less-inclusive data dimensions. (That is, in the table reference ELEMENT-THREE (3, 2, 1), the first value (3) refers to the occurrence within ELEMENT-ONE, the second value (2) refers to the occurrence within ELEMENT-TWO, and the third value (1) refers to the occurrence within ELEMENT-THREE.)

The lowest possible subscript value is 1, which points to the first occurrence within the table element. The next sequential elements are pointed to by subscripts with values 2, 3, and so forth. The highest permissible subscript value in any particular table element is the maximum number of occurrences specified in the OCCURS clause. (In the last example given, the highest possible subscript value for ELEMENT-ONE is 3, for ELEMENT-TWO is 3, and for ELEMENT-THREE is 2.)

Indexing

(see SUBSCRIPTING on previous page)

Indexing is the method of providing table references through the use of indexes. An index is a compiler-generated storage area used to store table element occurrence numbers; the index contains a displacement value from the beginning of the table element that is equivalent to an occurrence number.

Format

```
┌                    ┐   ┌    ┐
│ data-name-1        │   │ OF │
<                    > [ <    > data-name-2] ...
│ condition-name     │   │ IN │
└                    ┘   └    ┘

     ┌                     ┌   ┐             ┐
     │ index-name-1 [ < ± > literal-2]      │
   ( <                     └   ┘             >
     │ literal-1                             │
     └                                       ┘
         ┌                    ┌   ┐                 ┐
         │ index-name-2  [ < ± >  literal-4]       │
       [ <                    └   ┘                 >
         │ literal-3                                │
         └                                          ┘
            ┌                    ┌   ┐              ┐
            │ index-name-3 [ < ± > literal-6]      │
          [ <                    └   ┘              > ] ])
            │ literal-5                             │
            └                                       ┘
```

Each index-name identifies an index to be used in table references. The index-name is specified through the OCCURS clause. The same index-name defined with one table can be used to index another table if both table descriptions are identical: They must have the same number of levels, the same number of occurrences, as well as occurrences of the same length.

Each index named is a compiler-generated storage area, 4 bytes in length; each index contains a binary value that represents an actual displacement from the beginning of the table element; to be valid, the displacement value must correspond to that of an occurrence number in the table element. Two forms of indexing are provided: direct and relative.

In direct indexing, the index-name is in the form of a subscript. The value contained in the index is then calculated as the occurrence number minus one, multiplied by the length of the individual table entry. For example:

```
05  ELEMENT-A OCCURS 10 INDEXED BY INX-A PIC X(8).
```

For the fifth occurrence of ELEMENT-A, the binary value contained in INX-A is (5 - 1) * 8 = 32.

In relative indexing, the index-name is followed by a space, followed by a + or a -, followed by another space, followed by an unsigned numeric literal. The literal is considered to be an occurrence number, and is converted to an index value before being added to or subtracted from the index-name.

For example, if indexing is specified for the table shown in the previous table layout example for TABLE-THREE, as follows:

```
01  TABLE-THREE..
  05  ELEMENT-ONE OCCURS 3 TIMES INDEXED BY INX-1.
    10  ELEMENT-TWO OCCURS 3 TIMES INDEXED BY INX-2.
      15  ELEMENT-THREE OCCURS 2 INDEXED BY INX-3
        PICTURE X(8).
```

Then a relative indexing reference to

```
ELEMENT-THREE (INX-1 + 1, INX-2 + 2, INX-3 -1)
```

causes the following computation of the displacement:

```
  (address of ELEMENT-ONE)
+ (contents of INX-1) + (48 * 1)
+ (contents of INX-2) + (16 * 2)
+ (contents of INX-3) - (8 * 1)
```

(That is, each occurrence of ELEMENT-ONE is 48 characters in length; each occurrence of ELEMENT-TWO is 16 characters in length; and each occurrence of ELEMENT-THREE is 8 characters in length.)

To be valid during execution, an index value must correspond to a table element occurrence not less than one, nor greater than the highest permissible occurrence number. This restriction applies to both direct and relative indexing.

An index-name must be initialized through a SET, PERFORM, or SEARCH ALL statement before it is used in a table reference.

Further information on index-names is given in the description of the INDEXED BY option of the OCCURS clause, later in this chapter.

One or more index references (direct or relative) may be specified together with literal subscripts.

Restrictions on Subscripting and Indexing

1. A data-name must not be subscripted or indexed when it is being used as a subscript or qualifier.

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

3. An index may be modified only by a PERFORM, SEARCH, or SET statement.

4. When a literal is used in a subscript, it must be a positive or unsigned integer.

5. When a literal is used in relative indexing, it must be an unsigned integer.

6. For optimization of references to subscripted data-names, code is generated by the compiler to store the calculated address of a subscripted data-name in a SUBSCRIPT SAVE CELL. When the same subscripted reference to that data-name is used again, and no change to the subscript could have taken place between two references, the address of the subscripted data-name is not recalculated, but the saved address is used instead. It is possible that the subscript value may have changed whenever a new paragraph or section is encountered or whenever a verb is encountered that may transfer control outside the current paragraph, such as a PERFORM verb. Also, any direct alteration to the subscript variable, or a move to a group that contains the subscript will cause the subscripted address to be recalculated. However, altering the content of the subscript through indirect reference, such as changing the contents of an item that redefines the

subscript, is not recognized by the compiler, and will not cause the subscripted address to be recalculated.

7. Subscripting and indexing must not be used together in a single reference.

TABLE INITIALIZATION

A table can contain static values or dynamic values.

Static values remain the same through every execution of the object program. When this is true, the initial values of table elements can be specified in Working-Storage in one of two ways:

1. The table can be described as a record containing contiguous subordinate data description entries, each of which contains a VALUE clause for the initial value. The record is then redescribed through a REDEFINES entry that contains a subordinate entry with an OCCURS clause. Due to the OCCURS clause, the subordinate entries of the redefined entry are repeated. For example:

```
01  TABLE-ONE.
    05  ELEMENT-ONE PICTURE X VALUE "1".
    05  ELEMENT-TWO PICTURE X VALUE "2".
    05  ELEMENT-THREE PICTURE X VALUE "3".
    05  ELEMENT-FOUR PICTURE X VALUE "4".
01  TABLE-TWO REDEFINES TABLE-ONE.
    05  OCCURS-ELEMENT OCCURS 4 TIMES PICTURE X.
```

2. If the subordinate entries do not require separate handling, the VALUE of the entire entry can be given in the entry which names the table. The lower level entries then contain OCCURS clauses, and show the hierarchic structure of the table. The subordinate entries must not contain VALUE clauses. For example:

```
01  TABLE-ONE VALUE "1234".
    05   TABLE-TWO OCCURS 4 TIMES PICTURE X.
```

Dynamic values may change during one execution of the object program, or from one execution to another. If the dynamic values are always the same at the beginning of object program execution, they can be initialized in the same manner as static values. If the initial values change from one execution to the next, then the table can be defined without initial values, and the changed values can be placed in the table before any table reference is made.

TABLE HANDLING—DATA DIVISION

COBOL Data Division clauses used for Table Handling are the OCCURS clause and the USAGE IS INDEX clause.

OCCURS CLAUSE

The OCCURS clause eliminates the need for separate entries for repeated data items; it also supplies the information necessary for the use of subscripts or indexes.

Format 1—Fixed Length Tables

```
OCCURS integer-2 TIMES

      ┌ ASCENDING  ┐
  [   <            > KEY IS data-name-2 [data-name-3] ... ]
      | DESCENDING |
      └            ┘

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

Format 2—Variable Length Tables

```
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] ... ]
```

The subject of an OCCURS clause is the data-name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the described item.

Whenever it is used in any statement—other than SEARCH, or unless it is the object of a REDEFINES clause—the subject must be subscripted or indexed. When subscripted or indexed, the subject refers to one occurrence within the table element.

Whenever it is used in a SEARCH statement, or when it is the object of a REDEFINES clause, the subject must not be subscripted or indexed. When not subscripted or indexed, the subject represents the entire table element.

A table element must not be greater than 32767 bytes in length.

No table may be longer than 32767 bytes, except for fixed-length tables in the Working-Storage section or Linkage section, which may be as long as 131071 bytes.

All data-names used in the OCCURS clause may be qualified; they may not be subscripted or indexed.

All integers must be positive integers.

The OCCURS clause cannot be specified in a data description entry that:

- Has a level-01, level-66, level-77, or level-88 number.

- Describes an item of variable size (an item is of variable size if any subordinate entry contains an OCCURS DEPENDING ON clause).

- Describes a redefined data item. (However, a redefined item can be subordinate to an item containing an OCCURS clause.)

See REDEFINES Clause description in the Data Division chapter.

Format 1—Fixed Length Tables

When Format 1 is used, integer-2 specifies the exact number of occurrences.

Integer-2 must be greater than 0 and less than 32768.

Because three subscripts or indexes are allowed, three nested levels of the Format 1 OCCURS clause are allowed.

Format 2—Variable Length Tables

Format 2 specifies the OCCURS DEPENDING ON clause; integer-1 represents the minimum number of occurrences and integer-2 represents the maximum number of occurrences. The value of integer-1 must be 1 or greater; it must also be less than integer-2. Integer-2 must be less than 32768.

(Note that the length of the subject item is fixed; it is only the number of repetitions of the subject item that is variable.)

Data-name-1 specifies the object of the OCCURS DEPENDING ON clause; that is, the data item whose current value represents the current number of occurrences of the subject item. The object of the OCCURS DEPENDING ON clause:

- Must be described as a positive integer. (That is, if data-name-1 is described as a signed item, then at execution time it must contain positive data.)
- Must not occupy any storage position within the range of this table (that is, any storage position from the first character position in this table through the last character position in this record description entry).
- Must contain a value within the range of integer-1 and integer-2 inclusive.

IBM Extension

- Must not be an external-decimal Status Key data item.

End of IBM Extension

The value of the object of the OCCURS DEPENDING ON clause specifies that part of the table element available to the object program. (Items whose occurrence numbers exceed the value of the object are not available.) If, during execution, the value of the object is reduced, the contents of items whose occurrence numbers exceed the new value of the object are unpredictable.

When a group item containing a subordinate OCCURS DEPENDING ON item is referred to, the current value of the object determines that part of the table area used in the operation.

In one record description entry, any entry that contains an OCCURS DEPENDING ON clause may be followed only by items subordinate to it.

Note that the OCCURS DEPENDING ON clause cannot be specified as subordinate to another OCCURS clause. However, the Format 1 OCCURS clause may be specified as subordinate to the OCCURS DEPENDING ON clause; in this case, a table of as many as three dimensions may be specified.

ASCENDING/DESCENDING KEY Option—Formats 1 and 2

The ASCENDING/DESCENDING KEY option specifies that the repeated data is arranged in ascending or descending order (depending on the key word specified) according to the values contained in data-name-2, data-name-3, and so forth. The data-names are listed in their descending order of significance.

The order is determined by the rules for comparison of operands (see the relation condition description in the Conditional Expressions chapter). The ASCENDING/DESCENDING KEY data items are used by the SEARCH ALL statement for a binary search of the table element.

Data-name-2 must be the name of the subject entry, or the name of an entry subordinate to the subject entry.

If data-name-2 names the subject entry, that entire entry becomes the ASCENDING/DESCENDING KEY, and is the only key that may be specified for this table element.

If data-name-2 does not name the subject entry, then data-name-2, data-name-3, and so forth:

- Must be subordinate to the subject of the table entry itself

- Must not be subordinate to any other entry that contains an OCCURS clause

- Must not themselves contain an OCCURS clause

The following example illustrates the specification of ASCENDING/DESCENDING KEY data items:

```
WORKING-STORAGE SECTION.
01  TABLE-RECORD.
    05  EMPLOYEE-TABLE OCCURS 100 TIMES
        ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO
        INDEXED BY A,  B.
        10  EMPLOYEE-NAME              PIC X(20).
        10  EMPLOYEE-NO                PIC 9(6).
        10  WAGE-RATE                  PIC 9999V99.
        10  WEEK-RECORD OCCURS 52 TIMES
            ASCENDING KEY IS WEEK-NO INDEXED BY C.
            15  WEEK-NO                PIC 99.
            15  AUTHORIZED-ABSENCES    PIC  9.
            15  UNAUTHORIZED-ABSENCES  PIC  9.
            15  LATENESSES             PIC  9.
```

Note that the keys for EMPLOYEE-TABLE are subordinate to that entry; the key for WEEK-RECORD is subordinate to that subordinate entry.

When the ASCENDING/DESCENDING KEY option is specified, the following rules apply:

- Keys must be listed in decreasing order of significance.

- The total number of keys for a given table element must not exceed 12.

- The sum of the lengths of all the keys associated with one table element must not exceed 256.

- A key may have DISPLAY or COMPUTATIONAL usage.

IBM Extension

- A key may have COMPUTATIONAL-3 or COMPUTATIONAL-4 usage.

End of IBM Extension

- It is the programmer's responsibility to ensure that the data actually present in the table is arranged in

ASCENDING/DESCENDING sequence according to the collating sequence in use.

(In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE, and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in WEEK-RECORD must be arranged in ascending order of WEEK-NO. If they are not, SEARCH ALL statement results will be unpredictable.)

INDEXED BY Option—Formats 1 and 2

The INDEXED BY option specifies the indexes that can be used with this table element. The INDEXED BY option is required if indexing is used to refer to this table element.

IBM Extension

However, this IBM implementation allows a table without an INDEXED BY clause to be referred to through indexing.

End of IBM Extension

Each index-name must follow the rules for formation of a user-defined word; at least one character must be alphabetic.

Each index-name specifies an index to be created by the compiler for use by the program. These index-names are not data-names, and are not identified elsewhere in the COBOL program; instead, they can be regarded as private special registers for the use of this object program only. They are not data, or part of any data hierarchy; as such, each private special register must be unique.

In one table entry, up to 12 index-names may be specified.

(More information is given in the Indexing paragraphs of the preceding Table Handling Concepts section.)

USAGE IS INDEX CLAUSE

The USAGE IS INDEX clause specifies that the data item named has an index format. Such an item is an index data item.

Format

[USAGE IS] INDEX

An index data item is a 4-byte elementary item (not necessarily connected with any table) that can be used to save index-name values for future reference. Through the SET statement, an index data item can be assigned an index-name value (that is, (occurrence-number - 1) * entry length); such a value equals the displacement for an occurrence number in a table.

An index data item can be referred to directly only in a SEARCH statement, a SET statement, a relation condition, the USING option of the Procedure Division header, or the USING option of the CALL statement.

IBM Extension

An index data item can be referred to directly in the USING option of an ENTRY statement.

End of IBM Extension

An index data item can be part of a group item referred to in a MOVE statement or an input/output statement.

An index data item saves binary values equivalent to the table occurrences, yet is not itself necessarily defined as part of any table. Thus, when it is referred to directly in a SEARCH or

SET statement, or indirectly in a MOVE or input/output statement, there is no conversion of values when the statement is executed.

The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item cannot be a conditional variable.

The JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

IBM Extension

However, this compiler allows the use of SYNCHRONIZED when USAGE IS INDEX to obtain efficient use of the index data item.

End of IBM Extension

TABLE HANDLING—PROCEDURE DIVISION

In the Procedure Division, the SEARCH and SET statements may be specified with indexed tables. There are also special rules involving Table Handling elements when they are used in relation conditions.

RELATION CONDITIONS

Comparisons involving index-names and/or index data items conform to the following rules:

- The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
- In the comparison of an index-name with a data item (other than an index data item), or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
- In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion. Results of any other comparison involving an index data item are undefined.

Figure 62 gives comparisons for index-names and index data items.

Second Operand First Operand	Index-name	Index Data Item	Data-name (numeric integer only)	Numeric literal (integer only)
Index-name	Compare occurrence number	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index Data Item	Compare without conversion	Compare without conversion	Illegal	Illegal
Data-name (numeric integer only)	Compare occurrence number with data-name	Illegal		
Numeric literal (integer only)	Compare occurrence number with literal	Illegal		

Figure 62. Comparisons for Index-Names and Index Data Items

SET STATEMENT

The SET statement establishes reference points for table handling operations by placing values associated with table elements into indexes associated with index-names.

Format 1

```
     ┌                                        ┐     ┌                  ┐
     | index-name-1 [index-name-2] ...        |     | index-name-3     |
SET  <                                        > TO  < identifier-3     >
     | identifier-1 [identifier-2] ...        |     | literal-1        |
     └                                        ┘     └                  ┘
```

Format 2

```
SET index-name-4 [index-name-5] ...
    ┌         ┐  ┌                ┐
    | UP BY   |  | identifier-4   |
    <         >  <                >
    | DOWN BY |  | literal-2      |
    └         ┘  └                ┘
```

Index-names are related to a given table through the INDEXED BY option of the OCCURS clause; they are not further defined in the program.

When the sending and receiving fields in a SET statement share part of their storage (that is, the operands overlap), the result of the execution of such a SET statement is undefined.

Format 1—TO Option

When this form of the SET statement is executed, the value of the sending field replaces (with or without conversion) the current value of the receiving field.

The receiving field may be specified as:

- Index-name-1, index-name-2, and so forth, or
- Identifier-1, identifier-2, etc.—which must name either index data items or elementary numeric integer items.

The sending field may be specified as:

- Identifier-3—which must name either an index data item or an elementary numeric integer item, or
- Index-name-3—whose value before the SET statement is executed must correspond to an occurrence number of its associated table, or
- Literal-1—which must be a positive integer.

Figure 63 shows valid combinations of sending and receiving fields in a Format 1 SET statement.

Execution of the Format 1 SET statement depends upon the type of receiving field, as follows:

- Index-name Receiving Fields (index-name-1, index-name-2, and so forth), with one exception, are converted to a displacement value corresponding to the occurrence number indicated by the sending field. To be valid, the resulting index-name value must correspond to an occurrence number in its associated table element. The one exception to these rules is: If the sending field is an index data item, the value in the index data item is placed in the index-name without change.
- Index Data Item Receiving Fields (identifier-1, identifier-2, etc.) are set equal to the contents of the sending field (which must be either an index-name or an index data item); no conversion takes place. A numeric integer or literal sending field must not be specified.
- Integer Data Item Receiving Fields (identifier-1, identifier-2, and so forth) are set to an occurrence number that corresponds to the occurrence number associated with the sending field (which must be an index-name). An integer data item, an index data item, or a literal sending field must not be specified.

Receiving fields are acted upon in the left-to-right order in which they are specified. Any subscripting or indexing associated with an identifier receiving field is evaluated immediately before the field is acted upon.

The value used for the sending field is its value at the beginning of SET statement execution.

Sending Field		**Receiving Field**	
	Index-name	**Index Data Item**	**Integer Data Item**
Index-name	Valid	Valid*	Valid
Index Data Item	Valid*	Valid	
Integer Data Item	Valid		
Integer Literal	Valid		

*No conversion takes place

Figure 63. Sending and Receiving Fields for Format 1 SET Statement

The value for an index-name after execution of a SEARCH or PERFORM statement may be undefined; therefore, a Format 1 SET statement should reinitialize such index-names before other table handling operations are attempted.

Format 2—UP/DOWN BY Option

When this form of the SET statement is executed, the value of the receiving field is increased (UP BY) or decreased (DOWN BY) a value that corresponds to the value in the sending field.

The <u>receiving field</u> may be specified by index-name-4, index-name-5, and so forth. These index-name values both before and after the SET statement execution must correspond to an occurrence number in an associated table.

The <u>sending field</u> may be specified as identifier-4, which must be an elementary integer data item, or as literal-2, which must be an integer.

When the Format 2 SET statement is executed, the contents of the receiving field are increased (UP BY) or decreased (DOWN BY) a value that corresponds to the number of occurrences represented by the value of identifier-4 or literal-2. Receiving fields are acted upon in the left-to-right order they are specified. The value of the sending field at the beginning of SET statement execution is used for all receiving fields.

SEARCH STATEMENT

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element.

Format 1—Serial Search

```
                                 ┌                ┐
                                 | identifier-2   |
SEARCH identifier-1 [VARYING    <                  >   ]
                                 | index-name-1   |
                                 └                ┘

  [AT END imperative-statement-1]

                        ┌                          ┐
                        | imperative-statement-2   |
  WHEN condition-1     <                            >
                        | NEXT SENTENCE            |
                        └                          ┘

                        ┌                          ┐
                        | imperative-statement-3   |
 [WHEN condition-2     <                            > ] ...
                        | NEXT SENTENCE            |
                        └                          ┘
```

Format 2—Binary Search

```
SEARCH ALL identifier-1

  [AT END imperative-statement-1]

     ┌ relation-condition-1 ┐
WHEN <                      >
     │ condition-name-1     │
     └                      ┘

        ┌ relation-condition-2 ┐
    [AND <                      >  ] ...
        │ condition-name-2     │
        └                      ┘

    ┌ imperative-statement-2 ┐
    │                        │
    │ NEXT SENTENCE          │
    └                        ┘
```

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY option.

When specified in the SEARCH statement, identifier-1 must refer to all occurrences within the table element; that is, it must not be subscripted or indexed.

Identifier-1 can be a data item subordinate to a data item that contains an OCCURS clause; that is, it can be a part of a two or three dimensional table. In this case, the data description entry must specify an INDEXED BY option for each dimension of the table.

SEARCH statement execution modifies only the value in the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two or three dimensional table, it is necessary to execute a SEARCH statement for each dimension. Before each execution, SET statements must be executed to reinitialize the associated index-names.

In the AT END and WHEN options, if any of the specified imperative-statements do not end with a GO TO statement, control passes to the next sentence after the imperative-statement is executed.

Format 1—Serial Search

Format 1 SEARCH statement execution causes a serial search to be executed, beginning at the current index setting.

When the search begins, if the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, the following actions take place:

1. The condition(s) in the WHEN option are evaluated in the order they are written.

2. If none of the conditions is satisfied, the index-name for identifier-1 is increased to correspond to the next table element, and step 1 is repeated.

3. If, upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition.

4. If the end of the table is reached (that is, the increased index-name value is greater than the highest possible occurrence number) without the WHEN condition being satisfied, the search terminates as described in the next paragraph.

If, when the search begins, the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and, if specified, the AT END imperative-statement is executed. If the AT END option is omitted, control passes to the next sentence.

Each WHEN option condition may be any condition as described in the chapter on Conditional Expressions.

VARYING INDEX-NAME-1 OPTION: When the VARYING index-name-1 option is omitted, the first (or only) index-name for identifier-1 is used for the search.

When the VARYING index-name-1 option is specified, one of the following actions applies:

- If index-name-1 is an index for identifier-1, this index is used for the search. Otherwise, the first (or only) index-name is used.

- If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is increased by the same amount as the search index-name and at the same time.

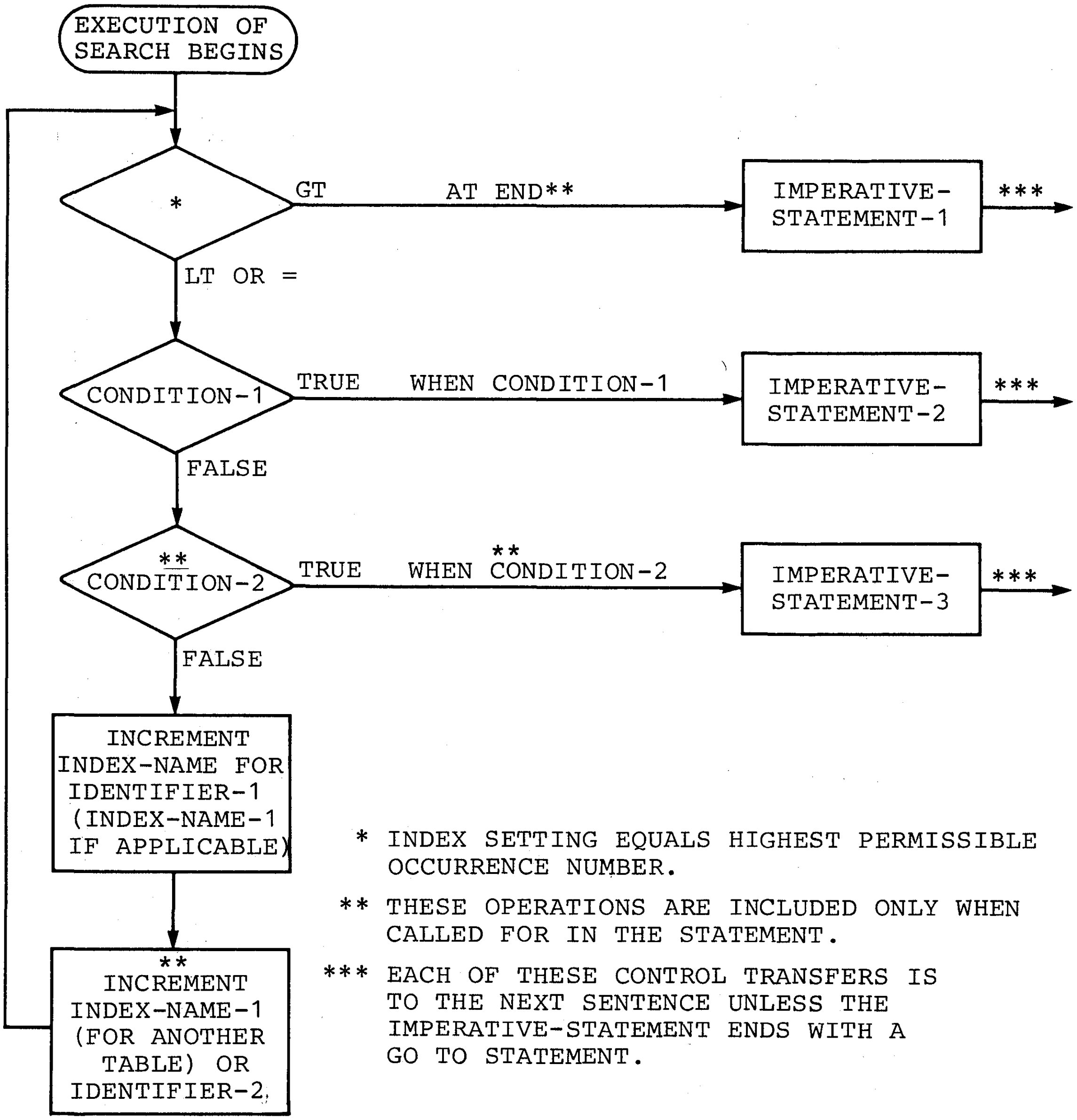
VARYING IDENTIFIER-2 OPTION: When this option is specified, the first (or only) index-name for identifier-1 is used for the search.

Identifier-2 must be either an index data item or an elementary integer item. During the search, one of the following actions applies:

- If identifier-2 is an index data item, then, whenever the search index is increased, the specified index data item is simultaneously increased by the same amount.

- If identifier-2 is an integer data item, then, whenever the search index is increased, the specified data item is simultaneously increased by 1.

Figure 64 is a flowchart of a Format 1 SEARCH operation containing two WHEN options.



* INDEX SETTING EQUALS HIGHEST PERMISSIBLE OCCURRENCE NUMBER.

** THESE OPERATIONS ARE INCLUDED ONLY WHEN CALLED FOR IN THE STATEMENT.

*** EACH OF THESE CONTROL TRANSFERS IS TO THE NEXT SENTENCE UNLESS THE IMPERATIVE-STATEMENT ENDS WITH A GO TO STATEMENT.

Figure 64. Format 1 SEARCH with Two WHEN Options

Format 2—Binary Search

Format 2 SEARCH ALL statement execution causes a binary search to be executed. The search index need not be initialized by SET statements, because its setting is varied during the search operation so that its value is at no time less than the value for the first table element, nor ever greater than the value for the last table element. The index used is always that associated with the first index-name specified in the OCCURS clause.

If the WHEN option cannot be satisfied for any setting of the index within this range, the search is unsuccessful. If the AT END option is specified, the AT END imperative-statement is executed. If the AT END option is not specified, control is passed to the next sentence. In either case, the final setting of the index is not predictable.

If the WHEN option can be satisfied, control passes to imperative-statement-2, and the index contains a value indicating an occurrence that allows the WHEN condition(s) to be satisfied.

WHEN CONDITION-NAME OPTION: If the WHEN condition-name option is specified, each specified condition-name must have only a single value, and each must be associated with an ASCENDING/DESCENDING KEY identifier for this table element.

WHEN RELATION-CONDITION OPTION: If the WHEN relation-condition is specified, each relation-condition must take the following form:

Format

```
           [ IS EQUAL TO ]  [ identifier-2          ]
data-name  <             >  < literal               >
           | IS =        |  | arithmetic-expression |
           L             J  L                       J
```

The following considerations apply:

- Data-name must specify an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be indexed by the first identifier-1 index-name, along with other indexes or literals as required. Each data-name may be qualified.

- Identifier-2 must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.

- Arithmetic-expression may be any of those defined in the chapter on Arithmetic Expressions with the following restriction: Any identifier in the arithmetic-expression must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.

- When an ASCENDING/DESCENDING KEY data item is specified, explicitly or implicitly, in the WHEN option, then all preceding ASCENDING/DESCENDING KEY data-names for identifier-1 must also be specified.

The results of a SEARCH ALL operation are predictable only when:

- The data in the table is ordered in ASCENDING/DESCENDING KEY order, and

- The contents of the ASCENDING/DESCENDING keys specified in the WHEN clause provide a unique table reference.

PROGRAMMING NOTES

Index data items cannot be used as subscripts or indexes, due to the restrictions on direct reference to them.

The use of a direct indexing reference together with a relative indexing reference for the same index-name allows reference to two different occurrences of a table element for comparison purposes, and so forth.

When the object of the VARYING option is an index-name for another table element, one Format 1 SEARCH statement steps through two table elements at once. (This is illustrated in the

INQUIRY-PRINT procedure of the TABLES sample program at the end of this chapter.)

One Format 4 PERFORM statement can search an entire multidimensional table.

To ensure correct execution of a PERFORM or SEARCH statement for a variable length table, the programmer must make sure that the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that correctly specifies the current length of the table.

IBM Extension

If indexing is used to search a table without an INDEXED BY clause, correct results are ensured only if both the table defined with the index and the table defined without the index have table elements of the same length and with the same number of occurrences.

End of IBM Extension

TABLE HANDLING SAMPLE PROGRAM

This program illustrates one method of building a table, as well as two methods of searching a table.

A chain store doing business in the continental United States divides the country into seven sales areas. A weekly report of sales is made for each area. This program places these sales figures into a cumulative table, and then issues a report comparing sales by area for the current week with those for the preceding week, as well as for the same week in the preceding year. Optional inquiries can be made to report—for any week of the current year—which areas increased sales over last year.

The following Table Handling features are illustrated: the OCCURS clause, the OCCURS DEPENDING ON clause, the INDEXED BY option, direct indexing, relative indexing, the PERFORM UNTIL statement used to search a table element, the SEARCH VARYING STATEMENT used to search two table elements simultaneously, the SET TO statement, and the SET UP BY statement.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABLES.
DATE-COMPILED. MAR 20,1981.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
SPECIAL-NAMES.
    CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CURRENT-FILE
        ASSIGN TO SYS002-UT-2400-S-TAPE1.
    SELECT YR-TO-DATE-FILE
        ASSIGN TO SYS003-DA-3330-S-DISK1.
    SELECT LAST-YR-FILE
        ASSIGN TO SYS004-DA-3330-S-DISK2.
    SELECT PRINT-FILE
        ASSIGN TO SYS001-UR-1403-S-SYSLST.
    SELECT OPTIONAL INQUIRY-FILE
        ASSIGN TO SYS005-DA-3330-S-DISK3.
DATA DIVISION.
FILE SECTION.
FD  CURRENT-FILE LABEL RECORDS OMITTED.
01  CURRENT-SALE-RECORD.
    05  CURR-WEEK PIC 99.
    05  CURR-FIGURES OCCURS 7 TIMES INDEXED BY WEEKC.
        10  CURR-AREA-NO PIC 9.
        10  CURR-AREA-SALE PIC S99999V99.
FD  YR-TO-DATE-FILE LABEL RECORDS STANDARD.
```

```
01  YR-TO-DATE-RECORD.
    05  YR-TO-DATE-TABLE OCCURS 52 TIMES INDEXED BY WEEKL.
        10  FILLER PIC 99.
        10  FILLER OCCURS 7 TIMES INDEXED BY WEEKW.
            15  FILLER PIC 9.
            15  FILLER PIC S99999V99.
FD  LAST-YR-FILE LABEL RECORDS STANDARD.
01  LAST-YR-RECORD.
    05  LAST-YR-WEEK-TABLE OCCURS 52 TIMES
            INDEXED BY WEEKY WEEKZ.
        10  LAST-YR-WEEK-NO PIC 99.
        10  LAST-YR-AREA-TOTALS OCCURS 7 TIMES INDEXED BY AREAY.
            15  LAST-YR-AREA-NO PIC 9.
            15  LAST-YR-AREA-FIGURES PIC S99999V99.
FD  PRINT-FILE LABEL RECORDS OMITTED.
01  PRINT-RECORD.
    05  FILLER PIC X.
    05  PRINT-DATA PIC X(95).
FD  INQUIRY-FILE LABEL RECORDS OMITTED.
01  INQUIRY-RECORD PIC 99.
WORKING-STORAGE SECTION.
77  THIS-WEEK PIC 99 COMP-3.
77  LAST-WEEK PIC 99 COMP-3.
77  ANY-INQ PIC 9 VALUE 0.
77  INQ-COND-MET    PIC 9.
77  NO-INQ-MSG PIC X(21) VALUE " NO INQUIRY WAS MADE.".
01  NONE-MET-COND.
    05  FILLER PIC X(17) VALUE " NO AREA IN WEEK ".
    05  COND-MSG-WK PIC Z9.
    05  FILLER PIC X(29) VALUE " EXCEEDED SALES OF LAST YEAR.".
01  YTD-WORK-TABLE.
    05  WEEK-TABLE OCCURS 1 TO 52 TIMES
        DEPENDING ON THIS-WEEK
        INDEXED BY WEEKT WEEKX.
        10  WEEK-NO PIC 99.
        10  AREA-TOTALS OCCURS 7 TIMES
            INDEXED BY AREAT AREAX.
            15  AREA-NO PIC 9.
            15  AREA-FIGURES PIC S99999V99.
01  CURRENT-REPORT-PRINT.
    05  FILLER PIC X(10) VALUE " WEEK NO.".
    05  PRINT-WEEK PIC Z9.
    05  PRINT-AREA VALUE "  AREA   ".
        10  FILLER PIC X(4).
        10  PRINT-AREA-NO PIC 9.
        10  FILLER PIC X(4).
    05  PRINT-AREA-FIGURES.
        10  FILLER PIC X(17) VALUE "THIS WEEK SALES ".
        10  PRINT-AREA-SALES PIC $$$,$$$.99.
        10  FILLER PIC X(5).
    05  PRINT-AREA-LAST-WEEK.
        10  FILLER PIC X(17) VALUE "LAST WEEK SALES ".
        10  LAST-WEEK-SALES PIC $$$,$$$.99.
        10  FILLER PIC XX.
    05  WEEK-PERCENT.
        10  FILLER PIC X(15) VALUE "% CHANGE      ".
        10  WEEK-CHANGE PIC ++9.99.
        10  FILLER PIC X(5).
    05  PRINT-AREA-LAST-YEAR.
        10  FILLER PIC X(17) VALUE "LAST YEAR SALES ".
        10  LAST-YEAR-SALES PIC $$$,$$$.99.
        10  FILLER PIC XX.
    05  YEAR-PERCENT.
        10  FILLER PIC X(15) VALUE "% CHANGE      ".
        10  YEAR-CHANGE PIC ++9.99.
01  INQUIRY-PRINT-DATA.
    05  FILLER PIC X(6) VALUE " WEEK ".
    05  INQ-WK-NO PIC Z9.
    05  FILLER PIC X(6) VALUE " AREA ".
    05  INQ-AREA PIC 9.
    05  FILLER PIC X(17) VALUE "     $ INCREASE ".
    05  INQUIRY-PRINT-FIGURE PIC $$,.99.
```

```
01  LAST-WEEK-COMPARE.
    05  FILLER PIC 99.
    05  LW-AREA-TABLE OCCURS 7 TIMES INDEXED BY LWX.
        10  LW-AREA-NO PIC 9.
        10  LW-AREA-FIGURES PIC S99999V99.
01  ERROR-MSG.
    05  FILLER PIC X.
    05  ERROR-WEEK-NO PIC Z9.
    05  ERROR-TEXT PIC X(40) VALUE
        "EXCEEDS THIS-WEEK NO.  INQUIRY IGNORED.".
PROCEDURE DIVISION.
HOUSEKEEPING-ROUTINE.
    OPEN INPUT CURRENT-FILE LAST-YR-FILE INQUIRY-FILE,
         OUTPUT PRINT-FILE.
    READ LAST-YR-FILE AT END GO TO CLOSE-ROUTINE.
    READ CURRENT-FILE AT END GO TO CLOSE-ROUTINE.
    MOVE CURR-WEEK TO PRINT-WEEK.
    WRITE PRINT-RECORD FROM CURRENT-REPORT-PRINT
        AFTER POSITIONING 0.
    MOVE SPACES TO CURRENT-REPORT-PRINT.
    MOVE CURR-WEEK TO THIS-WEEK.
    SET WEEKT WEEKX WEEKY TO THIS-WEEK.
    IF THIS-WEEK EQUAL 1 GO TO FIRST-WEEK-ROUTINE.
LATER-WEEK-ROUTINE.
    OPEN I-O YR-TO-DATE-FILE.
    COMPUTE LAST-WEEK = THIS-WEEK - 1.
    READ YR-TO-DATE-FILE INTO YTD-WORK-TABLE
        AT END GO TO CLOSE-ROUTINE.
    MOVE WEEK-TABLE (WEEKX - 1) TO LAST-WEEK-COMPARE.
WEEK-CALCULATION-ROUTINE.
    MOVE CURRENT-SALE-RECORD TO WEEK-TABLE (WEEKT).
    SET LWX AREAT AREAY TO 1.
    PERFORM WEEK-PRINT THRU WEEK-PRINT-END 7 TIMES.
WRITE-FILE-ROUTINE.
    REWRITE YR-TO-DATE-RECORD FROM YTD-WORK-TABLE.
INQUIRY-FIND-ROUTINE.
    READ INQUIRY-FILE AT END GO TO CLOSE-ROUTINE.
    MOVE 1 TO ANY-INQ.
    IF INQUIRY-RECORD > THIS-WEEK MOVE INQUIRY-RECORD
        TO ERROR-WEEK-NO,
        WRITE PRINT-RECORD FROM ERROR-MSG AFTER POSITIONING 2,
        GO TO INQUIRY-FIND-ROUTINE.
    SET WEEKT WEEKY TO INQUIRY-RECORD.
    SET AREAT AREAY TO 1.
    MOVE INQUIRY-RECORD TO INQ-WK-NO.
    MOVE 0 TO INQ-COND-MET.
    PERFORM INQUIRY-PRINT THRU INQUIRY-PRINT-END UNTIL
        AREAT = 8.
    IF INQ-COND-MET EQUAL 0, MOVE INQUIRY-RECORD TO COND-MSG-WK,
        WRITE PRINT-RECORD FROM NONE-MET-COND
        AFTER POSITIONING 2.
    GO TO INQUIRY-FIND-ROUTINE.
CLOSE-ROUTINE.
    IF ANY-INQ EQUAL 0, WRITE PRINT-RECORD FROM NO-INQ-MSG
        AFTER POSITIONING 2.
    CLOSE CURRENT-FILE YR-TO-DATE-FILE LAST-YR-FILE
        INQUIRY-FILE PRINT-FILE.
END-ROUTINE.
    STOP RUN.
FIRST-WEEK-ROUTINE.
    OPEN OUTPUT YR-TO-DATE-FILE.
    SET WEEKZ TO 52.
    MOVE LAST-YR-WEEK-TABLE (WEEKZ) TO LAST-WEEK-COMPARE.
    PERFORM WEEK-CALCULATION-ROUTINE.
    WRITE YR-TO-DATE-RECORD FROM YTD-WORK-TABLE.
    GO TO INQUIRY-FIND-ROUTINE.
```

```
WEEK-PRINT.
    COMPUTE WEEK-CHANGE = (AREA-FIGURES (WEEKT AREAT) -
       LW-AREA-FIGURES (LWX)) / LW-AREA-FIGURES (LWX) *
       100.00
    COMPUTE YEAR-CHANGE = (AREA-FIGURES (WEEKT AREAT) -
       LAST-YR-AREA-FIGURES (WEEKY AREAY)) /
       LAST-YR-AREA-FIGURES (WEEKY AREAY) * 100.00.
    MOVE AREA-NO (WEEKT AREAT) TO PRINT-AREA-NO.
    MOVE AREA-FIGURES (WEEKT AREAT) TO PRINT-AREA-SALES.
    MOVE LW-AREA-FIGURES (LWX) TO LAST-WEEK-SALES.
    MOVE LAST-YR-AREA-FIGURES (WEEKY AREAY) TO LAST-YEAR-SALES.
    WRITE PRINT-RECORD FROM CURRENT-REPORT-PRINT
        AFTER POSITIONING 1.
    SET LWX AREAT AREAY UP BY 1.
WEEK-PRINT-END. EXIT.
INQUIRY-PRINT.
    SEARCH AREA-TOTALS VARYING AREAY
        AT END SET AREAT TO 8 GO TO INQUIRY-PRINT-END
        WHEN AREA-FIGURES (WEEKT AREAT) >
            LAST-YR-AREA-FIGURES (WEEKY AREAY)
            MOVE 1 TO INQ-COND-MET,
            COMPUTE INQUIRY-PRINT-FIGURE =
                AREA-FIGURES (WEEKT AREAT) -
                LAST-YR-AREA-FIGURES (WEEKY AREAY).
    MOVE AREA-NO (WEEKT AREAT) TO INQ-AREA.
    WRITE PRINT-RECORD FROM INQUIRY-PRINT-DATA
        AFTER POSITIONING 2.
    SET AREAT AREAY UP BY 1.
INQUIRY-PRINT-END.  EXIT.
```

SORT/MERGE

Arranging records in a particular order or sequence is a common requirement in data processing; such record ordering can be accomplished using sorting or merging operations. While both operations accomplish record ordering, the functions and capabilities of a sort and a merge are different.

A sort produces an ordered file from one or more files that may be completely unordered as to sort sequence. Thus, the sort operation must accept unordered sort input and produce ordered sort output. For this reason, all records must be available to the sort operation before sorting can begin. For this reason, too, there is always an absolute limit to the number of records that can be processed in one sorting operation.

A merge produces an ordered file from two or more input files, each of which is already ordered in the merge sequence. The merge operation is accomplished by record comparison as the records are encountered during the reading of all the input files. Only a portion of the records need be available to the merge operation at one time. Thus, many more records can be processed in one merge operation than in one sort.

COBOL has special language features which assist in sort and merge operations so that the user need not program these operations in detail. This chapter discusses the special COBOL sort and merge language.

SORT/MERGE CONCEPTS

Sorting and merging have always constituted a large percentage of the workload in business data processing. COBOL sort and merge language makes these operations easy to both specify and modify. COBOL language support is through the File-Contro' entry in the Environment Division, the SD (sort-merge-file-description) entry in the Data Division, and the SORT and MERGE statements in the Procedure Division.

The sort or merge file is described through the File-Control SELECT sentence, and the SD entry in the Data Division. The sort or merge file is the working file used during the sort or merge; it can be considered an internal file. As such, blocking and internal storage allocation for this file are not under the control of the COBOL programmer. However, a sort or merge file, like any file, is a set of records, and a sort-merge file description can be considered a particular type of file description.

The sort-merge file is processed through a Procedure Division SORT or MERGE statement. The statement specifies the key field(s) within the record upon which the sort or merge is to be arranged. Keys can be specified as ascending or descending, or, when more than one key is specified, as a mixture of the two. The sequence of sorted or merged records conforms to the specified mixture of keys.

SORT CONCEPTS

Through the SORT statement, the COBOL user has access to input procedures (used before sorting) and output procedures (used after sorting) that can add, delete, alter, edit, or otherwise modify the records in the input and/or output files. Within the limits of virtual storage, a COBOL program can contain any number of sorts, each with its own independent input and/or output procedures. During SORT statement execution, these procedures are automatically executed at the specified point in processing; thus, extra passes through the sort file are avoided.

A COBOL program containing a sort is usually organized so that one or more input files is read and operated on by an input procedure. Within the input procedure a RELEASE statement (analogous to the WRITE statement) places a record in the sort file. That is, when input procedure execution is completed, a sort file has been created by placing records one at a time into the sort file through the RELEASE statement. If the user does not want to modify the records before the sorting operation begins, the SORT statement USING option releases the unmodified records to the sort file.

After all the input records have been placed in the sort file, the sorting operation is executed. This operation arranges the entire set of sort file records in the sequence specified by the key(s).

After completion of the sorting operation, sorted records can be made available from the sort file, one at a time, through a RETURN statement for modification in an output procedure. If the user does not want to modify the sorted records, the SORT statement GIVING option names the sorted output file.

MERGE CONCEPTS

Through the MERGE statement, the COBOL user has access to output procedures (used after merging) that can modify the records in the output file. The COBOL program can contain any number of merge operations, each with its own independent output procedures. During MERGE statement execution, these procedures are automatically executed at the specified point in processing.

MERGE statement execution begins the merge processing. This operation compares keys within the records of the input files, and arranges the records within the merged file in the sequence specified by the key(s).

Merged records can be made available, one at a time, through a RETURN statement for modification in an output procedure. If the user does not want to modify the merged records, the MERGE statement GIVING option names the merged output file.

Note: For ASCII considerations, see Appendix B.

SORT/MERGE—ENVIRONMENT DIVISION

In the Environment Division, the programmer must write File-Control entries for each file used as input to or output from a sort or merge operation. A File-Control entry for the sort-file or merge-file itself must also be written.

IBM Extension

If checkpoint records are to be written during the operation, an I-O-Control entry containing a RERUN clause may also be specified.

End of IBM Extension

FILE-CONTROL ENTRY

For the input and output files of a sort or merge operation, the File-Control entry for sequential files as described in the Environment Division chapter is valid.

IBM Extension

However, this IBM implementation allows the input and output files to be any file with sequential access, and the File-Control entry for any sequentially accessed file is valid.

End of IBM Extension

For the sort or merge file, the following format shows the allowable clauses in the File-Control Entry.

Format—Sort/Merge File

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

Each sort or merge file described in the Data Division must be named once, and only once, as a file-name in a File-Control entry.

When file-name specifies a sort or a merge file, only the ASSIGN clause may follow the SELECT clause in this File-Control entry.

The ASSIGN clause associates a sequential sort or merge file with a storage medium.

Assignment-name has the same rules of formation as it has for other files. See the System Dependencies chapter for further information.

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph for a sort or merge file may be specified as shown in the following format.

Format—Sort or Merge Files

```
┌──────────────────────────────┐
│ [RERUN ON assignment-name]   │
└──────────────────────────────┘

        ┌              ┐
        │ RECORD       │
[SAME   < SORT         >  AREA
        │ SORT-MERGE   │
        └              ┘
    FOR file-name-1 [file-name-2] ... ].
```

IBM Extension

The order in which the I-O-CONTROL paragraph clauses are written is not significant.

End of IBM Extension

RERUN Clause

IBM Extension

When the RERUN cause is specified in the I-O control paragraph, checkpoint records are written at logical intervals determined by the sort/merge program during execution of each SORT or MERGE statement in the program. When it is omitted, checkpoint records are not written.

Assignment-name must not be the same as any used in an ASSIGN clause, but must follow the same rules of formation. The System Dependencies chapter gives more information.

When the checkpoint record is written, the status of all open files, whether involved in the sort/merge operation or not, is recorded.

End of IBM Extension

SAME SORT/SORT-MERGE AREA Clause

The function of the SAME SORT AREA or SORT-MERGE AREA clause is to optimize storage area assignment to a given SORT or MERGE statement. The system handles storage assignment automatically; therefore, the SORT AREA or SORT-MERGE AREA options, when specified, are treated as documentation.

The SAME SORT AREA or SORT-MERGE AREA clause specifies one storage area available for sort/merge operations by each named sort or merge file. That is, the storage allocated for one such operation is available for reuse in another.

When the SAME SORT AREA or SAME SORT-MERGE AREA clause is specified, at least one specified file-name must name a sort or merge file. Files that are not sort or merge files may also be specified. The following rules apply:

- More than one SAME SORT AREA or SORT-MERGE AREA clause may be specified; however, a sort or merge file must not be named in more than one such clause.
- If a file that is not a sort or merge file is named in both a SAME AREA clause and in one or more SAME SORT AREA or SORT-MERGE AREA clauses, all of the files in the SAME AREA clause must also appear in that SAME SORT AREA or SORT-MERGE AREA clause.
- Files named in a SAME SORT AREA or SORT-MERGE AREA clause need not have the same organization or access.
- Files named in a SAME SORT AREA or SORT-MERGE AREA clause that are not sort or merge files do not share storage with each other unless the user names them in a SAME AREA or SAME RECORD AREA clause.

Rules for the specification of SAME RECORD AREA clause are given in the Environment Division chapter.

SORT/MERGE—DATA DIVISION

In the File Section, the programmer must write an FD entry for each file that is input to or output from the sort/merge operation, and a record description entry.

In addition, there must be an SD (sort-merge-file-description) entry for each sort or merge file, as well as a record description, for each sort or merge file in the program. The SD entry has the following format.

Format—SD Entry

```
SD  file-name

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

           ┌ RECORD IS   ┐
   [DATA  <               >  data-name-1 [data-name-2] ... ].
           │ RECORDS ARE │
           └             ┘
```

The level indicator, SD, identifies the beginning of the SD entry, and must precede the file-name.

The file-name must specify a sort or merge file.

The clauses that follow file-name are optional, and their order of appearance is not significant.

Both the RECORD CONTAINS clause and the DATA RECORDS clause are described in the chapter on the Data Division.

One or more record description entries must follow the SD entry. Note, however, that no input/output statements may be executed for this file.

See the System Dependencies chapter for further considerations.

The following example illustrates the File Section entries needed for a sort or merge file:

```
SD  SORT-FILE.

01  SORT-RECORD  PICTURE X(80).
```

SORT/MERGE—PROCEDURE DIVISION

The Procedure Division contains MERGE and SORT statements to describe the merge and sort operations and, optionally, sort input procedures and/or sort/merge output procedures. A sort input procedure must contain a RELEASE statement that makes each record available to the sorting operation. A sort/merge output procedure must contain a RETURN statement that makes a sorted/merged record available to the output procedure.

The Procedure Division may contain more than one SORT and/or MERGE statements appearing anywhere except in the Declaratives portion, or in the sort input or sort/merge output procedures.

Files specified in the USING and GIVING options of the SORT and MERGE statements must be described explicitly or implicitly in their File-Control entries as having sequential organization.

IBM Extension

However, this IBM implementation also allows the input and output files to be any VSAM file with sequential access.

End of IBM Extension

Only one file-name from a multiple file reel can appear in a SORT or MERGE statement.

Several SORT/MERGE statement options apply to both the SORT and the MERGE statements. These options are discussed after the descriptions of the MERGE and the SORT statements.

MERGE STATEMENT

The MERGE statement combines two or more identically sequenced files (that is, files that have already been sorted according to an identical set of ascending/descending keys) on one or more keys and makes records available in merged order to an ouput procedure or output file.

Format

```
MERGE file-name-1

      ┌ ASCENDING  ┐
   ON <            >  KEY data-name-1 [data-name-2] ...
      | DESCENDING |
      └            ┘

     ┌ ASCENDING  ┐
[ON  <            >  KEY data-name-3 [data-name-4] ... ] ...
     | DESCENDING |
     └            ┘

[COLLATING SEQUENCE IS alphabet-name]

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

┌                                                            ┐
| GIVING file-name-5                                         |
| OUTPUT PROCEDURE                                           |
<                                                            >
|                                 ┌ THROUGH ┐                |
|       IS section-name-1 [ <         >  section-name-2]     |
└                                 | THRU    |                ┘
                                  └         ┘
```

File-name-1 is the name given in the SD entry which describes the record. The maximum number of files that can be merged is eight.

No file-name may be repeated in the MERGE statement.

When the MERGE statement is executed, all records contained on file-name-2, file-name-3, and so forth, are accepted by the sort/merge program and then merged according to the key(s) specified. These files must not be open when the MERGE statement is executed; they are automatically opened and closed by the MERGE operation, with all implicit functions (such as execution of any associated USE procedures) performed. The files are closed as if the CLOSE statement is written with no optional processing.

Rules for the ASCENDING/DESCENDING KEY option, the COLLATING SEQUENCE option, the USING and GIVING options, and the sort/merge OUTPUT PROCEDURE option are given in the following section on SORT/MERGE Statement Options.

SORT STATEMENT

The SORT statement accepts records from one or more files, sorts them according to specified key(s), and makes records available either through an output procedure or in an output file.

Format

```
SORT file-name-1

        ┌ ASCENDING  ┐
   ON   <            >  KEY data-name-1 [data-name-2] ...
        | DESCENDING |
        └            ┘


        ┌ ASCENDING  ┐
  [ON   <            >  KEY data-name-3 [data-name-4] ... ] ...
        | DESCENDING |
        └            ┘

[COLLATING SEQUENCE IS alphabet-name]

┌                                                       ┐
| USING file-name-2 [file-name-3] ...                   |
| INPUT PROCEDURE                                       |
|                            ┌ THROUGH ┐                |
<                            |         |                >
|    IS section-name-1 [     <         >  section-name-2] |
|                            | THRU    |                |
└                            └         ┘                ┘

┌                                                       ┐
| GIVING file-name-4                                    |
| OUTPUT PROCEDURE                                      |
<                                                       >
|                            ┌ THROUGH ┐                |
|    IS section-name-3 [     <         > section-name-4] |
|                            | THRU    |                |
└                            └         ┘                ┘
```

File-name-1 is the name given in the SD entry which describes the records being sorted.

When the SORT statement is executed, all records contained on file-name-2, file-name-3, and so forth, are accepted by the sort/merge program, and then sorted according to the key(s) specified. These input files must not be open at the time the SORT statement is executed; they are automatically opened and closed by the SORT operation, with all implicit functions (such as execution of any associated USE procedures) performed. The files are closed as if the CLOSE statement is written with no optional processing.

Rules for the ASCENDING/DESCENDING KEY option, the COLLATING SEQUENCE option, the USING and GIVING options, the sort INPUT PROCEDURE option, and the sort/merge OUTPUT PROCEDURE option are given in the following section on SORT/MERGE Statement Options.

SORT/MERGE STATEMENT OPTIONS

The SORT/MERGE statement options are: the ASCENDING/DESCENDING KEY option, the COLLATING SEQUENCE option, the USING option, the GIVING option, and the OUTPUT PROCEDURE option. In addition, the SORT statement uses the INPUT PROCEDURE option. All are discussed in the following sections.

ASCENDING/DESCENDING KEY Option

This option specifies that records are to be processed in an ascending or descending sequence (depending on the option specified) based on the specified sort/merge keys.

Each data-name specifies a KEY data item on which the sort-merge will be based. Each such data-name must identify a data item in a record associated with file-name-1. The following rules apply:

- A specific KEY data item must be physically located in the same position and have the same data format in each input file; however, it need not have the same data-name.

- If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.

- KEY data items must be fixed-length items.

- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.

- A maximum of 12 KEY data items may be specified.

- The total length of all KEY data items must not exceed 256 bytes.

- All KEY data items must be located within the first 4092 bytes of the record.

- KEY data items may be qualified; they may not be subscripted or indexed.

The KEY data items are listed in order of decreasing significance, no matter how they are divided into KEY phrases. Using the format as an example, data-name-1 is the most significant key and records are processed in ascending or descending order on that key; data-name-2 is the next most significant key and within data-name-1 records are processed on data-name-2 in ascending or descending order. Within data-name-2, records are processed on data-name-3 in ascending or descending order; within data-name-3, records are processed on data-name-4 in ascending or descending order.

The direction of the sort/merge operation depends on the specification of the ASCENDING or DESCENDING key words as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.

- When DESCENDING is specified, the sequence is from the highest key value to the lowest.

- If the KEY data item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, the sequence of key values depends on the collating sequence used (see the following COLLATING SEQUENCE Option description).

- The key comparisons are performed according to the rules for comparison of operands in a relation condition (see the Relation Condition description in the Conditional Expressions chapter).

COLLATING SEQUENCE Option

This option specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sort/merge operation.

Alphabet-name must be specified in the SPECIAL-NAMES paragraph, alphabet-name clause. Any one of the alphabet-name clause options may be specified, with the following results:

- When NATIVE is specified, the EBCDIC collating sequence is used for all nonnumeric comparisons. (The EBCDIC collating sequence is given in Appendix G.)

- When STANDARD-1 is specified, the ASCII collating sequence is used for all nonnumeric comparisons. (The ASCII collating sequence is given in Appendix G.)

- When the literal option is specified, the collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When the COLLATING SEQUENCE option is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used. When both the COLLATING SEQUENCE option and the PROGRAM COLLATING SEQUENCE clause are omitted, the EBCDIC collating sequence is used.

USING Option

When the USING option is specified, all the records on file-name-2, file-name-3, and so forth, (that is, the input files), are transferred automatically to file-name-1. At the time the SORT or MERGE statement is executed, these files must not be open; the compiler opens, reads, makes records available, and closes these files automatically. If EXCEPTION/ERROR procedures are specified for these files, the compiler makes the necessary linkage to these procedures. Up to eight files may be specified.

The input files must have sequential organization.

IBM Extension

However, this IBM implementation also allows these files to be VSAM files with sequential access.

End of IBM Extension

All input files must be described in an FD entry in the Data Division, and their record descriptions must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the input records as having an equal number of character positions as the sort record.

GIVING Option

When the GIVING option is specified, all the sorted or merged records in file-name-1 are automatically transferred to the output file (MERGE file-name-5 or SORT file-name-4). At the time the SORT or MERGE statement is executed, this file must not be open; the compiler opens, reads, makes records available, and closes the output file automatically. If EXCEPTION/ERROR procedures are specified for the output file, the compiler makes the necessary linkage to these procedures.

The output file must have sequential organization.

IBM Extension

However, this IBM implementation also allows this file to be a VSAM file with sequential access.

End of IBM Extension

The output file must be described in an FD entry in the Data Division, and its record description(s) must describe records of the same size as the record described for the sort file. If the elementary items that make up these records are not identical, the user must describe the output record as having an equal number of character positions as the sort or merge record.

SORT INPUT PROCEDURE Option

This option specifies the section-name(s) of a procedure that is to modify input records before the sorting operation begins.

Section-name-1 specifies the first (or only) section in the input procedure. Section-name-2, when specified, identifies the last section of the input procedure.

The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file.

Control must not be passed to the input procedure except when a related SORT statement is being executed, because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. There are three restrictions on the procedural statements within an input procedure:

1. The input procedure must not contain any SORT or MERGE statements.

2. The input procedure must not contain any transfers of control to points outside the input procedure. The execution of a CALL statement to another program that follows standard linkage conventions, or the execution of USE declaratives are not considered transfers of control outside an input procedure. Because of this, they are allowed to be activated within these procedures.

IBM Extension

However, the compiler permits the ALTER, GO TO, and PERFORM statements in the input procedure to refer to procedure-names outside the input procedure. However, it is the user's responsibility to ensure a return to the input procedure after exiting through a GO TO or a PERFORM statement.

End of IBM Extension

3. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure (with the exception of the return of control from a Declaratives Section).

If an input procedure is specified, control is passed to the input procedure when the SORT program input phase is ready to receive the first record. The compiler inserts a return mechanism at the end of the last section of 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. The RELEASE statement transfers records from the Input Procedure to the sort file, which is then used in the input phase of the sort operation (see "RELEASE Statement").

SORT/MERGE OUTPUT PROCEDURE Option

This option specifies the section-name(s) of a procedure that is to modify output records from the sort or merge operation.

Section-name-1 specifies the first (or only) section in the output procedure. Section-name-2, when specified, identifies the last section of the output procedure.

The output procedure must consist of one or more sections that are written consecutively and do not form a part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted/merged records available for processing.

When all the records are sorted/merged, control is passed to the output procedure. The RETURN statement in the output procedure is a request for the next record (see "RETURN Statement").

Control must not be passed to the output procedure except when a related SORT or MERGE statement is being executed, because RETURN statements in the output procedure have no meaning unless they are controlled by a SORT or MERGE statement. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time, from the sort- or merge-file.

There are three restrictions on the procedural statements within the output procedure.

1. The output procedure must not contain any SORT or MERGE statements.

2. The output procedure must not contain any transfers of control to points outside the output procedure. The execution of a CALL statement to another program that follows standard linkage conventions, or the execution of USE declaratives is not considered a transfer of control outside an output procedure. Because of this, they are allowed to be activated within these procedures.

IBM Extension

However, the compiler permits the ALTER, GO TO, and PERFORM statements in the output procedure to refer to procedure-names outside the output procedure. However, it is the user's responsibility to ensure a return to the output procedure after exiting through a GO TO or a PERFORM statement.

End of IBM Extension

3. The remainder of the Procedure Division must not contain any transfers of control to points inside the procedure (with the exception of the return of control from a Declaratives Section).

When an output procedure is specified, control passes to it after the sort/merge file (file-name-1) has been placed in sequence by the sort/merge operation. 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 terminates the sort or merge, and passes control to the next executable statement after the SORT or MERGE statement.

INPUT/OUTPUT PROCEDURE Control

The INPUT or OUTPUT PROCEDURE options function in a manner similar to Option 1 of the PERFORM statement (the simple PERFORM). For example, naming a section in an OUTPUT PROCEDURE option causes execution of that section during the sort/merge operation to proceed as if that section were named in a PERFORM statement. As with the PERFORM statement, execution of the section is terminated after execution of its last statement. The last statement in Input and Output Procedures can be the EXIT statement (see "EXIT Statement"); this is useful for documentation purposes.

RELEASE STATEMENT (SORT STATEMENT ONLY)

The RELEASE statement transfers records from an input/output area to the initial phase of a sort operation.

Format

<u>RELEASE</u> record-name [<u>FROM</u> identifier]

The RELEASE statement may be specified only within an Input Procedure associated with a SORT statement. Within an Input Procedure at least one RELEASE statement must be specified.

When the RELEASE statement is executed, the current contents of record-name are placed in the sort file; that is, made available to the initial phase of the sort operation.

Record-name must specify a record associated with the SD entry for file-name-1. Record-name may be qualified.

When the FROM identifier option is specified, the RELEASE statement is equivalent to the statement MOVE identifier to record-name, followed by the statement RELEASE record-name. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING option.

Identifier and record-name must not refer to the same storage area.

After the RELEASE statement is executed, the information in record-name is no longer available, unless file-name-1 is specified in a SAME RECORD AREA clause, in which case record-name is still available as a record of the other files named in that clause. When the FROM identifier option is specified, the information is still available in identifier.

When control passes from the Input Procedure, the sort file consists of all those records placed in it by execution of RELEASE statements.

RETURN STATEMENT

The RETURN statement transfers records from the final phase of a sort or merge operation to an input/output area.

Format

```
RETURN file-name RECORD [INTO identifier]

     AT END imperative-statement
```

The RETURN statement may be specified only within an Output Procedure associated with a SORT or MERGE statement. Within an Output Procedure at least one RETURN statement must be specified.

When the RETURN statement is executed, the next record from file-name is made available for processing by the Output Procedure.

File-name must be described in a Data Division SD entry.

If more than one record description is associated with file-name, these records automatically share the same storage; that is, the area is implicitly redefined. After RETURN statement execution, only the contents of the current record are available; if any data items lie beyond the length of the current record, their contents are undefined.

When the INTO identifier option is specified, the RETURN statement is equivalent to the statement RETURN file-name, followed by the statement MOVE record-name TO identifier. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING option. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to identifier.

The INTO identifier option must not be specified if the sort or merge file contains records of varying sizes.

The record areas associated with file-name and identifier must not be the same storage area.

After all records have been returned from file-name, the AT END imperative-statement is executed, and no more RETURN statements may be executed.

IBM Extension

SORT/MERGE SPECIAL REGISTERS

Four special registers are available to Sort/Merge Feature users. These special registers provide object-time communication between the user and the Sort/Merge program; they aid performance optimization.

SORT-RETURN

SORT-RETURN is available to both Sort and Merge programs.

SORT-RETURN is the name of a binary data item whose PICTURE is S9(4). It contains a return code of 0 or 16 at the completion of a sorting operation to signify the success or failure, respectively, of the sort operation. If the sort operation abnormally terminates and there is no reference to this special register anywhere in the program, a message is displayed on the console. Then, the operator may continue or cancel the job.

SORT-RETURN must not be specified as an operand in the ACCEPT or DISPLAY statements.

(See also the chapter on System Dependencies.)

SORT-FILE-SIZE, SORT-MODE-SIZE, SORT-CORE-SIZE

SORT-FILE-SIZE, SORT-MODE-SIZE, and SORT-CORE-SIZE are available only to the Sort operation. These registers can have control information transferred into them at object time as the receiving field of statements such as MOVE. (They must not be operands in an ACCEPT or DISPLAY statement.) The control information must be transferred before the SORT statement is executed.

These registers are initialized to 0 by the compiler, but are not reset after a sort operation is completed. Thus, in a program with more than one SORT statement, any values in these registers at the end of one sorting operation will still be present at the beginning of the next, unless they are modified by the programmer.

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. If SORT-FILE-SIZE is omitted, the Sort Feature assumes that the file contains the maximum number of records that can be processed with the available storage size and number of work units. If the estimate exceeds the maximum, the estimate will be ignored.

SORT-MODE-SIZE is the name of a binary data item whose PICTURE is S9(5). 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. If SORT-MODE-SIZE is omitted, the average of the maximum and minimum record lengths is assumed. For example, if records vary in length from 20 to 100 bytes, but most records are 30 bytes long, the value 30 should be moved to SORT-MODE-SIZE.

SORT-CORE-SIZE is the name of a binary data item whose PICTURE is S9(8). It is used to specify the number of bytes of storage available to the sorting operation if it is different from the storage size that the Sort Feature would normally use.

See also the chapter on System Dependencies.

End of IBM Extension

PROGRAMMING NOTES

Extreme caution should be used when sorting variable-length records with embedded objects of the OCCURS DEPENDING ON clause. The following considerations apply:

1. The USING and GIVING options, since they preclude presetting of the OCCURS DEPENDING ON object in the receiving record, will probably lead to incorrect results.

2. Therefore, Input and Output Procedures should be specified.

   In the Input Procedure, before the RELEASE statement is executed, the user should move the OCCURS DEPENDING ON object to the sort file record; this presets the sort file record length. The RELEASE statement can then be correctly executed.

   In the Output Procedure, the RETURN statement must not use the INTO option. After the RETURN statement is executed, and before any processing operation, such as a WRITE statement, is issued, the object of the OCCURS DEPENDING ON clause should be moved to the proper receiving field; this presets the receiving record length before the processing operation takes place.

The limit on the number of records that can be processed in one sort operation is dependent on the amount of storage available and the size and number of the sort work files.

IBM Extension

The sort special registers are especially useful in optimizing a sorting operation to the specific number or size of records expected to be processed during one execution of the program, because values can be changed from one execution to the next.

End of IBM Extension

If extremely large files must be ordered, the program can be written to sort segments of the file into several sorted files, and then merge the sorted files into one output file.

IBM Extension

The RERUN statement is especially useful when sort/merge programs that order extremely large files are written. If for any reason the program must be interrupted, the RERUN clause allows the sort or merge to be restarted at the last checkpoint.

End of IBM Extension

SORT/MERGE SAMPLE PROGRAM

This program provides a sorted record, for the current month, of net sales within each on-site department of a manufacturing company. It also provides a year-to-date sorted record of net sales by all employees (on-site and off-site) for the current year.

The records for the current month are read in unsorted order.

First, the SORT statement for the current-month report is executed. The INPUT PROCEDURE for this sorting operation is SCREEN-DEPT SECTION, which removes the records for employees in departments 7 and 9 (off-site employees) before releasing the records for the sorting operation. Records for all on-site employees are then released to the first sorting operation. The records are then sorted in ascending order of department, and, within each department, in descending order of net sales. The output for this sort is then printed.

Next, the yearly report is updated. In this report, off-site employees are included (that is, the USING option of the SORT statement is specified, not the INPUT PROCEDURE option). For the yearly report, the current month's records are sorted in ascending order of department, and, within each department, in ascending order of employee number.

After the second sorting operation is completed, the current month's records are merged with the existing year-to-date records. The records in this file are merged in ascending order of department number, and, within each department, in ascending order of employee numbers, and, for each employee, in ascending order of months.

When the MERGE statement execution is complete, an updated year-to-date master file has been created.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORT-IT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370-163.
OBJECT-COMPUTER. IBM-370-163.
SPECIAL-NAMES. SYSLST IS PRINTER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT NET-FILE-IN ASSIGN TO SYS002-DA-3330-S-SORTIN.
    SELECT NET-FILE-OUT ASSIGN TO SYS003-DA-3330-S-SORTOUT.
    SELECT NET-FILE-WORK ASSIGN TO SYS001-DA-3330-S-SORTWK1.
    SELECT YTD-NET-FILE-IN ASSIGN TO SYS002-DA-3330-S-SORTIN.
    SELECT YTD-NET-FILE-MASTER
         ASSIGN TO SYS004-DA-3330-S-YTDMSTR.
DATA DIVISION.
FILE SECTION.
SD  NET-FILE-WORK
    DATA RECORD IS SALES-RECORD.
01  SALES-RECORD.
    05  EMPL-NO                  PICTURE 9(6).
    05  DEPT                     PICTURE 9(2).
    05  NET-SALES                PICTURE 9(7)V99.
    05  NAME-ADDR                PICTURE X(61).
    05  MONTH                    PICTURE X(2).
FD  NET-FILE-IN
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS NET-CARD-IN.
01  NET-CARD-IN.
    05  EMPL-NO-IN               PICTURE 9(6).
    05  DEPT-IN                  PICTURE 9(2).
        88  OFF-SITE-LOCATION VALUES ARE 7, 9.
    05  NET-SALES-IN             PICTURE 9(7)V99.
    05  NAME-ADDR-IN             PICTURE X(61).
    05  MONTH-IN                 PICTURE X(2).
FD  NET-FILE-OUT
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS NET-CARD-OUT.
01  NET-CARD-OUT.
    05  EMPL-NO-OUT              PICTURE 9(6).
    05  DEPT-OUT                 PICTURE 9(2).
    05  NET-SALES-OUT            PICTURE 9(7)V99.
    05  NAME-ADDR-OUT            PICTURE X(61).
    05  MONTH-OUT                PICTURE X(2).
FD  YTD-NET-FILE-IN
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS YTD-NET-CARD-IN.
01  YTD-NET-CARD-IN.
    05  YTD-EMPL-NO              PICTURE 9(6).
    05  YTD-DEPT                 PICTURE 9(2).
    05  YTD-NET-SALES            PICTURE 9(7)V99.
    05  YTD-NAME-ADDR            PICTURE X(61).
    05  YTD-MONTH                PICTURE X(2).
FD  YTD-NET-FILE-MASTER
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS YTD-NET-CARD-MASTER.
01  YTD-NET-CARD-MASTER.
    05  EMPL-NO-MASTER           PICTURE 9(6).
    05  DEPT-MASTER-MASTER       PICTURE 9(2).
    05  NET-SALES-MASTER-MASTER  PICTURE 9(7)V99.
    05  NAME-ADDR-MASTER-MASTER  PICTURE X(61).
    05  MONTH-MASTER-MASTER      PICTURE X(2).
PROCEDURE DIVISION.
ELIM-DEPT-OFF-SITE-NO-PRINTOUT.
    SORT NET-FILE-WORK
        ASCENDING KEY DEPT
        DESCENDING KEY NET-SALES
        INPUT PROCEDURE SCREEN-DEPT
        GIVING NET-FILE-OUT.
CHECK-RESULTS SECTION.
C-R-1.
    OPEN INPUT NET-FILE-OUT.
C-R-2.
    READ NET-FILE-OUT AT END GO TO C-R-FINAL.
    DISPLAY EMPL-NO-OUT DEPT-OUT NET-SALES-OUT
        NAME-ADDR-OUT UPON PRINTER.
C-R-3.
    GO TO C-R-2.
C-R-FINAL.
    CLOSE NET-FILE-OUT.
UPDATE-YEARLY-REPORT SECTION.
U-Y-R-1.
    SORT NET-FILE-WORK
        ASCENDING KEY DEPT
        ASCENDING KEY EMPL-NO
        USING NET-FILE-IN
        GIVING NET-FILE-OUT.
U-Y-R-2.
    MERGE NET-FILE-WORK
        ASCENDING KEY DEPT
        ASCENDING KEY EMPL-NO
        ASCENDING KEY MONTH
        USING YTD-NET-FILE-IN, NET-FILE-OUT
        GIVING YTD-NET-FILE-MASTER.
    STOP RUN.
SCREEN-DEPT SECTION.
S-D-1.
    OPEN INPUT NET-FILE-IN.
S-D-2.
    READ NET-FILE-IN AT END GO TO S-D-FINAL.
    DISPLAY EMPL-NO-IN DEPT-IN NET-SALES-IN
        NAME-ADDR-IN UPON PRINTER.
S-D-3.
    IF OFF-SITE-LOCATION GO TO S-D-2
        ELSE
            MOVE NET-CARD-IN TO SALES-RECORD
            RELEASE SALES-RECORD
            GO TO S-D-2.
S-D-FINAL.
    CLOSE NET-FILE-IN.
S-D-END.
    EXIT.
```

REPORT WRITER (IBM EXTENSION)

IBM Extension

The IBM Report Writer feature is designed according to the specifications of American National Standard COBOL, X3.23-1968. Thus, it can be considered an extension to American National Standard COBOL, X3.23-1974; that is, use of this feature does not guarantee results as defined in the 1974 standard.

The Report Writer allows the COBOL programmer to produce a report by specifying the physical appearance of the report, rather than by specifying the detailed procedures necessary to produce that report.

REPORT WRITER CONCEPTS

The Report Writer feature simplifies the programming needed to produce an output report in comparison with using general-purpose COBOL statements to produce the same report.

Using general-purpose COBOL, the programmer must write detailed procedures for determining the interrelationship of output lines, for recognizing page overflow, for constructing headings and footings, for recognizing ends of logical data groupings, for formatting output lines, and for moving and performing arithmetic on the data itself. Such programs are invariably large and complex, and usually require a large amount of debugging.

Using Report Writer, the programmer simply describes how the report is to look, and what items are to control report logic. The Report Writer feature then sets up the subroutines needed to produce the report in the requested format.

The report is described physically and logically in the Report Section of the Data Division; the few statements necessary to cause the object program to perform the Report Writer subroutines are written in the Procedure Division. The following paragraphs give an overall description of Report Writer logic.

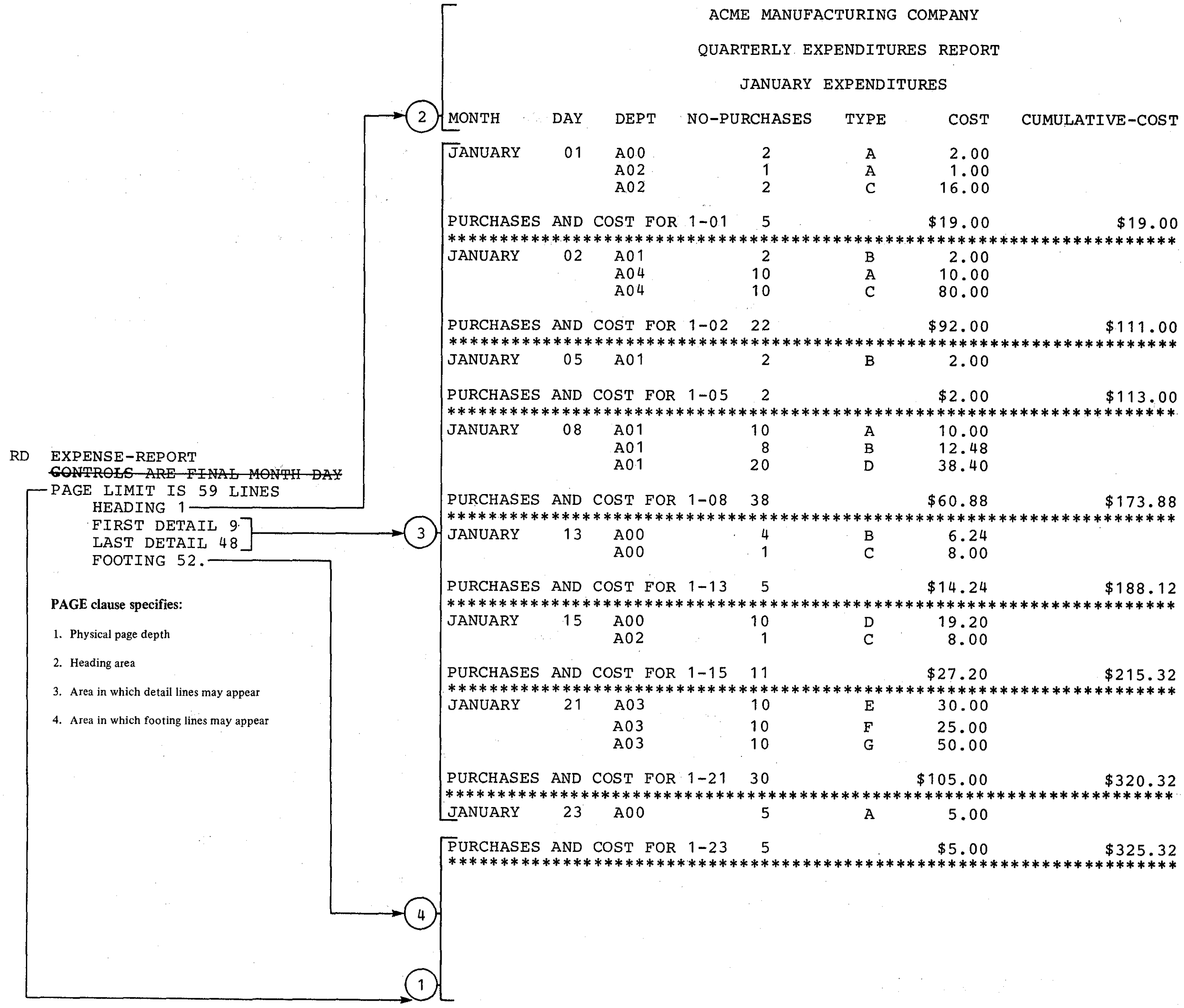
DATA DIVISION CONCEPTS

In the Report Section of the Data Division, the programmer completely describes the physical and logical aspects of each report to be produced, through a report description (RD) entry, and through associated report group description entries.

RD Entry

The RD entry is comparable to the FD entry for a file. This entry names the report, and allows optional specification of other clauses.

THE PAGE CLAUSE: allows the programmer, if page-by-page report organization is required, to set up page parameters that specify the total number of lines on each physical page, as well as line limits within which each type (heading, detail, footing) of report information can be printed. Figure 65 illustrates PAGE clause concepts.



```
RD   EXPENSE-REPORT
     ~~CONTROLS ARE FINAL MONTH DAY~~
     PAGE LIMIT IS 59 LINES
        HEADING 1
        FIRST DETAIL 9
        LAST DETAIL 48
        FOOTING 52.
```

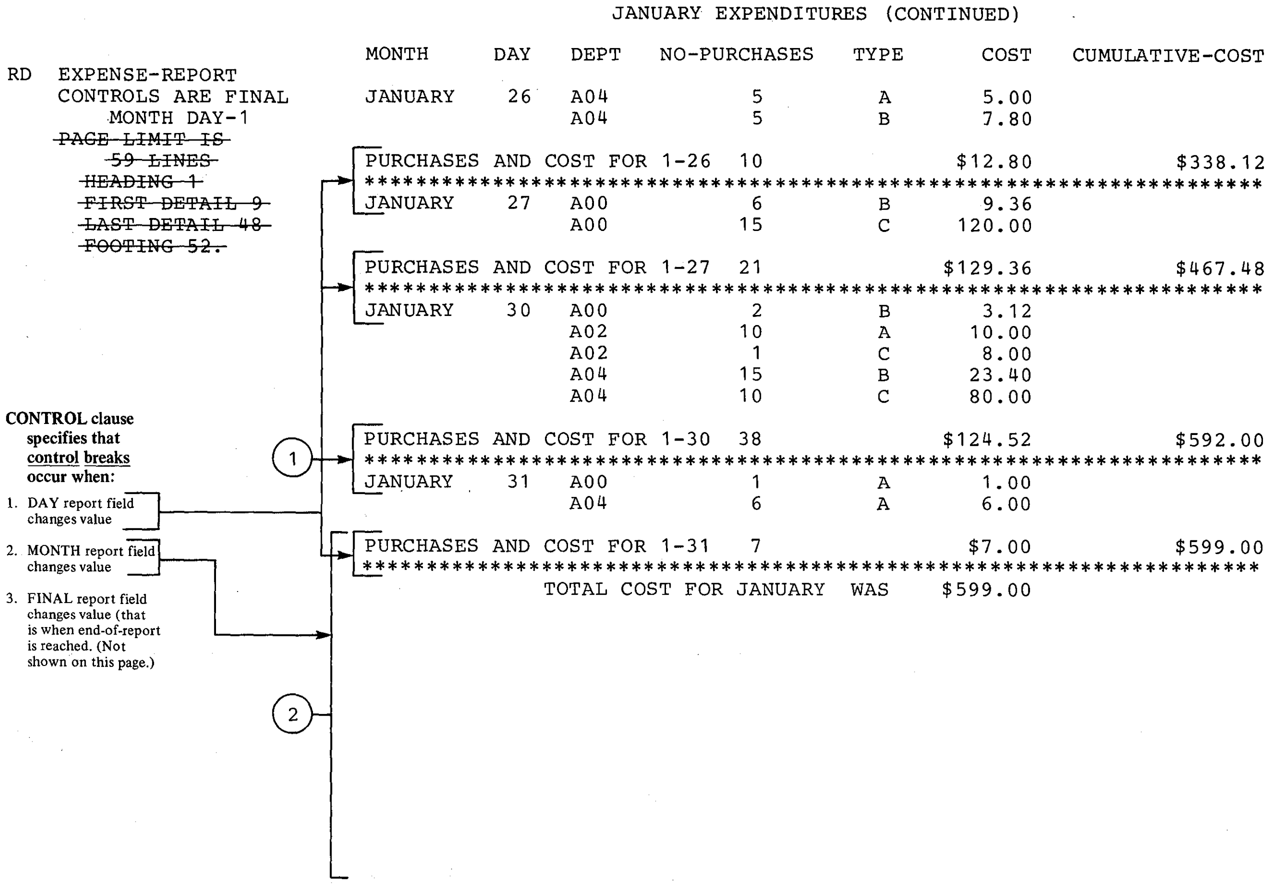
```
                              ACME MANUFACTURING COMPANY

                             QUARTERLY EXPENDITURES REPORT

                                 JANUARY EXPENDITURES

MONTH     DAY   DEPT   NO-PURCHASES    TYPE      COST    CUMULATIVE-COST

JANUARY    01   A00          2          A        2.00
                A02          1          A        1.00
                A02          2          C       16.00

PURCHASES AND COST FOR 1-01  5                  $19.00            $19.00
**********************************************************************
JANUARY    02   A01          2          B        2.00
                A04         10          A       10.00
                A04         10          C       80.00

PURCHASES AND COST FOR 1-02 22                  $92.00           $111.00
**********************************************************************
JANUARY    05   A01          2          B        2.00

PURCHASES AND COST FOR 1-05  2                   $2.00           $113.00
**********************************************************************
JANUARY    08   A01         10          A       10.00
                A01          8          B       12.48
                A01         20          D       38.40

PURCHASES AND COST FOR 1-08 38                  $60.88           $173.88
**********************************************************************
JANUARY    13   A00          4          B        6.24
                A00          1          C        8.00

PURCHASES AND COST FOR 1-13  5                  $14.24           $188.12
**********************************************************************
JANUARY    15   A00         10          D       19.20
                A02          1          C        8.00

PURCHASES AND COST FOR 1-15 11                  $27.20           $215.32
**********************************************************************
JANUARY    21   A03         10          E       30.00
                A03         10          F       25.00
                A03         10          G       50.00

PURCHASES AND COST FOR 1-21 30                 $105.00           $320.32
**********************************************************************
JANUARY    23   A00          5          A        5.00

PURCHASES AND COST FOR 1-23  5                   $5.00           $325.32
**********************************************************************
```

PAGE clause specifies:

1. Physical page depth
2. Heading area
3. Area in which detail lines may appear
4. Area in which footing lines may appear

Figure 65. RD Entry—PAGE Clause Concepts

THE CONTROL CLAUSE: specifies data items whose values control the report logic. For example, a date field might be a control item; a change in the date could cause extra spacing to be inserted in the report. When such a field changes value, a control break is said to occur. Figure 66 illustrates CONTROL clause concepts.



```
RD  EXPENSE-REPORT
    CONTROLS ARE FINAL
        MONTH DAY-1
    ~~PAGE LIMIT IS~~
        ~~59 LINES~~
     ~~HEADING 1~~
     ~~FIRST DETAIL 9~~
     ~~LAST DETAIL 48~~
     ~~FOOTING 52.~~
```

```
                     JANUARY EXPENDITURES (CONTINUED)

MONTH      DAY   DEPT   NO-PURCHASES   TYPE       COST   CUMULATIVE-COST

JANUARY     26   A04               5     A        5.00
                 A04               5     B        7.80

PURCHASES AND COST FOR 1-26   10                $12.80           $338.12
**********************************************************************
JANUARY     27   A00               6     B        9.36
                 A00              15     C      120.00

PURCHASES AND COST FOR 1-27   21               $129.36           $467.48
**********************************************************************
JANUARY     30   A00               2     B        3.12
                 A02              10     A       10.00
                 A02               1     C        8.00
                 A04              15     B       23.40
                 A04              10     C       80.00

PURCHASES AND COST FOR 1-30   38               $124.52           $592.00
**********************************************************************
JANUARY     31   A00               1     A        1.00
                 A04               6     A        6.00

PURCHASES AND COST FOR 1-31    7                 $7.00           $599.00
**********************************************************************
             TOTAL COST FOR JANUARY   WAS      $599.00
```

CONTROL clause specifies that control breaks occur when:

1. DAY report field changes value
2. MONTH report field changes value
3. FINAL report field changes value (that is when end-of-report is reached. (Not shown on this page.)

Figure 66. RD Entry—CONTROL Clause Concepts

THE CODE CLAUSE: allows the programmer, when two reports are written to one file, to specify a differentiating code character for each.

Report Group Description Entries

Following the RD entry are one or more associated level-01 entries, known as report group description entries. Each such level-01 entry, together with its hierarchy of subentries, describes a report group—a unit of output information to be produced in the report. A report group consists of one or more complete lines in the report; it can never be a partial line.

Through the report group description entries, the physical and logical characteristics of each report group are completely described. For each report group, the programmer specifies:

- Its function—Through a TYPE clause specified at the 01 level. The function of the group may be as a report, page, or control heading or footing, or as a detail line. (A control heading or footing is a report group produced only when a control break occurs.)

- Its vertical spacing—Both within itself (if this report group contains more than one complete line) and/or in relation to other report groups. Vertical spacing can be specified as absolute (as a specific line on a page) or as relative (as an increment to the last line printed).

- The LINE Clause—Specifies absolute or relative spacing for this report group.

- The NEXT GROUP Clause—Specifies absolute or relative spacing before the next report group is printed.

The Report Writer manipulates LINE and NEXT GROUP clause spacing in accordance with the page parameters set up through the PAGE clause in the RD entry.

- Its horizontal spacing—Specified through the COLUMN clause as an absolute displacement from the leftmost print position and through the length and editing characteristics of the associated PICTURE clause.

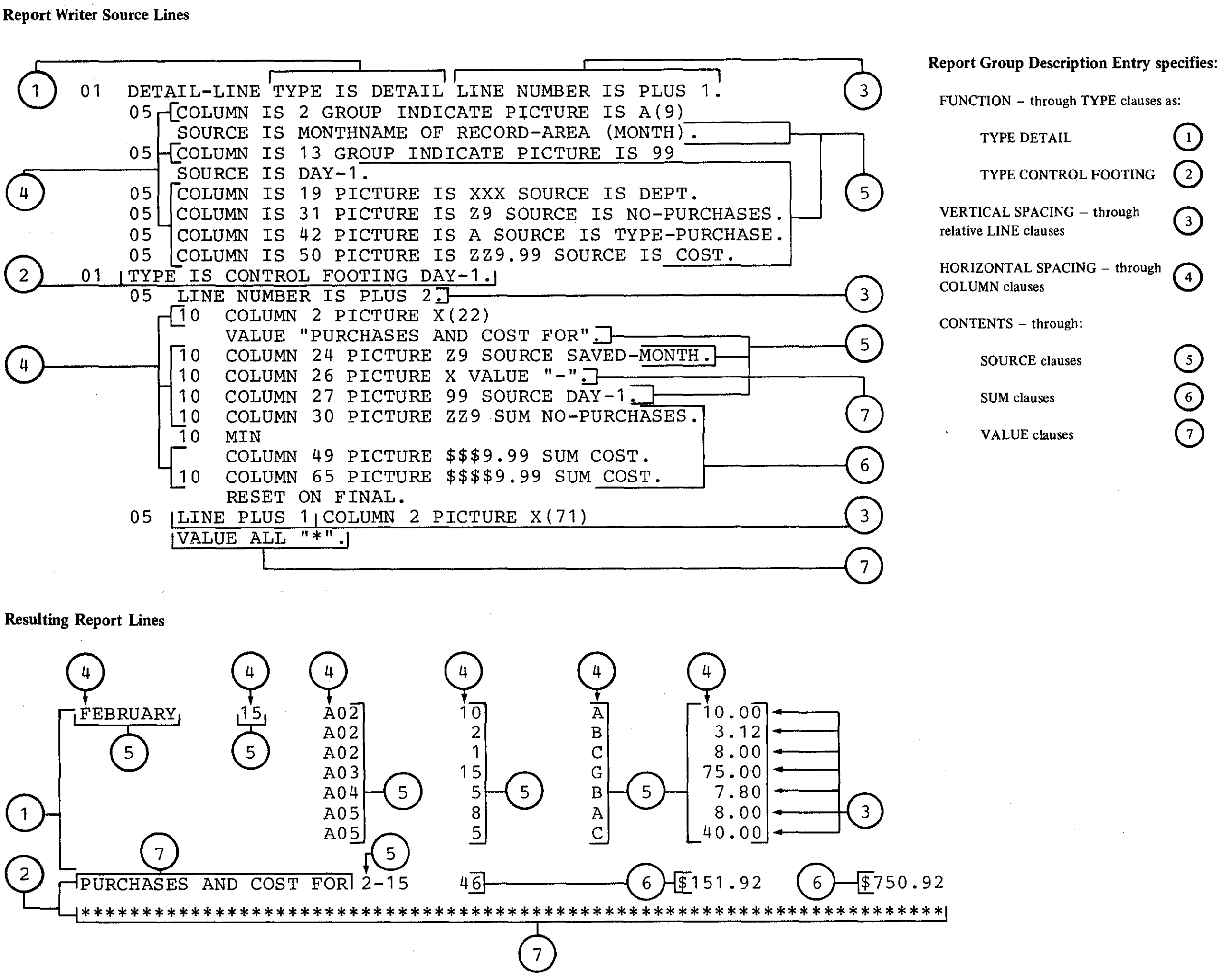
- Its contents—Specified through one of three clauses:

  The VALUE Clause—Identifies the contents of this entry as having a constant value to be used each time the entry is produced.

  The SOURCE Clause—Identifies a data item outside the Report Section; whenever this report entry is to be produced, the current value of this data item is used.

  The SUM Clause—Identifies the contents of this footing entry as being the result of addition operations. Through the SUM clause various totals can be calculated from the input data and printed as output in the final report.

These report group description entry concepts are illustrated in Figure 67.



Report Writer Source Lines

```
 01  DETAIL-LINE TYPE IS DETAIL LINE NUMBER IS PLUS 1.
     05 COLUMN IS 2 GROUP INDICATE PICTURE IS A(9)
        SOURCE IS MONTHNAME OF RECORD-AREA (MONTH).
     05 COLUMN IS 13 GROUP INDICATE PICTURE IS 99
        SOURCE IS DAY-1.
     05 COLUMN IS 19 PICTURE IS XXX SOURCE IS DEPT.
     05 COLUMN IS 31 PICTURE IS Z9 SOURCE IS NO-PURCHASES.
     05 COLUMN IS 42 PICTURE IS A SOURCE IS TYPE-PURCHASE.
     05 COLUMN IS 50 PICTURE IS ZZ9.99 SOURCE IS COST.
 01  TYPE IS CONTROL FOOTING DAY-1.
     05  LINE NUMBER IS PLUS 2.
         10  COLUMN 2 PICTURE X(22)
             VALUE "PURCHASES AND COST FOR".
         10  COLUMN 24 PICTURE Z9 SOURCE SAVED-MONTH.
         10  COLUMN 26 PICTURE X VALUE "-".
         10  COLUMN 27 PICTURE 99 SOURCE DAY-1.
         10  COLUMN 30 PICTURE ZZ9 SUM NO-PURCHASES.
         10  MIN
             COLUMN 49 PICTURE $$$9.99 SUM COST.
         10  COLUMN 65 PICTURE $$$$9.99 SUM COST.
             RESET ON FINAL.
     05  LINE PLUS 1 COLUMN 2 PICTURE X(71)
         VALUE ALL "*".
```

Report Group Description Entry specifies:

FUNCTION – through TYPE clauses as:

TYPE DETAIL ①

TYPE CONTROL FOOTING ②

VERTICAL SPACING – through relative LINE clauses ③

HORIZONTAL SPACING – through COLUMN clauses ④

CONTENTS – through:

SOURCE clauses ⑤

SUM clauses ⑥

VALUE clauses ⑦

Resulting Report Lines

```
 FEBRUARY   15  A02     10     A    10.00
                A02      2     B     3.12
                A02      1     C     8.00
                A03     15     G    75.00
                A04      5     B     7.80
                A05      8     A     8.00
                A05      5     C    40.00
 PURCHASES AND COST FOR 2-15     46           $151.92        $750.92
 ***********************************************************************
```

Figure 67. Report Group Description Entry Concepts

The Report Writer always treats a report group as a unit, even though it may be printed on more than one line. When page organization is specified, a report group is always printed on one page; it is never begun on one page and completed on another. Therefore, before any report group is produced, the Report Writer performs a page fit test, and if the complete report group will not fit on this page (that is, if a _page break_ occurs) any necessary page footing is produced, the page is ejected, and any necessary page heading is produced before this report group is printed.

PROCEDURE DIVISION CONCEPTS

In the Procedure Division, three basic Report Writer statements transfer control to the Report Writer subroutines: INITIATE, GENERATE, and TERMINATE. In addition, the USE BEFORE REPORTING Declarative sentence allows the programmer to specify special procedures to be executed before specific report groups are printed.

INITIATE Statement

The INITIATE statement is analogous to the OPEN statement for a file. An INITIATE statement must be executed before Report Writer processing can begin. Execution of the INITIATE statement initializes counters, and so forth.

GENERATE Statement

This statement causes the Report Writer to automatically produce the various report groups defined in the Report Section, where and when they are needed according to the report logic. Two kinds of reporting are provided: summary reporting and detail reporting.

SUMMARY REPORTING: In summary reporting, only heading and footing report groups are produced by the Report Writer; they are produced in the order required by the report logic. As specified by the programmer, counters are reset each time a report group is produced.

DETAIL REPORTING: In detail reporting, the Report Writer produces the TYPE DETAIL report group named in the GENERATE statement each time it is executed, as well as the heading and footing report groups produced in summary reporting.

TERMINATE Statement

This statement is analogous to a CLOSE statement for a file. Execution of the TERMINATE statement completes report processing as if a control break at the highest level has occurred; all footing report groups up to the highest level are produced, all counters are reset, and the Report Writer processing is ended.

USE BEFORE REPORTING Sentence

This sentence, after a section header in the Declaratives section, allows the programmer to write declarative procedures to modify any heading or footing report group before it is printed. Through such procedures, the programmer can specify any special processing requirements beyond the basic Report Writer functions.

Special processing may include any calculations required for the report besides the additions provided through the SUM clause, any editing of a line of the report, or some other modification desired by the programmer.

The sample program at the end of this chapter illustrates Report Writer features; it utilizes both detail and summary reporting.

Rules for writing COBOL programs using the Report Writer feature are given in the following sections.

<u>REPORT WRITER—ENVIRONMENT DIVISION</u>

In the Environment Division, there are special considerations for the Special-Names paragraph and the File-Control entry.

SPECIAL-NAMES PARAGRAPH

When more than one report is to be written on a file, the identifying character to be used in the CODE clause must be given a mnemonic-name.

Format

```
SPECIAL-NAMES.

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

When used with the CODE clause, function-name-1 must be a one-character nonnumeric literal.

Mnemonic-name follows the rules for formation of a user-specified name; at least one character must be alphabetic.

FILE-CONTROL ENTRY

The File-Control entry for the report file must specify a physical sequential file.

REPORT WRITER—DATA DIVISION

In the File Section, for each report to be produced, there must be one or more FD entries, each of which contains a REPORT clause.

In the Report Section, there must be a report description (RD) entry for each report to be produced. The RD entry describes the structure and organization of the report. Following each RD entry, there must be one or more report group description entries to describe the conceptual characteristics of the report.

FILE SECTION

The FD entry describes the physical structure of the file on which the report is to be written and names any reports associated with this file.

Format

```
FD  file-name

    ┌ REPORT IS    ┐
    <              > report-name-1 [report-name-2] ...
    | REPORTS ARE  |
    └              ┘

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

   [BLOCK CONTAINS Clause]
    LABEL RECORDS Clause
   [DATA RECORDS Clause]
   [VALUE OF Clause].
```

File-name must specify a physical sequential file.

Specification of the BLOCK CONTAINS, LABEL RECORDS, DATA RECORDS, and VALUE OF clauses is as described in the Data Division chapter.

Special considerations for the REPORT clause, the RECORD CONTAINS clause, and for record description entries follow.

REPORT CLAUSE

The REPORT clause associates RD entries in the Report Section with the FD entry (or entries) for the report in the File Section.

Each unique report-name must appear in the REPORT clause of the FD entry (or entries) for the file(s) on which the report is produced. A given report-name may appear in a maximum of two REPORT clauses.

The order in which report names appear in the REPORT clause is not significant.

Each report-name must be the subject of an RD entry in the Report Section. Each named report may have a unique size, format, logical structure, and so forth.

When an FD entry includes at least one REPORT clause, an associated record description entry need not be specified, because the associated RD entry and report group description entries completely describe the records associated with this file.

RECORD CONTAINS CLAUSE

The RECORD CONTAINS clause specifies the size of the report record.

Integer-2 specifies the maximum size of the record. This size includes the length of the print line, plus the carriage control character, and, if the CODE clause is specified, the report code character.

When integer-2 alone is specified, records are fixed length, and the size of each print line and each blank line is as follows:

- When CODE clause is not specified (integer-2-1)

- When CODE clause is specified (integer-2-2)

See the System Dependencies chapter.

When both integer-1 and integer-2 are specified, records are variable length. The size of each print line is calculated as for fixed length records. Blank lines are 17 characters in length.

If the RECORD CONTAINS clause is omitted, the compiler assumes a record size of 133 characters.

RECORD DESCRIPTION ENTRIES

Optionally, the FD entry for a report file may be followed by one or more record description entries.

When such entries are included and the file is used for OUTPUT, the WRITE AFTER ADVANCING statement must be specified when creating records for the file.

When Report Writer output is printed online, DISPLAY statements to the same device in the same program should not be used, or line control in the Report Writer output will be unpredictable.

REPORT SECTION

The Report Section describes one or more reports to be produced.

The Report Section begins with the header REPORT SECTION. The Report Section contains two types of entries. For each report, the entries are written in the following order:

1. The RD Entry—which names the report and describes the physical structure and organization of the report.

2. Report Group Description Entries—which immediately follow the associated RD entry, and which specify the conceptual characteristics of the report and the physical structure of each printed line.

RD ENTRY

The RD entry describes the structure and organization of the report.

Format

```
RD  report-name

    [WITH CODE mnemonic-name]

      ┌             ┐ ┌         ┐
      | CONTROL IS  | | FINAL   |
   [ <              >  <        > identifier-1 [identifier-2] ... ]
      | CONTROLS ARE| | [FINAL] |
      └             ┘ └         ┘

         ┌           ┐               ┌       ┐
         | LIMIT IS  |               | LINE  |
    [PAGE|           |  integer-1   <        >
         | LIMITS ARE|               | LINES |
         └           ┘               └       ┘

         [HEADING      integer-2]
         [FIRST DETAIL integer-3]
         [LAST DETAIL  integer-4]
         [FOOTING      integer-5] ].
```

The clauses that follow the name of the RD entry are optional. The order of their appearance is not significant. In one RD entry, each may be specified only once.

REPORT-NAME

The report-name must be unique and must immediately follow the level indicator RD. The report-name must follow the rules of formation for a user-specified name; at least one character must be alphabetic. The report-name must also be specified in the FD entry for the file on which this report is to be written.

Report-name is the only qualifier for the LINE-COUNTER and PAGE-COUNTER special registers and for all data-names associated with this report.

CODE CLAUSE

The CODE clause specifies an identifying character placed at the beginning of each report line. This identifying character has meaning only when more than one report is written on the file.

Mnemonic-name must be associated with a 1-character nonnumeric literal specified as function-name-1 in the Environment Division SPECIAL-NAMES paragraph. This 1-character literal is the code character associated with each line of the report.

The code character precedes the carriage-control character at the beginning of each line.

When the report is printed online, the CODE clause must not be specified.

CONTROL CLAUSE

The CONTROL clause specifies the control hierarchy for the report.

Identifier-1 and identifier-2 must each specify different fixed-length nonedited data items. That is, their descriptions must be such that they neither occupy (partially or completely) the same area of storage, nor do their PICTURE strings contain any editing characters. Identifier-1 and identifier-2 must not be defined within the Report Section.

Identifier-1 and identifier-2 are considered to be controls for this report; when a control changes in value (that is, when a control break occurs), special Report Writer actions are performed. The order in which the controls are specified represents their position, in descending order, within the control hierarchy for this report.

FINAL, explicitly or implicitly, represents the highest level of control. Identifier-1 represents the major control, identifier-2 represents the intermediate control, and so forth. The last specified identifier is the minor (lowest level) control.

In report groups for this report, FINAL and control identifiers are the only valid operands named in TYPE CONTROL FOOTING or RESET clauses.

The CONTROL clause is required when any control report groups other than FINAL are used. When only a FINAL control is used, the CONTROL clause may be omitted.

When a DETAIL report group is to be generated, controls are tested for a control break.

Controls are tested in the order in which they appear in the CONTROL clause: highest level first, then next highest level, etc. When a control break is found for a particular level, a control break is implied for each lower level control as well. A control break for FINAL occurs only once, at the beginning and ending of the report (that is, before the first detail line is printed and after the last detail line is printed).

To ensure that source fields in control headings and footings contain correct data during control break processing, these fields should be included in the CONTROL clause.

The action to be taken when a control break occurs depends on what the programmer defines. For each control, one CONTROL HEADING report group and/or one CONTROL FOOTING report group, or neither may be specified.

Before the DETAIL group causing the control break is printed, the specified control headings and footings are printed in the following order:

1. Lowest level control footing

2. Next higher control footing

.

.

.

x. Control footing for which control break occurred

y. Control heading for which control break occurred

z. Next lower control heading

.

.

.

n. Lowest level control heading

n+1. DETAIL line causing control break

If, while control footings are being printed, end-of-page is reached (that is, a page break occurs), any PAGE FOOTING is printed, a new page is begun, and any PAGE HEADING is printed.

PAGE CLAUSE

The PAGE clause defines the logical page size and the vertical subdivisions of the page size within which report groups are presented.

The PAGE clause is required only when page format must be controlled by the Report Writer. If page formatting is not desired, the PAGE clause can be omitted; in this case all LINE and NEXT GROUP clauses must specify relative line spacing.

When the PAGE clause is omitted, no PAGE-COUNTER or LINE-COUNTER special registers are generated for this RD entry.

The options of the PAGE clause are discussed in the following paragraphs. The permissible specifications for each option are dependent on what other options have been specified. The overall rules are:

- Integer-1 through integer-5 must be positive integers.
- Integer-2 through integer-5 must be in ascending order.
- Integer-5 must not exceed integer-1.

PAGE LIMIT integer-1

Integer-1 specifies the depth of the report page, and the last line on which anything may be printed. If the PAGE clause is specified, integer-1 must be specified. The report page depth may or may not be equal to the physical page depth.

Integer-1 must not exceed 999.

HEADING integer-2

Integer-2 specifies the first line number of the first heading print group. Integer-2 is the first line on which anything may be printed.

FIRST DETAIL integer-3

Integer-3 specifies the first line number of the first body group (a body group is a TYPE DETAIL, CONTROL HEADING, or CONTROL FOOTING report group). No body group is printed before integer-3.

LAST DETAIL integer-4

Integer-4 specifies the last line number of the last nonfooting body group. No CONTROL HEADING or DETAIL report group is printed after integer-4.

FOOTING integer-5

Integer-5 specifies the last line number of the last CONTROL FOOTING report group. No CONTROL FOOTING report groups are printed beyond integer-5. (PAGE FOOTING report groups are printed beyond integer-5.)

General Considerations

The Report Writer uses the values specified in the PAGE clause to establish lines on the logical page within which each TYPE of report group can be printed. Figure 68 shows the valid page area for each type of report group, and also which options of the PAGE clause are required.

TYPE Clause Option	Valid LINE Clause Specification Range within PAGE Clause Ranges	Required PAGE Clause Entries
REPORT HEADING RH (separate page)	integer-2 to integer-1	PAGE LIMIT integer-1
REPORT HEADING RH (same page)	integer-2 to integer-1	PAGE LIMIT integer-1 FIRST DETAIL integer-3
PAGE HEADING PH	integer-2 [1] to (integer-3 - 1)	PAGE LIMIT integer-1 FIRST DETAIL integer-3
CONTROL HEADING CH DETAIL DE	integer-3 to integer-4	PAGE LIMIT integer-1
CONTROL FOOTING CF	integer-3 to integer-5	PAGE LIMIT integer-1
PAGE FOOTING PF	(integer-5 + 1) [2] to integer-1	PAGE LIMIT integer-1 LAST DETAIL integer-4 FOOTING integer-5
REPORT FOOTING RF (same page)	integer-2 to integer-1	PAGE LIMIT integer-1 LAST DETAIL integer-4 FOOTING integer-5
REPORT FOOTING RF (separate page)	integer-2 to integer-1	PAGE LIMIT integer-1

[1] If a PAGE HEADING report group follows a REPORT HEADING report group on the same page, both must be printable in this area of the page.

[2] If a PAGE FOOTING report group precedes a REPORT FOOTING report group on the same page, both must be printable in this area of the page.

Figure 68. Valid LINE Clause Specifications for Each PAGE Clause Range

Figure 69 illustrates the areas defined in Figure 68.

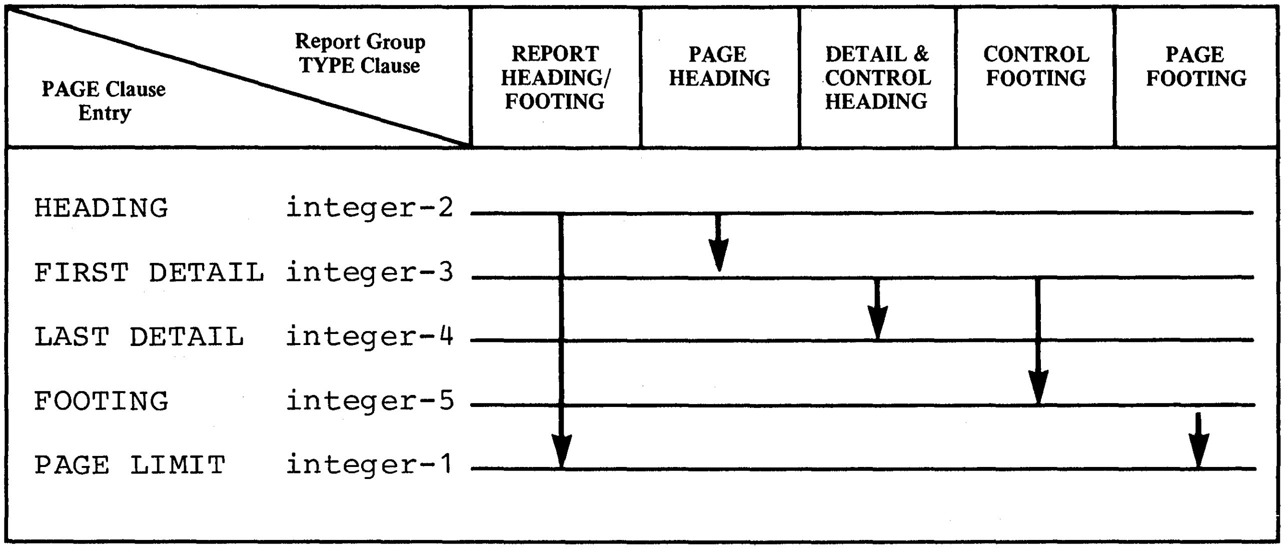


Figure 69. PAGE Clause Ranges

When the PAGE clause is specified, and some or all of the optional limits are omitted, the Report Writer assumes certain values for each omitted specification. Figure 70 defines the assumed values.

Omitted PAGE LIMIT Entry	Default Value Assumed
HEADING integer-2	integer-2 = 1
FIRST DETAIL integer-3	integer-3 = integer-2
LAST DETAIL integer-4	integer-4 = integer-5
FOOTING integer-5	integer-5 = integer-4
both LAST DETAIL integer-4 and FOOTING integer-5	integer-4 & integer-5 = integer-1

Figure 70. Values Assumed for Omitted PAGE Clause Options

If absolute line spacing is specified in the LINE clause for all report groups, then integer-2 through integer-5 need not be specified. If any of these limits are specified, and the report uses only absolute line spacing, the limits are ignored.

When relative line spacing is specified for any report group, all LINE NUMBER and NEXT GROUP spacing must be consistent with the parameters specified (implicitly or explicitly) in the PAGE clause.

REPORT GROUP DESCRIPTION ENTRY

The report group description entry defines the format and characteristics of a report group.

A report group consists of one or more printable lines treated as a unit of a report; a report group may also be a null group, that is, a nonprintable report group.

General Format 1—Level-01 Group Entry

```
01  [data-name]
    TYPE Clause
    [LINE clause]
    [NEXT GROUP clause]
    [USAGE clause].
```

General Format 2—Group Entry

```
level-number  [data-name]
    [LINE clause]
    [USAGE clause].
```

General Format 3—Elementary Entry

```
level-number [data-name]

   [COLUMN NUMBER IS integer]
   [GROUP INDICATE]
   [LINE Clause]
┌                                                           ┐
|   SOURCE IS identifier                                    |
|   SUM identifier-1 [identifier-2] ... [UPON data-name-2]  |
|                    ┌               ┐                      |
<       [RESET ON   | FINAL          |                      >
|                   <                >  ]                   |
|                   | identifier-3   |                      |
|                   └               ┘                       |
|   VALUE IS literal                                        |
└                                                           ┘

  PICTURE Clause
 [USAGE Clause]
 [BLANK WHEN ZERO Clause]
 [JUSTIFIED Clause].
```

General Format 4—Level-01 Elementary Entry

```
01 [data-name]
    TYPE Clause
    [LINE Clause]
    [NEXT GROUP Clause]
    [COLUMN Clause]
    [GROUP INDICATE Clause]

      ┌                 ┐
      | SOURCE Clause   |
      < SUM Clause      >
      | VALUE Clause    |
      └                 ┘

   PICTURE Clause
   [USAGE Clause]
   [BLANK WHEN ZERO Clause]
   [JUSTIFIED Clause].
```

A report is made up of one or more report groups, each described by a hierarchy of entries similar to data description entries.

General Format 1—Level-01 Group Entry

This format specifies the beginning of a report group description entry, and is the first entry for a report group.

General Format 2—Group Entry

This format specifies a group entry. Valid level-numbers are 02 through 48. A group entry has the following functions:

- To specify the LINE number of subordinate elementary entries

- To group subordinate elementary entries together

(If a group entry contains no LINE clause, and there are no subordinate sum counters, this group entry serves only as documentation.)

General Format 3—Elementary Entry

This format specifies an elementary entry. Valid level-numbers are 02 through 49. An elementary entry has the following functions:

- To describe a printable group. In this case a COLUMN clause, a PICTURE clause, and either a SOURCE, SUM, or VALUE clause must be specified.

- To describe a null group. In this case no COLUMN, VALUE, or LINE clause may be specified.

Elementary groups (both printable and null) perform the following summing functions:

- To describe an elementary DETAIL entry that specifies which SOURCE items are to be summed when this DETAIL group is produced (see SOURCE clause).

- To define a sum counter in an elementary CONTROL FOOTING entry (see SUM clause).

General Format 4—Level-01 Elementary Entry

This format describes a report group that consists of one elementary entry. When General Format 4 is specified, this entry must be the only entry for the report group.

General Considerations

The following rules apply for the specification of report group description entries:

1. The first entry within the report group must be at level 01, and must contain a TYPE clause. It may be either a General Format 1 group entry, or, if it is the only entry in this report group, it may be a General Format 4 elementary entry.

2. Immediately subordinate entries may follow. These may be either General Format 2 group entries or General Format elementary entries.

3. Any General Format 2 group entry must have at least one General Format 3 elementary entry subordinate to it.

4. The report group description entry ends when the next RD entry or level-01 entry is encountered, or when the end of the Report Section is reached.

5. Except for the data-name clause, which, when written, must immediately follow the level-number, the order in which the clauses are written is not significant.

6. A <u>null group</u> is a report group for which no LINE or COLUMN clauses have been specified.

Figure 71 shows which report group clauses are permitted, which are required, and which are invalid for each report group TYPE.

Group or Elementary Report Groups						
TYPE Clause	LINE Clause			NEXT GROUP Clause		
	Absolute	Relative	NEXT PAGE	Absolute	Relative	NEXT PAGE
RH (S)	P	P				R
RH (NS)	P	P		P	P	
PH	P	P				
CH	P	P	P	P	P	P
DE	P	P	P	P	P	P
CF	P	P	P	P	P	P
PF	P	P				
RF (NS)	R[1]	R[1]				
RF (S)	P	P	R			

Elementary Report Groups							
TYPE Clause	GROUP INDICATE Clause	COLUMN[2] Clause	PICTURE[2] Clause	VALUE[3] Clause	SOURCE[3] Clause	SUM[3] Clause	RESET[4] Option
RH (S)		P	P	P	P		
RH (NS)		P	P	P	P		
PH		P	P	P	P		
CH		P	P	P	P		
DE	P	P	P	P	P		
CF		P	P	P	P	P	P
PF		P	P	P	P		
RF (NS)		P	P	P	P		
RF (S)		P	P	P	P		

(S) = on separate page
(NS) = not on separate page

P = permitted
R = required
space = not permitted

1 – only one is required, both are permitted
2 – required if item is to be printed
3 – one is required if item is to be printed
4 – permitted only when SUM clause is specified

Figure 71. Valid Report Group Clauses—by Report Group TYPE

DATA-NAME CLAUSE

The data-name clause identifies the entry.

Format

level-number [data-name]

When specified, data-name must immediately follow level-number.

Data-name is required only for:

- A TYPE DETAIL report group
- A report group named as an operand in a SUM clause
- A HEADING or FOOTING report group referred to in a USE BEFORE REPORTING Declarative procedure

In all other cases, data-name may be omitted. (Note that, when data-name is omitted in the Report Section, the key word FILLER must not be used in its place.) Within a given report, data-name must be unique.

TYPE CLAUSE

The TYPE clause specifies the function of this report group, and when this report group is to be processed.

Format

```
          ┌                                                ┐
          | ┌ REPORT HEADING ┐                             |
          | <                >                             |
          | | RH             ┘                             |
          | └                                              |
          | ┌ PAGE HEADING ┐                               |
          | <              >                               |
          | | PH           ┘                               |
          | └                                              |
          | ┌ CONTROL HEADING ┐  ┌ FINAL          ┐        |
          | <                 >  <                >        |
          | | CH              ┘  | identifier-n   ┘        |
          | └                    └                         |
          | ┌ DETAIL ┐                                     |
TYPE IS < <          >                                     >
          | | DE     ┘                                     |
          | └                                              |
          | ┌ CONTROL FOOTING ┐  ┌ identifier-n   ┐        |
          | <                 >  <                >        |
          | | CF              ┘  | FINAL          ┘        |
          | └                    └                         |
          | ┌ PAGE FOOTING ┐                               |
          | <              >                               |
          | | PF           ┘                               |
          | └                                              |
          | ┌ REPORT FOOTING ┐                             |
          | <                >                             |
          | | RF             ┘                             |
          | └                                              |
          └                                                ┘
```

The TYPE clause must be specified only for a level-01 entry in a report group description entry.

The options are self-explanatory, and the abbreviations are the equivalent of the fully spelled-out option. A TYPE REPORT HEADING specifies that the line(s) in this report group are to be treated as the heading of the report. The TYPE PAGE HEADING specifies that the line(s) of this report group are to be treated as the page heading for this report, etc.

Figure 72 gives the options of the TYPE clause, the number of each allowed, when each is produced, and the formatting sequence. Additional rules follow:

TYPE Clause Option[1]	Number Allowed	When Produced	Page Placement
REPORT HEADING	1	First GENERATE statement only	precedes all other groups
PAGE HEADING	1	First GENERATE statement and each page break	precedes CH, DE, CF, PF on each page
CONTROL HEADING FINAL	1	First GENERATE statement only	precedes DE, CF, PF on first page
CONTROL HEADING identifier	1 for each identifier	control break[2]	precedes DE, CF, PF on each page
DETAIL	as logically needed	Each GENERATE statement[3]	follows all HEADING, precedes all FOOTING groups
CONTROL FOOTING identifier	1 for each identifier	control break[2]	follows DE groups
CONTROL FOOTING FINAL	1	TERMINATE statement	follows last DE and other CF groups
PAGE FOOTING	1	at each page break	follows DE and CF groups on each page
REPORT FOOTING	1	TERMINATE statement	follows all other groups

[1] the report groups may be specified in any sequence.

[2] the control break must occur at the level of this identifier, or at a higher level in the control hierarchy.

[3] a printable DETAIL group is printed only if the GENERATE data-name form of the statement is used.

Figure 72. TYPE Clause Presentation Rules

REPORT HEADING: Nothing precedes a REPORT HEADING group in a report. An associated SOURCE clause refers to the value of the SOURCE data item at the time the first GENERATE statement is executed.

PAGE HEADING: The beginning of the page is determined according to the page condition rules as specified in the PAGE clause.

CONTROL HEADING FINAL: This report group is produced once during execution of the first GENERATE statement, before any DETAIL group is printed. FINAL need not be named in the associated CONTROL clause. An associated SOURCE clause refers to the value of the SOURCE data item at the time the first GENERATE statement is executed.

CONTROL HEADING IDENTIFIER-N: A control break must occur to produce this report group. Identifier-n must be named in the associated CONTROL clause. An associated SOURCE clause refers to the value of the SOURCE data item at the time this CONTROL HEADING is presented.

Note: Identifier-n must be qualified (or not qualified) in exactly the same way as the corresponding identifier in the CONTROL clause.

DETAIL: This report group must have a unique data-name. An associated SOURCE clause refers to the value of the SOURCE data item at the time the DETAIL group is produced.

CONTROL FOOTING IDENTIFIER-N: A control break must occur to produce this report group. Identifier-n must be named in the associated CONTROL clause. An associated SOURCE clause refers to the value of the SOURCE data item at the time the CONTROL FOOTING is presented.

Note: Identifier-n must be qualified (or not qualified) in exactly the same way as the corresponding identifier in the CONTROL clause.

CONTROL FOOTING FINAL: This report group is produced once at the termination of the report. FINAL need not be named in the associated CONTROL clause. An associated SOURCE clause refers to the value of the SOURCE data item at the time the TERMINATE statement is executed.

PAGE FOOTING: The ending of the page is determined according to the page condition rules as specified in the PAGE Clause.

REPORT FOOTING: Nothing follows a report footing group in a report. An associated SOURCE clause refers to the value of the SOURCE data item at the time the TERMINATE statement is executed.

Figure 71 gives the relation of the TYPE clause to the other report group description clauses.

Note: When a USE BEFORE REPORTING declarative procedure is associated with a CONTROL FOOTING report group:

- For a SOURCE item named in a CONTROL clause, the value used is the value of the item before the control break.

- For any other SOURCE item, the value used is the value of the item after the control break.

LINE CLAUSE

The LINE clause specifies the absolute or relative line number of this entry in reference to either the page or the preceding entry.

Format

```
                  ┌                  ┐
                  │ integer-1        │
[LINE NUMBER IS   < PLUS integer-2   >   ]
                  │ NEXT PAGE        │
                  └                  ┘
```

A LINE clause must be associated with each line of a report.

In any report group, the LINE clause must be specified as follows:

1. For the first line, at the 01-level, or before or with the first elementary item in the line

2. For subsequent lines, before or with the first elementary item in the line

When an entry with a LINE clause is specified, subsequent entries are implicitly presented on the same line until another LINE clause or the end of the report group is encountered.

Integer-1 and integer-2 must be positive integers.

LINE NUMBER IS INTEGER-1: This option is an absolute LINE clause; it specifies the fixed position on the page at which this line is produced. LINE-COUNTER is set to the value of integer-1, and is used for positioning this and following entries until a different value of LINE-COUNTER is specified. (See the description of Report Writer Special Registers later in this chapter.)

Within a report group description entry, absolute LINE clause entries must be specified in ascending order.

Within a report group description entry, an absolute LINE clause must not be preceded by a relative LINE clause.

LINE NUMBER IS PLUS INTEGER-2: This option is a relative LINE clause. The line is printed relative to the previous line position. LINE-COUNTER is increased by the value of integer-2 and is used for positioning this and following entries until a different value is specified. (See the following discussion of report group types for exceptions to this rule.)

If the print line of the first body group on a page contains a relative LINE clause, the line is printed on line FIRST DETAIL integer-3, unless LINE-COUNTER contains an equal or greater value. In this case, the line is printed on line LINE-COUNTER plus 1.

Additional rules for relative LINE clauses depend on the report group type. The following discussion on report group types gives additional rules.

LINE NUMBER IS NEXT PAGE: This option specifies that this report group is to be printed on the next following page, rather than on the current page. It may be used only as the first LINE clause within a report group.

If the report group entry for a body group contains a LINE NUMBER IS NEXT PAGE clause, and the first line contains a relative LINE clause, the first line is printed relative to FIRST DETAIL integer-3 or relative to LINE-COUNTER, whichever is greater.

The line is printed on line FIRST DETAIL integer-3, unless LINE-COUNTER contains an equal or greater value. In this case, the line is printed on line LINE-COUNTER 1.

Note: At the beginning of a new page, LINE-COUNTER can contain a value equal to or greater than FIRST DETAIL integer-3 only as the result of a NEXT GROUP clause specified in the previous body group. (See "NEXT GROUP Clause.")

Report Group Types

Figure 71 shows permissible combinations of the report group LINE clause and the TYPE clause and with other report group description clauses. Special considerations for each report group type follow:

1. REPORT HEADING. If the first line contains a relative LINE clause, it is relative to line HEADING integer-2.

2. PAGE HEADING. If the first line contains a relative LINE clause, it is relative to line HEADING integer-2 or LINE-COUNTER, whichever is greater. (If the value in LINE-COUNTER is greater than HEADING integer-2, it means that a REPORT HEADING print group precedes the PAGE HEADING group on this page.)

3. PAGE FOOTING. If the first line contains a relative LINE clause, it is relative to line FOOTING integer-5 plus one.

4. REPORT FOOTING. If the first line contains a relative LINE clause, it is relative to line FOOTING integer-5 or LINE-COUNTER, whichever is greater. (If the value in LINE-COUNTER is greater than FOOTING integer-5, it means that a PAGE FOOTING report group precedes the REPORT FOOTING group on this page.)

   If LINE IS NEXT PAGE is the only LINE clause in this report group, the line is printed on line HEADING integer-2.

NEXT GROUP CLAUSE

The NEXT GROUP clause specifies the spacing condition following the last line of a report group.

Format

```
                     ┌                 ┐
                     │ integer-1       │
[NEXT GROUP IS       < PLUS integer-2  >   ]
                     │ NEXT PAGE       │
                     └                 ┘
```

The NEXT GROUP clause may be specified only in a level-01 entry for a REPORT HEADING, PAGE FOOTING, or body group.

Integer-1 and integer-2 must be positive integers.

If a USE BEFORE REPORTING declarative procedure causes a report group to be suppressed, the NEXT GROUP clause functions are not performed for this execution of this report group.

NEXT GROUP IS INTEGER-1: This option specifies an absolute NEXT GROUP clause.

Integer-1 specifies an absolute line number. After the last line of this report group is produced, LINE-COUNTER is set to integer-1.

NEXT GROUP IS PLUS INTEGER-2: This option specifies a relative NEXT GROUP clause.

Integer-2 specifies a relative line number. After the last line of this report group is produced, LINE-COUNTER is incremented by integer-2.

NEXT GROUP IS NEXT PAGE: This option specifies that after this report group is produced, LINE-COUNTER is set so that the next report group will be produced on the next page.

Report Group Types

Figure 71 shows permissible combinations of the NEXT GROUP clause and the report group TYPE clause and with other report group description clauses. There are also special considerations for each valid report group type.

REPORT HEADING GROUP: An absolute or relative NEXT GROUP clause must not cause LINE-COUNTER to be set to a value greater than FIRST DETAIL integer-3 minus 1.

BODY GROUPS: The following considerations apply:

- If a NEXT GROUP clause implies a page change, the change occurs only when the next body group is to be produced.

- If an absolute NEXT GROUP clause causes a page change, LINE-COUNTER is set to the value of integer-1 before the first line of the next printable body group is formatted.

- If the first line of the next body group contains a relative LINE clause, the line is printed on line LINE-COUNTER + 1.

- If the first line of the next body group contains an absolute LINE clause that is not greater than integer-1, a page containing only PAGE HEADING and PAGE FOOTING report groups will be printed, and the next body group will be printed on the following page.

- An absolute or relative NEXT GROUP clause may cause LINE-COUNTER to be set to a value in the range from line FIRST DETAIL integer-3 through line FOOTING integer-5. (This is an exception to the PAGE clause rules governing placement of CONTROL HEADING and DETAIL report groups.)

- NEXT GROUP IS NEXT PAGE specifies that no more body groups are to be printed on this page.

- For a CONTROL FOOTING group, when the NEXT GROUP clause is specified, the spacing is performed only when a control break at this level occurs.

COLUMN CLAUSE

The COLUMN clause specifies the horizontal positioning on the printed page of the leftmost character of an elementary item.

Format

COLUMN NUMBER IS integer

The COLUMN clause may be specified only for elementary level report group entries.

Integer must be a positive integer. The value 1 represents the leftmost print position on the line; successively higher values represent positions successively further right on the line.

The value of integer must be in a range that allows the associated elementary item to be printed within the print line for this report. That is, it may not exceed the implicit or explicit RECORD CONTAINS specification minus one (minus two if a CODE is used), minus the length of the associated elementary item.

If the COLUMN clause is omitted for an elementary entry, the elementary entry is not printed at the time its associated report group is printed.

Within a report group and within a specific line, COLUMN entries need not be specified in left to right order.

If the COLUMN clause is specified, this entry must also include a PICTURE clause and either a SOURCE, SUM, or VALUE clause.

Figure 71 shows permissible combinations of the COLUMN clause and other report group description entries.

PICTURE CLAUSE

For elementary entries, the PICTURE clause as described in the Data Division is applicable.

When the COLUMN clause is specified, the PICTURE clause must also be specified. The combination of COLUMN number specification and PICTURE clause description must not exceed the limits of the printable line.

When the COLUMN and PICTURE clauses are specified, either a SOURCE, SUM, or VALUE clause must also be specified.

Figure 71 shows permissible combinations of the PICTURE clause and other report group description entries.

GROUP INDICATE CLAUSE

The GROUP INDICATE clause specifies that this entry is to be produced only the first time the report group is produced after a control break or page break.

Format

GROUP INDICATE

The GROUP INDICATE clause may be specified only for an elementary entry in a TYPE DETAIL report group.

Only the first time after a control break or page break occurs is this elementary entry produced. At all other times it is suppressed.

Figure 71 shows permissible combinations of the GROUP INDICATE clause and other report group description entries.

Examples of the use of the GROUP INDICATE Clause are given in the Report Writer sample program at the end of this chapter.

SOURCE CLAUSE

The SOURCE clause specifies a data item whose current value is used for this elementary entry.

Format

SOURCE IS identifier

The SOURCE clause is valid only for elementary entries within a report group.

The SOURCE clause specifies a data item that is moved to this elementary entry just before this report group is produced.

Identifier must be one of the following:

- The name of a data item defined outside the Report Section
- The name of a SUM counter
- Any special register that is valid as the sending field in a MOVE statement

Identifier and the elementary report item described in its PICTURE clause must conform to the rules for sending items in a MOVE statement.

The SOURCE clause has two functions:

1. To specify a data item that is to be printed
2. To specify an item to be summed in a CONTROL FOOTING report group (see SUM clause)

Figure 71 shows permissible combinations of the SOURCE clause and other report group description entries.

For a discussion of special SOURCE clause considerations, see the description of the GROUP INDICATE clause.

SUM CLAUSE

The SUM clause establishes an elementary item as the receiving field (sum counter) for addition operations upon SOURCE data items and other sum counters.

Format

```
SUM identifier-1 [identifier-2] ... [UPON data-name]

                 ┌                ┐
                 | FINAL          |
   [ RESET ON   <                  >   ]
                 | identifier-3   |
                 └                ┘
```

The SUM clause is valid only at the elementary level within a CONTROL FOOTING report group (level-01 or subordinate), and establishes this elementary level report group as a sum counter.

If this report group is named, the data-name represents this sum counter in any program reference. When referred to in the Report Section, the data-name must not be qualified; when referred to in the Procedure Division, it may be qualified by the appropriate report-name. If this report group is not explicitly referred to within the program, it need not be named.

Each sum counter report group must contain a numeric or numeric edited PICTURE clause of sufficient size to prevent truncation of integral digits. Internally, the sum counter is always treated as a numeric item; if its PICTURE is numeric edited, the editing is performed only upon presentation of this report group.

Identifier-1 and identifier-2 are SUM clause operands. Three types of operands are allowed:

- Elementary numeric data items in the File, Working-Storage, or Linkage Sections

- Sum counters within any lower level CONTROL FOOTING report group for this report

- Sum counters in this CONTROL FOOTING report group

Except when they are sum counters, identifier-1 and identifier-2 may be qualified, subscripted, or indexed.

In any summing operation, operands are added in the physical order in which they appear in this report group description:

- SOURCE data items are subtotaled in this sum counter upon each generation of the DETAIL report group in which they are specified. (This is called SOURCE/SUM correlation.) If any SOURCE data item appears more than once in the DETAIL report group, it is added only once to this sum counter. For each such data-name, qualification, indexing, or subscripting within the SOURCE clause and the SUM clause must be identical.

- Counter rolling is performed; that is, sum counters from lower level CONTROL FOOTING report groups are added to this sum counter when the lower level CONTROL FOOTING report group is presented.

- Cross footing is performed; that is, sum counters in this CONTROL FOOTING report group are added to this sum counter before presentation of this CONTROL FOOTING report group.

UPON data-name Option

This option allows selective summing of particular data items named as source items in two or more DETAIL report groups.

Data-name must be the name of a DETAIL report group in the same report. Data-name may be qualified only by the report-name.

Identifier-1 and identifier-2 may be SOURCE data items within the data-name report group, or they may be items outside the Report Section.

RESET Option

This option specifies that a sum counter is to be reset to 0 only after a control break occurs at the level specified in the RESET option.

Identifier-3 must be specified in a CONTROL clause for this report, and must name a higher level control than the control for this report group.

When the RESET option is not specified, this sum counter is automatically reset to 0 after each presentation of this report group.

When the RESET option is specified, this sum counter is reset to 0 only after a control break in identifier-3, or after the FINAL control break.

General Considerations

1. The sum counter is initialized to 0 upon execution of the INITIATE statement.
2. SOURCE operands are subtotaled upon generation of their DETAIL report group(s).
3. Counter rolling is performed before presentation of a subordinate sum counter.
4. Cross footing is performed before presentation of this sum counter.
5. Any USE BEFORE REPORTING procedure for this report group is executed.
6. This report group (if it is not a null group) is presented.
7. When the RESET option is not specified, the sum counter is reset to 0 after presentation of this report group.
8. When the RESET option is specified, the sum counter is reset to 0 after the named higher level control break occurs.

Figure 71 shows permissible combinations of the SUM clause and of the RESET option, and other report group description entries.

VALUE CLAUSE

The VALUE clause causes an elementary entry to assume a specified value each time the report group is produced.

Format

<u>VALUE</u> IS literal

When the VALUE clause is specified, this elementary entry must also contain a PICTURE clause.

The literal must conform to the PICTURE clause specification. For example, if the PICTURE is numeric, the literal must be a numeric literal whose size does not exceed the size specified in the PICTURE clause.

Each time the report group is produced, the elementary entry assumes the value of literal. (However, see the description of the GROUP INDICATE clause for special considerations.)

Figure 71 shows permissible combinations of the VALUE clause and other report group description entries.

USAGE CLAUSE

USAGE DISPLAY must be implicitly or explicitly specified for each elementary report group description entry.

The DISPLAY option of the USAGE clause as described in the Data Division is applicable to report group description entries.

BLANK WHEN ZERO CLAUSE

For numeric or numeric edited elementary entries, the BLANK WHEN ZERO clause as described in the Data Division is applicable.

JUSTIFIED CLAUSE

For alphabetic or alphanumeric elementary entries, the JUSTIFIED clause as described in the Data Division is applicable.

REPORT WRITER—PROCEDURE DIVISION

To produce a report, the user must specify the INITIATE, GENERATE, and TERMINATE statements in the Procedure Division. In addition, a USE BEFORE REPORTING declarative procedure may be written to execute special instructions before a heading or footing report group is produced. There are also special considerations for the OPEN statement (see below).

OPEN STATEMENT

Before any Report Writer statement can be executed, the report file must be open in the OUTPUT mode.

INITIATE STATEMENT

The INITIATE statement begins the processing of a report.

Format

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

Each report-name must be defined by an RD entry in the Data Division Report Section.

Execution of the INITIATE statement does the following:

- Initializes all SUM counters to 0
- Sets up all TYPE CONTROL HEADING and/or TYPE CONTROL FOOTING items in the proper order, as defined by the user
- When the PAGE clause is specified:
  - Initializes this PAGE-COUNTER to 1
  - Initializes this LINE-COUNTER to 0
- Initializes the PRINT-SWITCH to 0

An INITIATE statement for a report must be executed after the OPEN statement for the report file, and before the first GENERATE statement for this report.

A second INITIATE statement for this report must not be executed before a subsequent TERMINATE statement for this report has been executed.

GENERATE STATEMENT

The GENERATE statement causes a report to be produced in the format specified in the Report Section.

Format

```
           ┌                ┐
           | data-name      |
GENERATE  <                  >
           | report-name    |
           └                ┘
```

Data-name must specify a TYPE DETAIL report group.

Report-name must specify the name of an RD entry. Report-name must be unique.

When the data-name option is specified, detail reporting is performed. When report-name is specified, summary reporting is performed. Both detail and summary reporting for the same report may be specified within one execution of a Report Writer program.

Detail Reporting

In detail reporting, at least one TYPE DETAIL report group must be specified. Execution of the GENERATE statement produces this TYPE DETAIL report group, and performs all the automatic operations of the Report Writer as specified in the report group description entries. (See "Automatic Operations of the GENERATE Statement," following.)

Summary Reporting

In summary reporting, the GENERATE statement performs all of the automatic operations of the Report Writer, as specified in the report group description entries, but does not produce any TYPE DETAIL report groups. (See "Automatic Operations of the GENERATE Statement" following.)

In summary reporting, a TYPE DETAIL report group need not be specified. (Note, however, that in this case sum counters are never incremented.)

If more than one TYPE DETAIL report group is specified, all SUM counters are algebraically incremented as if successive GENERATE statements were issued for each TYPE DETAIL report group—and in the order in which the TYPE DETAIL report groups appear in the report group description entries.

No matter how many TYPE DETAIL report groups are present, only one control break test is made; the test is made before summary reporting is performed.

Automatic Operations of the GENERATE Statement

When the GENERATE statement is executed, the following actions take place.

For the first GENERATE statement in the program:

1. REPORT HEADING report group is produced.

2. PAGE HEADING report group is produced.

3. CONTROL HEADING FINAL through CONTROL HEADING minor report groups are produced, in this order.

4. For detail reporting only, the specified TYPE DETAIL report group is produced.

(Figure 73 shows the sequence of operations for the first GENERATE statement.)

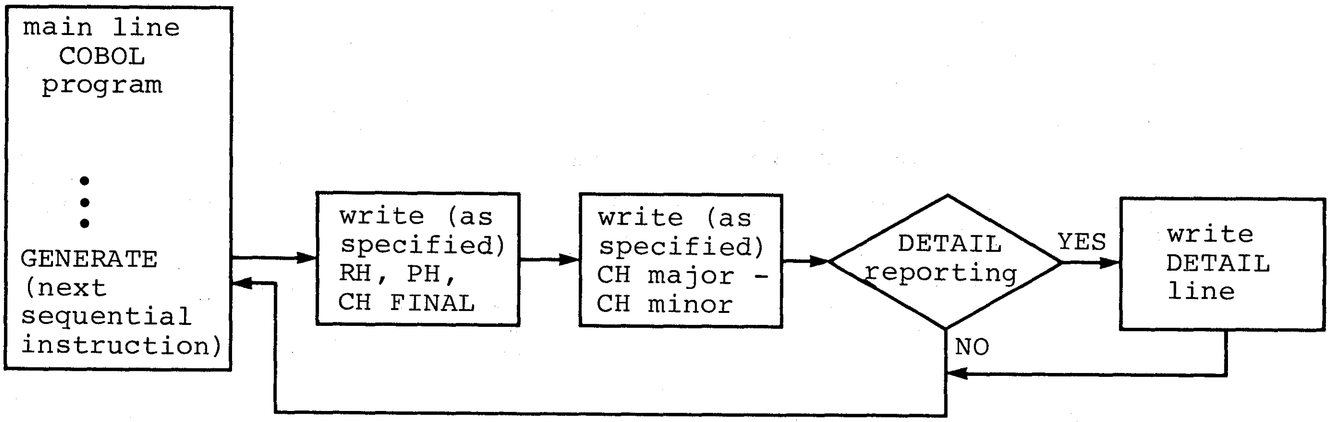

main line COBOL program
•
•
•
GENERATE (next sequential instruction)
write (as specified) RH, PH, CH FINAL
write (as specified) CH major - CH minor
DETAIL reporting
YES
NO
write DETAIL line

Figure 73. First GENERATE Statement—Sequence of Operations

For subsequent GENERATE statements in the program:

1. Appropriate CONTROL FOOTING and/or CONTROL HEADING report groups are produced upon recognition of the specified control breaks.

2. Any specified USE BEFORE REPORTING declarative procedures are executed before the associated report group is produced.

3. For detail reporting only, the specified TYPE DETAIL report group is produced.

4. If the PAGE clause is specified, the LINE-COUNTER for this report is incremented after each report group is produced. Before the next report group is produced, the LINE-COUNTER is tested, and, when necessary, PAGE FOOTING and/or PAGE HEADING report groups are produced, and PAGE-COUNTER is incremented.

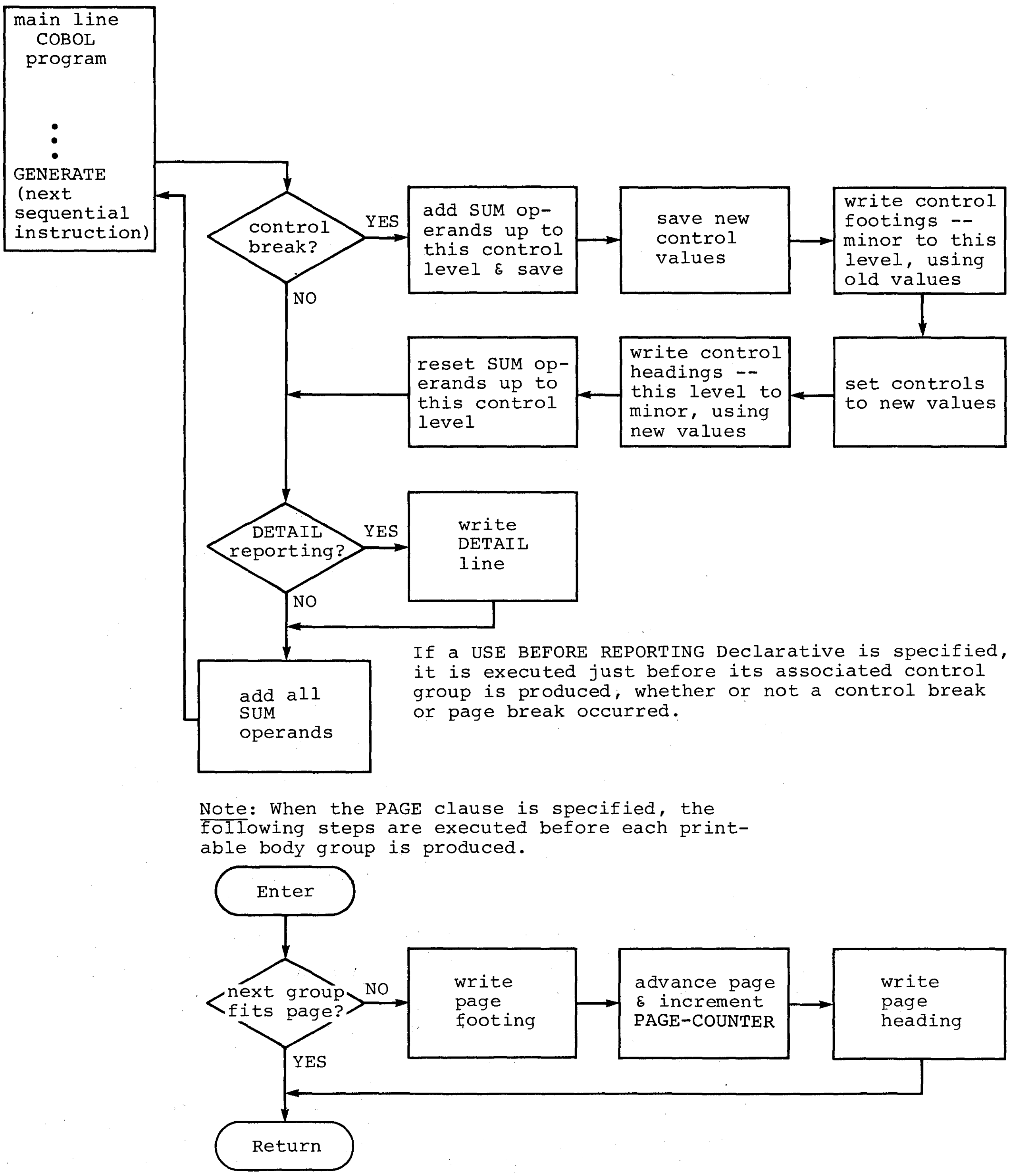
5. SUM operands are incremented.

6. Sum counters are reset as specified.

Figure 74 shows the sequence of operations for subsequent GENERATE statements.

When a control break is recognized, control groups (when specified) are produced in the order specified in the CONTROL clause.

Under Report Writer control, data is moved into the report group entry and is edited according to the rules for the MOVE statement (see MOVE Statement description in "Procedure Division").


main line COBOL program
•
•
•
GENERATE (next sequential instruction)
control break?
YES
NO
add SUM operands up to this control level & save
save new control values
write control footings -- minor to this level, using old values
set controls to new values
write control headings -- this level to minor, using new values
reset SUM operands up to this control level
DETAIL reporting?
YES
NO
write DETAIL line
add all SUM operands
Enter
next group fits page?
NO
YES
write page footing
advance page & increment PAGE-COUNTER
write page heading
Return

If a USE BEFORE REPORTING Declarative is specified, it is executed just before its associated control group is produced, whether or not a control break or page break occurred.

Note: When the PAGE clause is specified, the following steps are executed before each printable body group is produced.

Figure 74. Subsequent GENERATE Statements—Sequence of Operations

TERMINATE STATEMENT

The TERMINATE statement completes the processing of a report.

Format

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

Each report-name must be defined by an RD entry in the Data Division Report Section.

Execution of the TERMINATE statement does the following:

- Adds all SUM operands, and saves their values
- Saves current values of all control items
- Produces all CONTROL FOOTING report groups as if a control break at the highest level has just occurred
- Resets all control items to their current values (that is, their values at the time the TERMINATE statement is executed)
- Produces the CONTROL FOOTING FINAL report group
- Produces the specified PAGE FOOTING and/or PAGE HEADING report groups if a page break has occurred
- Produces the REPORT FOOTING report group

(Any SOURCE items specified in CONTROL FOOTING FINAL or REPORT FOOTING report groups refer to their values at the time the TERMINATE statement is executed.)

Figure 75 shows the sequence of operations for the TERMINATE statement.

A TERMINATE statement must be executed to complete the processing of a report. A second TERMINATE statement for this report must not be executed before a subsequent INITIATE statement for this report has been executed.

The TERMINATE statement does not close the report file; a CLOSE statement must be issued after the TERMINATE statement is executed.

If, at object time, no GENERATE statement is executed for this report, the TERMINATE statement produces no report groups, and does not perform SUM processing.

Because it forces and processes control breaks, a TERMINATE statement accesses all the data items that were specified as controls for the report. Also, in preparing headings and footings for presentation as a result of the forced control breaks, TERMINATE accesses the data items specified as SOURCE or SUM operands in those headings and footings. Therefore, the control data items and SOURCE/SUM operands must be available at the time the TERMINATE statement is executed.

For example, if the control or SOURCE/SUM data items are in a data record of a file or in a sort record, the record must be available when TERMINATE is executed. Input areas should not be used as control variables, because input areas are not available until following end-of-file. It is the user's responsibility to ensure that such data items are available.

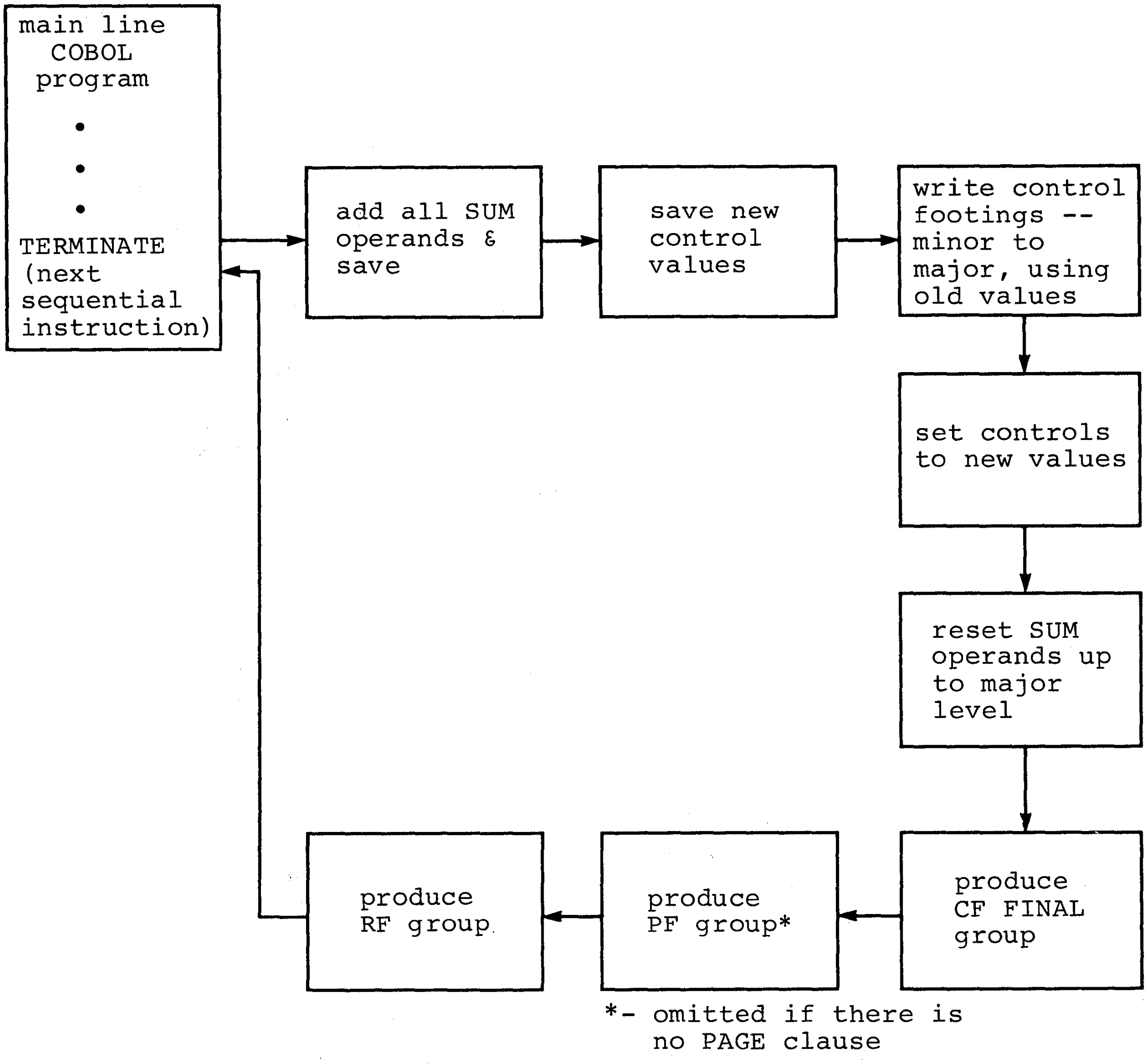

main line
COBOL
program
•
•
•
TERMINATE
(next
sequential
instruction)
add all SUM operands & save
save new control values
write control footings -- minor to major, using old values
set controls to new values
reset SUM operands up to major level
produce CF FINAL group
produce PF group*
produce RF group
*- omitted if there is no PAGE clause

Figure 75. TERMINATE Statement—Sequence of Operations

USE BEFORE REPORTING SENTENCE

The USE BEFORE REPORTING sentence identifies Declarative procedures to be executed just before a specified heading or footing report group is produced.

Format

USE BEFORE REPORTING data-name.

The USE BEFORE REPORTING sentence, when specified, must immediately follow a section header in the Declaratives portion of the Procedure Division (see the description of Declaratives in "Procedure Division"), and must immediately be followed by a period followed by a space.

The remainder of this declaratives section consists of one or more procedural paragraphs that define the procedures to be executed.

The USE BEFORE REPORTING sentence is never executed; it simply defines the condition that causes execution of this Declaratives section.

Data-name identifies a heading or footing report group, and must not appear in more than one USE BEFORE REPORTING sentence.

The procedures specified in this Declaratives section are executed just before the data-name report group is produced, regardless of any associated page or control breaks.

The Declaratives procedure may consist of any COBOL procedural statements, with the following restrictions:

- No Report Writer statement (INITIATE, GENERATE, TERMINATE) may be specified.

- There must be no reference to any nondeclarative procedures.

In the nondeclaratives portion of the Procedure Division, PERFORM statements may refer to procedure-names in the Declaratives portion; no other references to Declaratives procedures may be made.

Print Suppression

The PRINT-SWITCH allows the user to suppress printing of a specific execution of a specific heading or footing report group.

Format

MOVE 1 TO PRINT-SWITCH

In a Declaratives procedure, when MOVE 1 TO PRINT-SWITCH is specified, this report group is not printed when the statement is encountered; the statement must be executed for each presentation of each report group whose printing is to be suppressed.

When MOVE 1 TO PRINT-SWITCH is encountered, the following actions take place:

- This execution of this report group is not printed.

- The LINE-COUNTER is not altered.

- If a NEXT GROUP clause appears in this report group, its action is nullified.

- After the previous actions are taken, PRINT-SWITCH is reset to 0.

REPORT WRITER SPECIAL REGISTERS

Special registers PAGE-COUNTER and LINE-COUNTER are numeric counters generated by the Report Writer, and do not require data description clauses. Both PAGE-COUNTER and LINE-COUNTER can be used as SOURCE data items for elementary report group entries.

One PAGE-COUNTER and one LINE-COUNTER is generated for each report whose RD entry contains a PAGE clause. If there is more than one such report in a program, each reference to PAGE-COUNTER or a LINE-COUNTER must be qualified by the report-name.

Internally, both PAGE-COUNTER and LINE-COUNTER are 3-byte COMPUTATIONAL-3 items. When printed, each is produced to conform to the PICTURE clause of the entry for which it is a SOURCE item.

PAGE-COUNTER

The PAGE-COUNTER special register can be used to present the page number on a report line.

PAGE-COUNTER can be used as a SOURCE data item for elementary report group description entries. If more than one PAGE-COUNTER is so used, the PICTURE clauses of the report group description entries must be identical, and large enough to prevent overflow.

PAGE-COUNTER can be referred to, and its value can be modified in Procedure Division statements.

After an INITIATE statement, PAGE-COUNTER contains the value 1, and each time a page break is recognized it is automatically incremented by 1 after all PAGE FOOTING report groups are produced and before all PAGE HEADING report groups are produced.

If the user wants report pages to begin at a value other than one, the user can change the contents of PAGE-COUNTER after the INITIATE Statement is executed.

LINE-COUNTER

The Report Writer uses the LINE-COUNTER special register to determine when PAGE HEADING and/or PAGE FOOTING report groups are to be presented.

LINE-COUNTER can be used as a SOURCE data item in elementary report group entries.

LINE-COUNTER can be referred to, and its value can be modified, in Procedure Division statements. Note, however, that if the value in LINE-COUNTER is modified, Report Writer page format control may become unpredictable.

After an INITIATE statement, LINE-COUNTER contains the value 0. LINE-COUNTER is automatically tested and incremented by the Report Writer according to PAGE clause specifications, and the values specified in LINE and NEXT GROUP clauses.

The value contained in LINE-COUNTER during any Procedure Division test represents either:

- The last line produced by a previously generated report group, or

- The number of the last line skipped to by a previous NEXT GROUP clause specification

During execution, each time a page break occurs, LINE-COUNTER is reset to 0, except when an absolute LINE NUMBER or NEXT GROUP specification is less than or equal to the LINE-COUNTER value. When this occurs, after execution of the specified report groups a page break occurs and LINE-COUNTER is reset to the value specified in the absolute LINE or NEXT GROUP clause.

Successive report groups or report lines cannot be printed on the same line of the same page.

REPORT WRITER SAMPLE PROGRAM

This program is run after the close of the last working day of each month of the year. It supplies a record of purchases made by the Acme Manufacturing Company during the current year. Detail purchases for the current month are reported. For previous months, only a summary of each day's purchases is reported. In addition, a summary of the total number and amount of purchases is presented for each month, each quarter, and for the year to date.

INFILE, the input file, is an indexed VSAM file of the current year's purchases ordered on the invoice number of each purchase, with alternate ordering both on the date of purchase and on department. (This same file is used for report programs other than the one illustrated here.) INFILE is created via a remote terminal from the comptroller's office; after each purchase order is signed by the comptroller, details of the purchase are added to the file from the terminal.

REPORT-FILE, the output file, is a physical sequential file, assigned to the printer.

```
000005  IDENTIFICATION DIVISION.
000010  PROGRAM-ID. ACME.
000015  REMARKS. THE REPORT WAS PRODUCED BY THE REPORT WRITER.
000020  ENVIRONMENT DIVISION.
000025  CONFIGURATION SECTION.
000030  SOURCE-COMPUTER.  IBM-370.
000035  OBJECT-COMPUTER.  IBM-370.
000040  INPUT-OUTPUT SECTION.
000045  FILE-CONTROL.
000050      SELECT INFILE ASSIGN TO SYS123
000055          ORGANIZATION IS INDEXED
000060          ACCESS IS DYNAMIC
000065          RECORD KEY IS ORDER-NO PASSWORD IS DEPT-PASS
000070          ALTERNATE RECORD KEY IS DATE-PURCHASE
000075              WITH DUPLICATES
000080          ALTERNATE RECORD KEY IS DEPT
000085              WITH DUPLICATES
000090          FILE STATUS IS SK.
000095      SELECT REPORT-FILE ASSIGN TO SYS124.
000100  DATA DIVISION.
000110  FILE SECTION.
000115  FD  INFILE
000120      LABEL RECORDS ARE STANDARD
000125      DATA RECORD IS INPUT RECORD.
000130  01  INPUT-RECORD.
000135      05  ORDER-NO  PIC  X(6).
000140      05  DEPT      PIC  X(3).
000145      05  DATE-PURCHASE.
000150          10  MM    PIC  99.
000155          10  DD    PIC  99.
000160          10  YY    PIC  99.
000165      05  NO-PURCHASES  PIC 99.
000170      05  TYPE-PURCHASE PIC X.
000175      05  COST          PIC 999V99.
000180      05  FILLER        PIC X(57).
000185  FD  REPORT-FILE
000190      LABEL RECORDS ARE OMITTED
000195      RECORD CONTAINS 121 CHARACTERS
000200      REPORT IS EXPENSE-REPORT.
000205  WORKING-STORAGE SECTION.
000210  77  SAVED-MONTH        PIC 9(2)   VALUE ZERO.
000215  77  CONTINUED          PIC X(11)  VALUE SPACES.
000220  77  QUARTER-END        PIC X(7)   VALUE SPACES.
000225  77  MONTH-END          PIC X(7)   VALUE SPACES.
000230  77  DEPT-PASS          PIC X(8)   VALUE SPACES.
000235  77  QUARTER            PIC 9      VALUE ZERO.
000240  77  QRT-PASS           PIC 9      VALUE ZERO.
000245  77  REM                PIC 9      VALUE ZERO.
000250  77  SK                 PIC 9(2)   VALUE ZERO.
000255  77  PRINT-HDG-SW       PIC X(1)   VALUE SPACE.
```

```
000260  01  TODAYS-DATE.
000265      05  YR            PIC 99.
000270      05  MO            PIC 99.
000275      05  DA            PIC 99.
000280  01  QUARTER-NAMES VALUE "1ST2ND3RD4TH".
000285      05  QUARTERNAME  PIC X(3) OCCURS 4.
000290  01  DATE-USED.
000295      05  THIS-CENT    PIC XX.
000300      05  THIS-YEAR    PIC XX.
000305      05  FILLER       PIC XXX VALUE " IS ".
000310  01  MONTH-TABLE-1.
000315      05  RECORD-MONTH.
000320          10  FILLER  PIC X(9)   VALUE  "JANUARY  ".
000325          10  FILLER  PIC X(9)   VALUE  "FEBRUARY ".
000330          10  FILLER  PIC X(9)   VALUE  "MARCH    ".
000335          10  FILLER  PIC X(9)   VALUE  "APRIL    ".
000340          10  FILLER  PIC X(9)   VALUE  "MAY      ".
000345          10  FILLER  PIC X(9)   VALUE  "JUNE     ".
000350          10  FILLER  PIC X(9)   VALUE  "JULY     ".
000355          10  FILLER  PIC X(9)   VALUE  "AUGUST   ".
000360          10  FILLER  PIC X(9)   VALUE  "SEPTEMBER".
000365          10  FILLER  PIC X(9)   VALUE  "OCTOBER  ".
000370          10  FILLER  PIC X(9)   VALUE  "NOVEMBER ".
000375          10  FILLER  PIC X(9)   VALUE  "DECEMBER ".
000380      05  RECORD-AREA REDEFINES RECORD-MONTH.
000385          10  MONTHNAME  PIC X(9) OCCURS 12 TIMES.
000390  REPORT SECTION.
000395  RD  EXPENSE-REPORT
000400      CONTROLS ARE FINAL QUARTER MM DD
000405      PAGE LIMIT IS 59 LINES
000410      HEADING 1
000415      FIRST DETAIL 9
000420      LAST DETAIL 52
000425      FOOTING 56.
000430  01  TYPE IS REPORT HEADING.
000435      05  LINE NUMBER IS 2
000440          COLUMN NUMBER IS 27
000445          PIC X(26) VALUE "ACME MANUFACTURING COMPANY".
000450      05  LINE NUMBER IS 3
000455          COLUMN NUMBER IS 26
000460          PIC X(27) VALUE "RUNNING EXPENDITURES REPORT".
000465  01  PAGE-HEAD
000470      TYPE IS PAGE HEADING.
000475      05  LINE NUMBER IS 6.
000480          10  COLUMN 30 PIC X(9)  SOURCE MONTHNAME (MM).
000485          10  COLUMN 39 PIC X(13) VALUE "EXPENDITURES".
000490          10  COLUMN 53 PIC X(11) SOURCE CONTINUED.
000495      05  LINE IS 8.
000500          10  COLUMN  2  PIC X(35)
000505              VALUE  "MONTH      DAY   DEPT  NO-PURCHASES".
000510          10  COLUMN  40 PIC X(33)
000515              VALUE  "TYPE  COST CUMULATIVE-COST".
000520  01  DETAIL-LINE TYPE IS DETAIL LINE NUMBER IS PLUS 1.
000525      05  COLUMN  2  PIC X(9) SOURCE MONTHNAME (MM)
000530          GROUP INDICATE.
000535      05  COLUMN 13  PIC 9(2) SOURCE IS DD
000540          GROUP INDICATE.
000545      05  COLUMN 19  PIC X(3)   SOURCE IS DEPT.
000550      05  COLUMN 31  PIC Z9(1)  SOURCE IS NO-PURCHASES.
000555      05  COLUMN 42  PIC X(1)   SOURCE IS TYPE-PURCHASE.
000560      05  COLUMN 50  PIC ZZ9.99 SOURCE IS COST.
000565  01  TYPE IS CONTROL FOOTING DD.
000570      05  LINE NUMBER IS PLUS 2.
000575          10  COLUMN 2 PIC X(22)
000580              VALUE "PURCHASES AND COST FOR".
000585          10  COLUMN 24  PIC Z9    SOURCE SAVED-MONTH.
000590          10  COLUMN 26  PIC X     VALUE "-".
000595          10  COLUMN 27  PIC 99    SOURCE DD.
000600          10  COLUMN 30  PIC ZZ9   SUM NO-PURCHASES.
000605          10  MIN
000610              COLUMN 49  PIC $$$9.99 SUM COST.
000615          10  COLUMN 65  PIC $$$9.99 SUM COST
000620              RESET ON FINAL.
000625      05  LINE PLUS 1 COLUMN 2 PICTURE X(71)
```

```
000630          VALUE ALL "*".
000635  01  MM-FOOT TYPE CONTROL FOOTING MM
000640      LINE PLUS 1 NEXT GROUP NEXT PAGE.
000645      05  COLUMN 16    PIC X(14) VALUE "TOTAL COST FOR".
000650      05  COLUMN 31    PIC X(9)  SOURCE MONTHNAME (MM).
000655      05  COLUMN 43    PIC X(7)  SOURCE MONTH-END.
000660      05  INT
000665          COLUMN 50    PIC $$$$9.99 SUM MIN.
000670  01  QUARTER-FOOT TYPE CONTROL FOOTING QUARTER
000675          LINE PLUS 4 NEXT GROUP NEXT PAGE.
000680      05  COLUMN  5   PIC X(10)  VALUE ALL "*".
000685      05  COLUMN 16   PIC X(15)
000690          VALUE "TOTAL COST FOR ".
000695      05  COLUMN 31   PIC X(3)
000700          SOURCE QUARTERNAME (QUARTER).
000705      05  COLUMN 34   PIC X(9)
000710          VALUE "QUARTER".
000715      05  COLUMN 43   PIC X(7)
000720          SOURCE QUARTER-END.
000725      05  COLUMN 51   PIC $$$$9.99
000730          SUM INT.
000735  01  FINAL-FOOT TYPE CONTROL FOOTING FINAL
000740          LINE NEXT PAGE.
000745      05  COLUMN 16   PIC X(20)
000750          VALUE  "TOTAL COST FOR YEAR ".
000755      05  COLUMN 36   PIC X(8)
000760          SOURCE  DATE-USED.
000765      05  COLUMN 44   PIC $$$$9.99
000770          SUM INT.
000775  01  TYPE PAGE FOOTING LINE 57.
000780      05  COLUMN 59   PIC X(12) VALUE "REPORT-PAGE-".
000785      05  COLUMN 71   PIC 9(3) SOURCE PAGE-COUNTER.
000790  01  TYPE REPORT FOOTING
000795      LINE PLUS 1 COLUMN 32 PIC X(13)
000800      VALUE "END OF REPORT".
000805  PROCEDURE DIVISION.
000810  DECLARATIVES.
000815  PAGE-HEAD-RTN SECTION.
000820      USE BEFORE REPORTING PAGE-HEAD.
000825  PAGE-HEAD-RTN-TEST.
000830      IF MM NOT = SAVED-MONTH
000835         MOVE "(CONTINUED)" TO CONTINUED
000840      ELSE
000845         MOVE SPACES TO CONTINUED
000850         MOVE MM TO SAVED MONTH
000855* TO SUPRESS PRINTING OF HEADINGS FOR QUARTER AND
000856* FINAL FOOTING
000860      IF PRINT-HDG-SW = "N"
000865      MOVE SPACE TO PRINT-HDG-SW
000870      MOVE 1 TO PRINT-SWITCH.
000875  PAGE-HEAD-RTN-EXIT.
000880      EXIT.
000890  MM-FOOT-RTN SECTION.
000895      USE BEFORE REPORTING MM-FOOT.
000900  MM-FOOT PROCESS.
000905      IF YR > YY
000910         MOVE "WAS"      TO MONTH-END
000915      ELSE
000920         IF MO > MM
000925            MOVE "WAS"  TO MONTH-END
000930         ELSE
000935            MOVE "TO DATE" TO MONTH-END.
000940  MM-FOOT-RTN-EXIT.
000945      EXIT.
000950  QUARTER-FOOT-ROUTINE SECTION.
000955      USE BEFORE REPORTING QUARTER-FOOT.
000960  QUARTER-FOOT-PROCESS.
000965      IF LINE-COUNTER > 52
000970         MOVE "N"  TO PRINT-HDG-SW.
000975      IF YR > YY
000980         MOVE "WAS    " TO QUARTER-END
000985      ELSE
000990         IF (QUARTER = QRT-PASS) and (MM NOT = MO)
000995            MOVE "WAS"  TO QUARTER-END
```

```
001000         ELSE
001005            MOVE "TO DATE"  TO QUARTER-END.
001010 QUARTER-FOOT-PROCESS-END.
001015     EXIT.
001020 FINAL-FOOT-ROUTINE SECTION.
001025     USE BEFORE REPORTING FINAL-FOOT.
001030 FINAL-FOOT-PROCESS.
001035     IF YR > YY
001040        MOVE "19" TO THIS-CENT
001045        MOVE "YY" TO THIS-YEAR
001050     ELSE MOVE "TO DATE" TO DATE-USED.
001051     MOVE "N" TO PRINT-HDG-SW.
001055 FINAL-FOOT-PROCESS-END.
001060     EXIT.
001065 END DECLARATIVES.
001070 NONDECLARATIVES SECTION.
001075 BEGIN-PROCESS.
001080     ACCEPT DEPT-PASS FROM SYSIN.
001085     ACCEPT TODAYS-DATE FROM DATE.
001090     OPEN INPUT INFILE OUTPUT REPORT-FILE.
001095     IF SK NOT = "00" GO TO EXCEPTION-2.
001100     START INFILE KEY NOT LESS THAN DATE-PURCHASE.
001105     INITIATE EXPENSE-REPORT.
001110 READATA.
001115     READ INFILE NEXT.
001120     IF SK NOT = "00" GO TO EXCEPTION-1.
001125     IF MM NOT = SAVED-MONTH
001130         PERFORM QUARTER-CALC-1 THRU QUARTER-CALC-END.
001135**   THE FOLLOWING STATEMENTS COMMENTED OUT     **
001136**   CAN BE INCLUDED                            **
001140**   IF A SUMMARY FOR A DAY'S PURCHASES IS**
001141**   PREFERABLE TO AN                           **
001145**   INDIVIDUAL LISTING OF THESE PURCHASES.     **
001146**   THE SEQUENCE OF                            **
001150**   PROCESSING FOR DETAIL REPORTING IS**
001151**   SLIGHTLY DIFFERENT                         **
001155**   FROM THAT OF SUMMARY REPORTING. HENCE,     **
001156**   THE DECLARATIVE                            **
001160**   SECTION MAY HAVE TO BE ALTERED TO ALLOW    **
001161**   FOR USE OF THE                             **
001165**   SUMMARY REPORTING.                         **
001170*    IF MM = MO
001175*        ALTER SWITCH-PARA TO PROCEED TO DETAIL-REPORT.
001180* SWITCH-PARA.
001185*    GO TO SUMMARY-REPORT.
001190* SUMMARY-REPORT.
001195*    GENERATE EXPENSE-REPORT.
001200*    GO TO READATA.
001205 DETAIL-REPORT.
001210     GENERATE DETAIL-LINE.
001215     GO TO READATA.
001220 QUARTER-CALC-1.
001225     DIVIDE MM  BY 3 GIVING REMAINDER REM.
001230     MOVE QUARTER TO QRT-PASS.
001235     IF REM NOT = 0
001240        ADD 1 TO QUARTER.
001245 QUARTER-CALC-END.
001250     EXIT.
001255 EXCEPTION-1.
001260     MOVE "N" TO PRINT-HDG-SW.
001265     TERMINATE EXPENSE-REPORT.
001270     IF SK = "10" GO TO END-PROGRAM.
001275     DISPLAY "INPUT RECORD = " INPUT-RECORD UPON CONSOLE.
001280 EXCEPTION-2.
001285     DISPLAY "STATUS KEY = " SK UPON CONSOLE.
001290     DISPLAY "SAVE CONSOLE SHEET" UPON CONSOLE.
001295 END-PROGRAM.
001300     CLOSE INFILE REPORT-FILE.
001305     STOP RUN.
```

End of IBM Extension

SEGMENTATION FEATURE

The Segmentation feature provides programmer-controlled storage optimization of the Procedure Division, by allowing that division to be subdivided, both physically and logically.

SEGMENTATION CONCEPTS

Although it is not required, the Procedure Division of a source program is usually written as a consecutive group of sections, each of which is made up of a series of related operations that perform a particular function. Because of this the entire Procedure Division is made up of a number of logical subdivisions. Segmentation allows the programmer to physically divide the Procedure Division into segments, each of which has specific physical and logical attributes.

When Segmentation is used, the entire Procedure Division must be divided into sections. Each section must then be classified as to its physical and logical attributes, through a system of priority-numbers. All sections given one priority-number make up one program segment.

Priority numbers must be in the range from 0 through 99.

PROGRAM SEGMENTS

There are three types of program segments: fixed permanent, fixed overlayable, and independent.

Fixed Segments

A fixed permanent segment is always physically present during object program execution (that is, it cannot be overlaid, except when the system is executing another program). A fixed segment, thus, is always made available in its last-used state.

A fixed overlayable segment is logically always in storage during object program execution; therefore, it is always available in its last-used state. Any overlay of such a segment is transparent to the user. Thus, a fixed overlayable segment is logically identical with a fixed permanent segment.

Fixed permanent segments and fixed overlayable segments make up the fixed portion; that is, that part of the Procedure Division which is logically treated as if it were always physically present in virtual storage. Fixed-portion priority-numbers must be in the range 0 through 49.

Independent Segments

Logically, an independent segment can overlay and be overlaid by other segments during object program execution.

An independent segment is made available in its initial state the first time control is passed to it (explicitly or implicitly) during object program execution.

An independent segment is made available in its initial state during subsequent transfers of control when:

- The transfer is the result of an implicit transfer of control between consecutive statements from a segment with a different priority-number (that is, when control drops through into the independent segment from the physically preceding segment).

- The transfer is the result of an implicit transfer to a SORT input procedure or SORT/MERGE output procedure in an independent segment from a SORT or MERGE statement in a segment with a different priority.

- An explicit transfer of control from a section with a different priority-number takes place (as, for example, during the transfer of control in a PERFORM n TIMES statement).

An independent segment is made available in its last-used state during subsequent transfers of control when:

- With the two preceding exceptions, an implicit transfer from a section with a different priority takes place (as, for example, when control is returned to the independent segment from a Declarative procedure).

- An explicit transfer results from an EXIT PROGRAM statement.

IBM Extension

- An explicit transfer results from a GOBACK statement.

End of IBM Extension

Independent segments must be assigned priority-numbers in the range from 50 through 99.

Note: The segmentation rules for control transfers to and from independent segments as previously implemented are given under "PERFORM Statement in Segmented Programs" in Appendix A, "IBM Full ANS COBOL Items—(IBM Extension)."

SEGMENTATION LOGIC

In a segmented program, the sections are classified by a system of priority-numbers, using the following criteria:

- Frequency of Reference—Much-used sections, or those which must be available for reference at all times, should usually be within fixed permanent segments. Less frequently used sections should usually be within either fixed overlayable or independent segments, depending on the program logic.

- Frequency of Use—The more frequently a section is referred to, the lower its priority-number; the less frequently it is referred to, the higher its priority-number.

- Logical Relationships—Sections which frequently communicate with each other should be given identical priority-numbers.

SEGMENTATION CONTROL

Except for specific transfers of control, the logical sequence and the physical sequence of program instructions is the same. When segmentation is used, reordering of the object module may be necessary to handle the flow of control from segment to segment, since a given segment may have its sections scattered throughout the Procedure Division of the source program.

When the object module is thus reordered, the compiler provides control transfers to maintain the logic flow of the source program. The compiler also inserts any instructions necessary to initialize a segment.

It is not necessary to transfer control to the beginning of a segment, or to the beginning of a section within a segment. Instead, control may be transferred to any paragraph in the Procedure Division.

COBOL SOURCE PROGRAM CONSIDERATIONS

Two elements of a COBOL source program implement the Segmentation feature:

1. The SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph of the Environment Division. This clause allows the programmer to control the specification of fixed permanent and fixed overlayable segments.

2. Procedure Division priority-numbers, which group sections into segments. The priority-numbering scheme also allows specification of independent segments, fixed permanent segments, and (in conjunction with the SEGMENT-LIMIT Clause) fixed overlayable segments.

SEGMENTATION—ENVIRONMENT DIVISION

In the OBJECT-COMPUTER paragraph, the SEGMENT-LIMIT clause allows the user to reduce the number of fixed permanent segments, while retaining the properties of fixed portion segments.

Format

SEGMENT-LIMIT IS priority-number

The SEGMENT-LIMIT clause allows the programmer to specify certain permanent segments as capable of being overlaid by independent segments while still retaining the logical properties of fixed portion segments.

Priority-number must be an integer ranging in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified:

- Fixed permanent segments are those with priority-numbers ranging from up to, but not including, the specified priority-number.

- Fixed overlayable segments are those with priority-numbers ranging from that specified through 49, inclusive.

For example, if SEGMENT-LIMIT IS 25 is specified, sections with priority-numbers 0 through 24 are fixed permanent segments, and sections with priority-numbers 25 through 49 are fixed overlayable segments.

When the SEGMENT-LIMIT clause is omitted, all sections with priority-numbers from 0 through 49 are fixed permanent segments.

SEGMENTATION—PROCEDURE DIVISION

In the Procedure Division of a segmented program, section classification is specified through priority-numbers in the section headers.

Format

section-name SECTION [priority-number].

All sections with the same priority-number make up one program segment. Such sections need not be contiguous in the source program.

The priority-number must be an integer in the range from 0 through 99.

Segments with priority-numbers 0 through 49, inclusive, are in the fixed portion of the object program. Declarative sections may be assigned only to these priority-numbers.

Segments with priority-numbers in the range from 50 through 99, inclusive, are independent segments.

If the priority-number is omitted from the section header, the priority-number is assumed to be 0.

SEGMENTATION—SPECIAL CONSIDERATIONS

When segmentation is used, there are restrictions on the ALTER, PERFORM, MERGE, and SORT statements.

There are also special considerations for calling and called programs.

These are all detailed below.

ALTER STATEMENT

A GO TO statement in an independent segment must not be referred to by an ALTER statement in a different segment. All other uses of the ALTER statement are valid and are performed, even if the GO TO statement referred to is in a fixed overlayable segment.

PERFORM STATEMENT

A PERFORM statement in the fixed portion can have in its range, in addition to any Declarative procedures whose execution is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs contained within a single independent segment

A PERFORM statement in an independent segment can have within its range, in addition to any Declarative procedures whose execution is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs wholly contained in the same independent segment as the PERFORM statement

IBM Extension

However, this IBM implementation allows any PERFORM statement to have sections with any priority-numbers within its range of procedures.

End of IBM Extension

SORT AND MERGE STATEMENTS

If a SORT or MERGE statement appears in the fixed portion, then any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- In one independent segment

If a SORT or MERGE statement appears in an independent segment, then any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- The same independent segment as the SORT or MERGE statement

IBM Extension

However, this IBM implementation allows a SORT or MERGE statement to have sections with any priority-numbers within an input and/or output procedure.

End of IBM Extension

CALLING AND CALLED PROGRAMS

The CALL statement may appear anywhere within a segmented program. When a CALL statement appears in an independent segment, that segment is in its last-used state when control is returned to the calling program.

A called program may be segmented.

SOURCE PROGRAM LIBRARY

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures without recoding them. These entries and procedures can then be saved in user-created libraries; they can then be included in the source program by means of the COPY statement.

COPY STATEMENT

The COPY statement places prewritten text in a COBOL program.

Format

```
                     ┌ OF ┐
COPY text-name [     <    >  library-name]
                     │ IN │
                     └    ┘
    ┌──────────┐
    │[SUPPRESS]│
    └──────────┘
                     ┌                  ┐
                     │ ==pseudo-text-1==│
    [REPLACING   ┌   │ identifier-1     │
                 <   <                  >
                 └   │ literal-1        │
                     │ word-1           │
                     └                  ┘

                     ┌                  ┐
                     │ ==pseudo-text-2==│
                     │ identifier-2     │ ┐
              BY     <                  > >   ... ].
                     │ literal-2        │ ┘
                     │ word-2           │
                     └                  ┘
```

Compilation of the source program containing COPY statements is logically equivalent to processing all COPY statements before processing the resulting source program.

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 word COPY and ending with the period, inclusive. When the REPLACING option is not specified, the library text is copied unchanged.

The text-name must follow the rules for formation of a program-name. These first eight characters of the text-name are used as the identifying name; these first eight characters must therefore be unique within one COBOL library.

If more than one COBOL library is available during compilation, text-name must be qualified by the library-name of the library on which it resides. See "System Dependencies."

The library-name must follow the rules for formation of a program-name. The first eight characters of the library-name are used as the identifying name; these first eight characters must therefore be unique within the system.

The uniqueness of the text-name and the library-name is determined after the formation and conversion rules for a program-name have been applied. (These rules are given in the PROGRAM-ID Paragraph of the Identification Division chapter.)

Each COPY statement must be preceded by a space and ended with a period followed by a space.

A COPY statement may appear in the source program anywhere a character string or a separator may appear; however, a COPY statement must not be specified within a COPY statement. The resulting copied text must not contain a COPY statement.

Comment lines may appear in library text. Comment lines in library text are copied into the source program unchanged, and are interpreted logically as a single space.

Debugging lines may appear in library text.

When a COPY statement is specified on a debugging line, the copied text is treated as though it appeared on a debugging line, except that comment lines in the library text appear as comment lines in the resulting source program.

Because 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.

Library text copied from the library is placed into the same area of the resultant program as it is in the library. Library text must conform to the rules for standard COBOL format.

Note: Because copy processing uses column one for control characters, characters outside the standard character set should not appear in column one of copy members.

IBM Extension

SUPPRESS Option

The SUPPRESS option specifies that the library text is not to be printed on the source program listing.

Note: The SUPPRESS option has no effect when using LISTER (LSTC,LSTO) or FIPS (LVL=).

End of IBM Extension

REPLACING Option

In the discussion that follows, each <u>operand</u> may consist of one of the following: pseudo-text, an identifier, a literal, or a COBOL word. When the REPLACING option is specified, each operand-1 from the library text is replaced by its associated operand-2.

PSEUDO-TEXT: Pseudo-text is a sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters (==). Both characters of each pseudo-text delimiter must appear on one line; however, character-strings within pseudo-text can be continued. Because of the replacement rules, the continued line of pseudo-text-2 can begin in Area A.

Note: A character string for pseudo-text is a complete COBOL word; the prefix portion of a data name, for example, cannot be replaced using pseudo-text unless the entire data name is used. Figure 76 shows an example of this.

```
Library-member A consists of the following COBOL text:

Area
A    Area
|    B
|    |
|    |
v    v

X.   10 AA-ID    PIC X(9).


A COPY statement that replaces data description AA-ID with
data description BB-ID, would be written as follows:

Area
A    Area
|    B
|    |
|    |
v    v           COPY A REPLACING == AA-ID == BY ==

BB-ID == .
```

Figure 76. Pseudo-Text Continuation Lines

Pseudo-text-1, which is the library text, must not be null, nor may it consist solely of the space character and/or of comment lines. Beginning and ending blanks are not included in the text comparison process. Embedded blanks are used in the text comparison process to indicate multiple text elements.

Pseudo-text-2 may be null; it may consist solely of space characters and/or comment lines. Each word in pseudo-text-2 that is to be copied into the program is placed in the same area of the resultant program as it appears in pseudo-text-2.

IDENTIFIER: Each identifier may be defined in any Data Division section.

LITERAL: Each literal may be numeric or nonnumeric.

COBOL WORD: Each COBOL word may be any single COBOL word.

REPLACING OPTION PROCESSING: When the REPLACING option is specified, the library text is copied, and each properly matched occurrence of operand-1 within the library text is replaced by the associated operand-2.

For purposes of matching, each identifier-1, literal-1, or word-1 is treated, respectively, as pseudo-text containing only identifier-1, literal-1, or word-1.

Notes:

1. Arithmetic and logical operators are considered to be text words and may be replaced only through the pseudo-text option.

2. When a figurative constant is operand-1, it will match only exactly as specified. For example, if ALL "AB" is specified in the library text, then "ABAB" is not considered a match; only ALL "AB" is considered a match.

3. When replacing a PICTURE character-string, the pseudo-text option must be used; to avoid ambiguities, pseudo-text-1 must specify the entire PICTURE clause, including the key word PICTURE or PIC.

The comparison proceeds as follows:

- Any separator comma, semicolon, and/or space preceding the leftmost word in the library text is copied into the source program. Beginning with the leftmost library text word and the first operand-1 specified in the REPLACING option, the entire REPLACING operand that precedes the key word BY is compared to an equivalent number of contiguous library text words.

- Operand-1 matches the library text if, and only if, the ordered sequence of text words in operand-1 is equal, character for character to the ordered sequence of library words. For matching purposes, each occurrence of a comma or semicolon separator is considered to be a single space; however, when operand-1 consists solely of a separator comma or semicolon it participates in the match as a text word (in this case, the space following the comma or semicolon separator may be omitted). 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 successive operand-1, if specified, until either a match is found or there are no further REPLACING operands.

- Whenever a match occurs between operand-1 and the library text, the associated operand-2 is copied into the source program.

- When all operands have been compared and no match is found, the leftmost library text word is copied into the source program.

- The next successive uncopied library text word is then considered the leftmost text word, and the comparison process is repeated, beginning with the first operand-1. The process continues until the rightmost library text word has been compared.

- A comment line occurring in operand-1 and in the library text is interpreted for matching purposes as a single space. A comment line appearing in operand-2 is copied unchanged into the source program.

- Debugging lines are not permitted in operand-1. Debugging lines, however, are permitted in library text and in operand-2. Text words in a debugging line are matched as if no D appeared in column 7.

- Text words after replacement are placed in the source program according to standard COBOL format rules.

4. There is an established limit of 150 COPY replacing argument pairs that may be used in any source program. When this limit is exceeded, a message is issued and the COPY statement is ignored.

COPY Statement Example

In this example, the library entry PAYLIB consists of the following Data Division entries:

```
02  B    PIC S99.
02  C    PIC S9(5)V99.
02  D    PIC S9999 OCCURS 1 TO 52 TIMES
    DEPENDING ON B OF A.
```

The programmer can use the COPY statement in the Data Division of a program as follows:

```
01  PAYROLL.
    COPY PAYLIB.
```

In this program, the library entry is then copied; the resulting entry is treated as if it had been written as follows:

```
01    PAYROLL.
    02  B    PIC S99.
    02  C    PIC S9(5)V99.
    02  D    PIC S9999 OCCURS 1 TO 52 TIMES
        DEPENDING ON B OF A.
```

To change some (or all) of the names within the library entry, the programmer can use the REPLACING option:

```
01    PAYROLL.
      COPY PAYLIB REPLACING A BY PAYROLL
        B BY PAY-CODE C BY GROSS-PAY.
```

In this program, the library entry is copied; the resulting entry is treated as if it had been written as follows:

```
01    PAYROLL.
    02  PAY-CODE    PIC S99.
    02  GROSS-PAY   PIC S9(5)V99.
    02  D           PIC S9999 OCCURS 1 TO 52 TIMES
        DEPENDING ON PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The entry as it appears in the library remains unchanged.

Note: The COPY statement as previously implemented is discussed in Appendix A.

Programming Notes

Sequences of code (such as file and data descriptions, error and exception routines, and so forth) that are common to a number of programs can be cataloged, and then used in conjunction with the COPY statement. If naming conventions are established for such common code, then the REPLACING option need not be specified. If the names will change from one program to another, then the REPLACING option can be used to supply meaningful names for this program.

IBM Extension

EXTENDED SOURCE PROGRAM LIBRARY

A complete program may be included as an entry in a user's library and may be used as the source for a compilation. Compiler input is a BASIS statement, optionally followed by any number of INSERT and/or DELETE statements.

BASIS STATEMENT

The BASIS statement provides a complete COBOL program as the source for a compilation.

Format

[sequence-number] BASIS basis-name

A sequence-number followed by a space may, optionally, appear in columns 1 through 7 of the card-image statement.

The word BASIS followed by basis-name may appear anywhere in columns 1 through 72. There must be no other text in the statement.

The basis-name must follow the rules for formation of a program-name; it is the name by which the library entry is known to the control program. The first eight characters are used as the identifying name, and must therefore be unique within this library.

INSERT/DELETE STATEMENTS

The INSERT statement allows the programmer to add COBOL statements to the BASIS source program. The DELETE statement allows the programmer to remove COBOL statements from and add COBOL statements to the BASIS source program.

Format

```
                    ┌ INSERT ┐
[sequence-number]   <        >  sequence-number-field
                    │ DELETE │
                    └        ┘
```

A sequence-number followed by a space may optionally appear in columns 1 through 7 of the card-image statement.

The word INSERT or DELETE, followed by a space, followed by the sequence-number-field may appear anywhere within columns 1 through 72. There must be no other text in the statement.

Each number in the sequence-number-field must be equal to a sequence number in the BASIS source program. This sequence number is the 6-digit number the programmer assigns in columns 1 through 6 of the COBOL coding form.

The numbers specified must be in ascending numeric order from the first INSERT/DELETE statement to the last INSERT/DELETE statement in the program.

INSERT Statement

The sequence-number-field must be a single number (for example, 000130). At least one new source program statement must follow the INSERT statement for insertion after the statement number specified by the sequence-number-field.

DELETE Statement

The sequence-number-field must be any one of the following:

- A single number

- A series of single numbers

- A range of numbers (indicated by separating the two bounding numbers of the range by a hyphen)

- A series of ranges of numbers

- Any combination of one or more single numbers and one or more ranges of numbers

Each entry in the sequence-number-field must be separated from the preceding entry by a comma followed by a space. For example:

```
000250 DELETE  000010-000050, 000400, 000450
```

Notes:

1. Source program statements may follow a DELETE Extended Source Program Library statement: These source statements are then inserted into the BASIS source program before the statement following the last statement deleted. (That is,

in the example above, before the next statement following deleted statement 000450.)

2. If a DELETE statement is specified beginning in column 12 or higher, and valid sequence numbers do not follow the key word DELETE, the compiler assumes that this DELETE statement is the COBOL DELETE verb.

End of IBM Extension

SUBPROGRAM LINKAGE FEATURE

Complex data processing problems are often solved by the use of separately compiled but logically interdependent programs which, at execution time, form logical and physical subdivisions of a single run unit. (A run unit is the total machine-language program necessary to solve a data processing problem; it includes one or more object programs, and may include object programs from source languages other than COBOL.)

SUBPROGRAM LINKAGE CONCEPTS

When the solution to a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other through transfers of control and/or through reference to common data.

CONTROL TRANSFERS

In the Procedure Division, a calling program can transfer control to a called program, and a called program may itself transfer control to yet another called program. However, a called program must not directly or indirectly call its caller (such as program A calling program B; program B calling program C; and program C then calling program A) or results are unpredictable.

When control is passed to a called program, execution proceeds in the usual way, as described in the Procedure Division chapter. When called program processing is completed, the program can either transfer control back to the calling program, call another program, or end the run unit.

COMMON DATA

Program interaction may require that both programs have access to the same data.

In a calling program the common data items are described in the same manner as other File, Working-Storage, or Communication Section items: storage is allocated for these items in the calling program.

In a called program, common data items are described in the Linkage Section; storage is not allocated to them in the called program. (Because a calling program may itself be a called program, common data items may be described in the calling program's Linkage Section, in which case storage is not allocated for these items in the calling program itself, but rather in the program that called the calling program. That is, program A calls program B which calls program C. Data items in program A can be described in the Linkage Sections of programs B and C, and the one set of data can be made available to all three programs.)

When control is transferred from the calling to the called program, the programmer must furnish a list of the common data items in both programs. The sequence of identifiers in both lists determines the match of identifiers between the calling and called programs; that is, a corresponding pair of identifiers in the list names a single set of data that is available to both programs. While the called program is executing, any reference to one of these identifiers is a reference to the calling program's corresponding data.

COBOL LANGUAGE CONSIDERATIONS

In the Data Division of the source programs, the programmer defines the common data items to be used by both the calling and called programs. In the calling program, these items can be defined in the File, Working-Storage, Communications, or Linkage Sections. In the called program, these items must be defined in the Linkage Section. Common data items need not have the same name and data description; they must contain the same number of characters.

In the Procedure Division, the list of common data items is established through the USING option, which names those data items available to both programs. In the called program, only those items named in the called program's USING list are available from the calling program's data storage. Figure 77 illustrates this concept.

Calling Program Description

```
WORKING-STORAGE SECTION.
01  PARAM-LIST.
    05  PARTCODE  PIC A.
    05  PARTNO    PIC X(4).
    05  U-SALES   PIC 9(5).
        .
        .
        .
PROCEDURE DIVISION.
        .
        .
        .
    CALL CALLED-PROG
        USING PARAM-LIST.
```

In the calling program, the code for parts (PARTCODE) and the part number (PARTNO) are referred to separately.

Called Program Description

```
LINKAGE SECTION.
01  USING-LIST.
    10  PART-ID   PIC X(5).
    10  SALES     PIC 9(5).
        .
        .
        .
PROCEDURE DIVISION
     USING   USING-LIST.
```

In the called program, the code for parts and the part number are combined into one data item (PART-ID); in the called program, a reference to PART-ID is the only valid reference to them.

Figure 77. Common Data Items in Subprogram Linkage

A CALL statement in the calling program transfers control to the first nondeclarative procedural statement in the called program. When the called program has completed execution, control is returned to the calling program by an EXIT PROGRAM statement. The entire run unit can be ended by a STOP RUN statement.

IBM Extension

A CALL statement in the calling program can transfer control to an alternative entry point in the called program. The alternative entry point is identified by an ENTRY statement in the called program. When the CALL statement is executed, control is transferred to the next executable statement following the ENTRY statement.

Control can also be returned to the calling program by a GOBACK statement.

A called program can be returned to its initial state by a CANCEL statement for that program; the next CALL statement to that called program then makes it available in its initial state. In addition, the CALL statement makes provision for transferring control to one or more called programs whose names are unknown at compile time, and also for determining the object-time availability of storage for the called program.

End of IBM Extension

SYSTEM CONSIDERATIONS

STATIC CALL STATEMENT: In the CALL statement, the main COBOL program and all called programs are part of the same load module. When control is transferred to the called program, it is already resident in storage, and a branch to the called program takes place. Subsequent executions of the CALL statement make the called program available in its last-used state.

IBM Extension

If alternative entry points are specified, any CALL statement can use any alternative entry point to enter the called subprogram.

End of IBM Extension

DYNAMIC CALL STATEMENT: In this form of the CALL statement, the called subprogram is not link-edited with the main program, but is instead link edited into a separate load module, and at execution time is loaded only if and when it is required (that is, when called).

Each subprogram invoked with a dynamic CALL statement may be part of a different load module, which is a member of the system link library or of a user-supplied private library. The execution of the dynamic CALL statement to a subprogram that is not resident in storage results in the loading of that subprogram from secondary storage into the region/partition containing the main program and a branch to the subprogram.

Thus, the first dynamic CALL to a subprogram obtains a fresh copy of the subprogram. Subsequent calls to the same subprogram (by either the original caller or by any other subprogram within the same region/partition) result in a branch to the same copy of the subprogram in its last-used state. However, when a CANCEL statement is issued for that subprogram, the storage occupied by the subprogram is freed, and a subsequent CALL to the subprogram will function as though it were the first. A CANCEL statement referring to a called subprogram may be issued by a program other than the original caller.

IBM Extension

If the called subprogram has more than one entry point, differing entry points must not be specified in the CALL statement until an intervening CANCEL statement has been executed.

Note: See the System Dependencies chapter on the CALL statement. p. 360

End of IBM Extension

For further information on specification of static and dynamic CALL statements, see the System Dependencies chapter.

SUBPROGRAM LINKAGE—DATA DIVISION

In the Data Division of a called program, the programmer specifies in the Linkage Section those data items that are common with the calling program.

Format

```
LINKAGE SECTION

[ 77    ]
<       >  data-name/FILLER Clause
| 01-49 |
L       J

    [REDEFINES Clause]
    [BLANK WHEN ZERO Clause]
    [JUSTIFIED Clause]
    [OCCURS Clause]
    [PICTURE Clause]
    [SIGN Clause]
    [SYNCHRONIZED Clause]
    [USAGE Clause].

 [88  condition-name VALUE Clause.]

 [66  RENAMES Clause.]
```

The Linkage Section has meaning if, and only if, this object program functions under control of a CALL statement that contains the USING option.

The Linkage Section describes data available within the calling program and referred to in both the calling and called programs. Items described in the Linkage Section do not have space allocated for them in this 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. For index-names, no such correspondence is established; index-name references in the calling and called programs always refer to separate indexes.

Items defined in the Linkage Section can be referred to in the Procedure Division only if they are one of the following:

- Operands of a USING option in this program
- Data items subordinate to such a USING option operand
- Items associated with such a USING operand (such as condition-names or index-names)

Because neither can be qualified, each Linkage Section record-name and noncontiguous data-name must be unique, Descriptions of each valid clause in the Linkage Section are given in the Data Description Entry chapter. The following additional considerations apply.

RECORD DESCRIPTION ENTRIES

Items that have a hierarchic relationship with one another must be grouped into 01-level records according to the rules for formation of record descriptions. Data description clauses can be used to complete the description of the entry. Except for level-88 condition names, the VALUE clause must not be specified.

See the System Dependencies chapter.

DATA ITEM DESCRIPTION ENTRIES

Items that have no hierarchic relationship with each other can be defined as noncontiguous items with level-number 77. The following clauses are required:

- Level-number 77
- Data-name
- PICTURE or USAGE IS INDEX

Other data description clauses are optional, and, when necessary, can be used to complete the description of the item. Except for level-88 condition names, the VALUE clause must not be specified.

See the System Dependencies chapter.

SUBPROGRAM LINKAGE—PROCEDURE DIVISION

In the Procedure Division, control is transferred between COBOL object programs by means of the CALL statement.

Reference to common data is provided through the USING option, which can be specified in the CALL statement and in the called program's Procedure Division header.

IBM Extension

The ENTRY statement makes provision for alternative entry points into a called program; the USING option can also be specified in the ENTRY statement of a called program.

The GOBACK statement allows termination of called program processing or of main COBOL program processing.

End of IBM Extension

The EXIT PROGRAM statement allows termination of called program processing. The STOP RUN statement allows termination of the run unit.

The CANCEL statement releases storage used by a called subprogram.

Because it is common to several Subprogram Linkage Feature elements, the USING option is discussed before the other Procedure Division statements.

USING OPTION

The USING option makes data items in a calling program available to a called program.

Format

USING identifier-1 [identifier-2] ...

The USING option is valid for the CALL statement in a calling program, and in the Procedure Division header of a called program entered at the beginning of the nondeclaratives portion.

In a calling program, the USING option is valid for the CALL statement; each USING identifier must be defined as a level-01 or level-77 item anywhere in the Data Division.

IBM Extension

However, this IBM implementation allows each USING identifier in a calling program to be a data item anywhere in the Data Division except the Report Section.

End of IBM Extension

In a called subprogram entered at the beginning of the nondeclaratives portion, the USING option is valid in the Procedure Division header; each USING identifier must be defined as a level-01 or level-77 item in the Linkage Section of the called subprogram.

IBM Extension

In a called subprogram entered at the first executable statement following an ENTRY statement, the USING option is valid in the ENTRY statement; each USING identifier must be defined as a level-01 or level-77 item in the Linkage Section of the called subprogram.

End of IBM Extension

Formats for these individual items show the correct syntax for specifying the USING option.

The USING option is specified if, and only if, the called subprogram is to operate under control of a CALL statement and that CALL statement itself contains a USING option. That is, for each CALL USING statement in a calling program, there must be a corresponding USING option specified in a called subprogram.

The order of appearance of USING identifiers in both calling and called programs determines the correspondence of single sets of data available to both programs. The correspondence is positional and not by name. Corresponding identifiers must contain the same number of characters, although their data descriptions need not be the same. For index-names, no correspondence is established; index-names in calling and called programs always refer to separate indexes.

The identifiers specified in a CALL USING statement name data items available to the calling program that may be referred to in the called program; a given identifier may appear more than once. These items are defined in any Data Division section.

IBM Extension

However, this IBM implementation allows the data items named by the identifiers specified in a CALL USING statement to be defined anywhere in the Data Division except the Report section.

End of IBM Extension

In a called subprogram, USING identifiers must be defined in the Linkage Section. Of those items defined in the Linkage Section, only those named in the USING option are available to the object program. Within the called subprogram, USING identifiers are processed according to their definition within this program.

When the USING option is specified, the object program executes as if each reference to a USING identifier in the called program Procedure Division is replaced by a reference to the corresponding USING identifier in the calling program.

Examples illustrating these concepts are given in Figure 77 and in the section on Subprogram Linkage Feature Examples.

Note: The USING option must not be specified for a called subprogram that contains a CD entry for INITIAL input.

Further information is given in the chapter on System Dependencies.

CALL STATEMENT

The CALL statement causes control to be transferred from one object program to another within the run unit.

Format 1—Static CALL

```
CALL literal-1 [USING identifier-1 [identifier-2] ...]
```

Format 2—Dynamic CALL

```
        ┌ literal-2     ┐
CALL    <               >
        │ identifier-3  │
        └               ┘

     [USING identifier-1 [identifier-2] ... ]
     [ON OVERFLOW imperative-statement]
```

Each literal must be nonnumeric and must conform to the rules for formation of program-names (see PROGRAM-ID Paragraph in the Identification Division chapter). The first eight characters of the literal are used to make the correspondence between the calling program and the called program.

When the called subprogram is to be entered at the beginning of the Procedure Division, the literal must specify the program-name of the called subprogram.

IBM Extension

When the called subprogram is entered through an ENTRY statement, the literal must be the same as that specified in the called subprogram's ENTRY statement. This literal must not be the called subprogram's program-name but must follow the rules for formation of a program-name.

End of IBM Extension

Rules for USING option identifiers are given in the preceding section on the USING option.

The program containing the CALL statement is the calling program; the program identified in the CALL statement is the called subprogram.

Called subprograms may contain CALL statements; however, a called subprogram must not contain a CALL statement that directly or indirectly calls the calling program.

CALL statement execution causes control to pass to the called subprogram. The first time a called program is entered, its state is that of a fresh copy of the program. Each subsequent time a called program is entered, the state is as it was upon the last exit from that program. Thus, the reinitialization of the following items is the responsibility of the programmer:

- GO TO statements that have been altered
- Data items
- PERFORM statements

See the System Dependencies chapter for further information.

Segmentation Considerations

A CALL statement may appear anywhere within a segmented program; the compiler ensures that the proper logic flow is maintained. Therefore, if a CALL statement appears in an independent segment, that segment is made available in its last-used state when control is returned from the called program.

CANCEL STATEMENT

The CANCEL statement releases the storage occupied by a called subprogram.

Format

```
       ┌               ┐   ┌               ┐
       │ literal-1     │   │ literal-2     │
CANCEL <               >   │               │ ...
       │ identifier-1  │   │ identifier-2  │
       └               ┘   └               ┘
```

Each literal or identifier specified in the CANCEL statement must be nonnumeric. The contents must conform to the rules for formation of a program-name (see PROGRAM-ID Paragraph in the Identification Division chapter.) The first eight characters of the literal or identifier are used to make the correspondence between the calling and called program.

Each literal or identifier specified in the CANCEL statement must be the same as the literal or identifier specified in the associated CALL statement(s).

For a program in which the CALL literal statement is static:

- The CANCEL literal statement is invalid.

- The CANCEL identifier statement is accepted, but the compiler then uses the COBOL Library Management Facility.

Subsequent to the execution of a CANCEL statement, the program referred to therein ceases to have any logical relationship to the program in which the CANCEL statement appears. A subsequently executed CALL statement by any program in the run unit naming the same program will result in that program being entered in its initial state.

A logical relationship to a canceled subprogram is established only by execution of a subsequent CALL statement.

A called subprogram is canceled either by being directly referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

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.

Called subprograms may contain CANCEL statements. However, a called subprogram must not contain a CANCEL statement that directly or indirectly cancels the calling program itself, or any other program higher than itself in the calling hierarchy. In such a case, the run unit is terminated.

A program named in a CANCEL statement must not refer to any program that has been called and has not yet returned control to the calling program. A program may, however, cancel a program that it did not call, if the calling hierarchy is higher than or equal to the program it is canceling. For example, A calls B, and B calls C; when A receives control, it can cancel C; or A calls B, and A calls C; when C receives control, it can then cancel B.

See the System Dependencies chapter for further information.

IBM Extension

ENTRY STATEMENT

The ENTRY statement establishes an alternative entry point into a COBOL called subprogram.

Format

ENTRY literal [USING identifier-1 [identifier-2] ...]

When a CALL statement naming the alternative entry point is executed in a calling program, control is transferred to the next executable statement following the ENTRY statement.

The literal must not be the name of the called subprogram, but it must follow the same rules of formation; at least one character must be alphabetic. The first eight characters are used to make the correspondence between the calling and the called program.

The literal must be nonnumeric; it must be unique within this run unit.

Execution of the called program begins at the first executable statement following the ENTRY statement whose literal corresponds to the CALL statement literal or identifier.

End of IBM Extension

EXIT PROGRAM STATEMENT

The EXIT PROGRAM statement specifies the logical end of a called program.

Format

paragraph-name. EXIT PROGRAM.

The EXIT statement must be preceded by a paragraph-name, and must be the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches an EXIT PROGRAM statement, and no CALL statement is active, control passes through the exit point to the first sentence of the next paragraph.

For further considerations, see the chapter on System Dependencies.

STOP RUN STATEMENT

The STOP RUN statement is discussed in the chapter on Procedure Branching Statements.

For Subprogram Linkage considerations, see the chapter on System Dependencies.

IBM Extension

GOBACK STATEMENT

The GOBACK statement specifies the logical end of a called program.

Format

GOBACK.

A GOBACK statement must appear as the only statement, or as the last of a series of imperative statements, in a sentence.

If control reaches a GOBACK statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

See the System Dependencies chapter for the action taken when a GOBACK statement is executed and no CALL statement is active.

End of IBM Extension

SUBPROGRAM LINKAGE FEATURE EXAMPLES

The Format-1 CALL statement (static call) is illustrated in the following program example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLSTAT.
     .
     .
     .
DATA DIVISION.
     .
     .
     .
WORKING-STORAGE SECTION.
01  RECORD-2  PIC X.
01  RECORD-1.
    05  SALARY       PICTURE S9(5)V99.
    05  RATE         PICTURE S9V99.
    05  HOURS        PICTURE S99V9.
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
    CALL "SUBPROG" USING RECORD-1.
     .
     .
     .
    CALL "PAYMASTR" USING RECORD-1 RECORD-2.
     .
     .
     .
    STOP RUN.
```

The Format 2 CALL statement (dynamic CALL) is illustrated in the following example, which achieves results different from the preceding example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLDYNA.
     .
     .
     .
DATA DIVISION.
     .
     .
     .
WORKING-STORAGE SECTION.
77  IDENT PICTURE X(8).
     .
     .
     .
01  RECORD-2  PIC X.
01  RECORD-1.
    05  SALARY       PICTURE S9(5)V99.
    05  RATE         PICTURE S9V99.
    05  HOURS        PICTURE S99V9.
     .
     .
     .
PROCEDURE DIVISION.
     .
     .
     .
    MOVE "SUBPROG" TO IDENT.
    CALL IDENT USING RECORD-1.
     .
     .
     .
    CANCEL IDENT
```

```
          .
          .
          .
        MOVE "PAYMASTR" TO IDENT.
        CALL IDENT USING RECORD-1 RECORD-2.
          .
          .
          .
    STOP RUN.
```

The following called subprogram can be associated with either of the two preceding calling programs.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
          .
          .
          .
DATA DIVISION.
          .
          .
          .
LINKAGE SECTION.
01  PAYREC.
    10  PAY            PICTURE S9(5)V99.
    10  HOURLY-RATE    PICTURE S9V99.
    10  HOURS          PICTURE S99V9.
77  CODE PIC 9.
          .
          .
          .
PROCEDURE DIVISION USING PAYREC.
          .
          .
          .
    EXIT PROGRAM.
    ENTRY "PAYMASTER" USING PAYREC CODE.
          .
          .
          .
    GOBACK.
```

Processing begins in the calling program—which may be either CALLSTAT or CALLDYNA. When the first CALL statement is executed, control is transferred to the first statement of the Procedure Division in SUBPROG, which is the called program.

Note that, in each of the CALL statements, the operand of the first USING option is identified as RECORD-1.

When SUBPROG receives control, the values within RECORD-1 are made available to SUBPROG; however, in SUBPROG they are referred to as PAYREC.

Note that the PICTURE character-strings within PAYREC and CODE contain the same number of characters as RECORD-1 and RECORD-2, although the descriptions are not identical.

When processing within SUBPROG reaches the EXIT PROGRAM statement, control is returned to the calling program. Processing continues in that program until the second CALL statement is issued.

Note that in CALLSTAT (statistically linked) the CANCEL statement would not be valid.

statically
?

IBM Extension

In CALLDYNA, however, because the second CALL statement refers to another entry point within SUBPROG, a CANCEL statement is issued before the second CALL statement.

With the second CALL statement in the calling program, control is again transferred to SUBPROG, but this time processing begins at the statement following the ENTRY statement in SUBPROG. The values within RECORD-1 and RECORD-2 are again made available to PAYREC. In addition, the value in RECORD-2 is now made available to SUBPROG through the corresponding USING operand CODE.

When processing reaches the GOBACK statement, control is returned to the calling program at the statement immediately following the second CALL statement.

Note that, when control is transferred the second time from CALLSTAT, SUBPROG is made available in its last-used state (that is, if any values in SUBPROG storage were changed during the first execution, those changed values are still in effect). When control is transferred from CALLDYNA, however, SUBPROG is made available in its initial state.

End of IBM Extension

In any given execution of these two programs, if the values within RECORD-1 are changed between the time of the first CALL and the second, the values passed at the time of the second CALL statement will be the changed, not the original, values. If the user wishes to use the original values, then they must be saved.

COMMUNICATION FEATURE

The communication feature allows the COBOL programmer to access, process, and create messages and portions of messages, and to control the flow of messages through a communications network. Communication with local and remote communications devices is through a Message Control System (MCS).

COMMUNICATION CONCEPTS

The Communication Feature permits the COBOL programmer to create device-independent message processing programs for data communication applications. A communication network consists of a central computer, remote or local station(s), and the communication lines connecting such station(s) to the central computer.

In communication applications, data flow into the system is random and proceeds at relatively slow speeds. Data in the system exists as messages from remote stations, or as messages generated by internal programs. Once delivered to the computer, the messages can be processed at computer speeds. Thus, communication applications require a Message Control System (MCS) that acts as an interface between the COBOL program and the remote stations.

The MCS acts as the logical interface between the entire network of communications devices and the COBOL program, in much the same manner as the operating system acts as an interface between the COBOL object program and conventional input/output devices. The MCS also must perform device-dependent tasks such as character translation, and insertion of control characters, so that the COBOL program itself is device-independent. The MCS and the COBOL communication program operate asynchronously; that is, there is no fixed time relationship between the receipt of a message by the MCS and its subsequent processing by the COBOL communication program.

The MCS has two constituent parts:

1. A user-written Telecommunications Access Method (TCAM) Message Control Program (MCP) coded in assembler language.

2. COBOL Communications Feature object-time subroutines.

QUEUE CONCEPTS

To store the messages until they are to be processed, the MCS uses message queues, which may be thought of as sequential data sets. The queues act as buffers between the COBOL communication program and the remote stations. To the COBOL communication program, the MCS queue from which it accepts messages is logically an input queue; the queue; into which it places messages is logically an output queue. In this publication, these terms have these meanings.

Input Queues and Queue Structures

Associated with each MCS input queue can be logical paths from any number of remote stations—in COBOL terms, symbolic sources. A symbolic source (remote station) may, conversely, have associated with it logical paths to any number of MCS input queues. COBOL uses symbolic queue structures to access the MCS input queues.

At execution time, the COBOL object program retrieves messages from the MCS through a system of predefined symbolic queue structures. Each queue structure is constructed like a tree and can consist of a named queue with up to three levels of subordinate named sub-queues. The queue structure need not include all three levels of sub-queues; however, if a lower level is specified, all higher levels for that branch of the structure must also be specified. In any branch of the structure, the lowest level sub-queue corresponds to an MCS input queue. Figure 78 is an example of a predefined input queue structure.

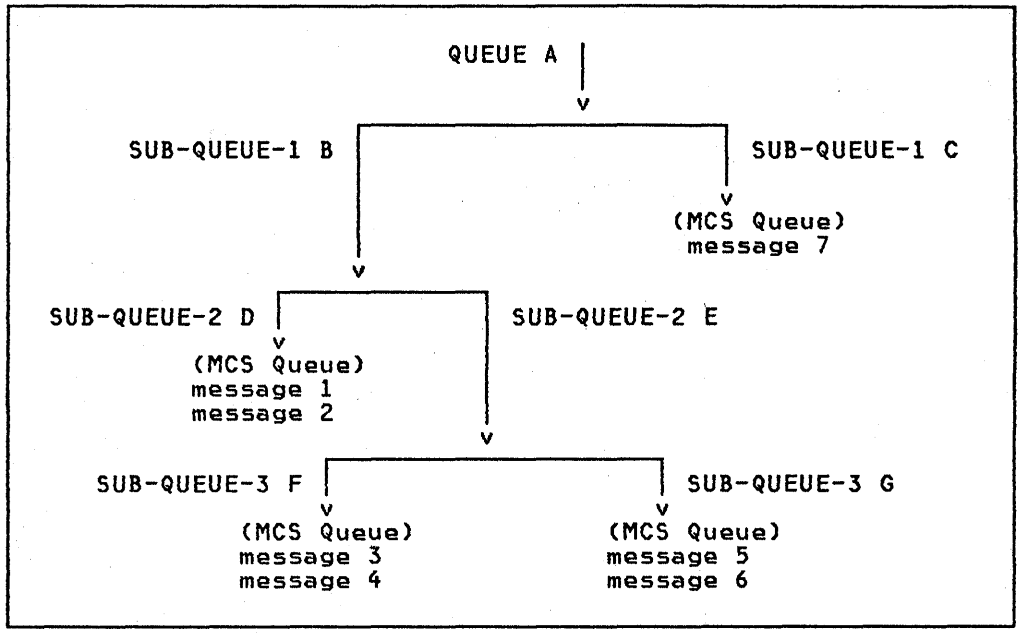


Figure 78. Predefined Input Queue Structure Example

The predefined queue structures are analogous in function and form to a file-processing program FD entry and its associated record description. As in a COBOL FD entry, a reference to a higher level includes an implicit reference to all levels subordinate to the named level. Thus, in the preceding example, a reference to QUEUE A is a reference to the entire queue structure (as a reference to a file-name in an FD entry is a reference to all record descriptions associated with the file). Similarly, a reference to SUB-QUEUE-2 E is also implicitly a reference to both SUB-QUEUE-3 F and SUB QUEUE-3 G. In this way, COBOL allows the programmer to specify which parts of a queue structure are to be accessed.

The messages in a queue structure are available in left-to-right order. In the preceding example, the following references make the indicated messages available:

- QUEUE A—messages 1 through 7, in ascending order
- SUB-QUEUE-1 B—messages 1 through 6, in ascending order
- SUB-QUEUE-2 E—messages 3 through 6, in ascending order
- SUB-QUEUE-1 C—only message 7

Output Queues

Associated with each MCS output queue is a logical path to a single symbolic destination (remote station). For this reason, the MCS output queue is not explicitly named, but is referred to implicitly whenever the symbolic destination is referred to. The COBOL program can refer to output queues through a destination table of symbolic destination names—the number of names accessed is controlled by the COBOL programmer.

DATA DIVISION CONCEPTS

The interface between COBOL and the MCS is established through the Communication Description (CD) entries in the Communication Section. If input communication operations are to be performed there must be at least one CD entry for input; if output communication operations are to be performed, there must be at least one CD entry for output. Multiple input and/or output CD entries are allowed. Level-01 data description entries (which implicitly redefine the CD area) may also be specified, following the CD entry.

Each CD entry is an implicitly-defined fixed storage area into which information about messages is placed for use by both the COBOL program and the MCS.

PROCEDURE DIVISION CONCEPTS

Five statements are used by the COBOL object program to request MCS services:

- ENABLE statement—which allows data transfer between the MCS and the communications network.

- DISABLE statement—which prevents data transfer between the MCS and the communications network.

- RECEIVE statement—which causes data in an MCS input queue associated with a specified queue structure to be passed to the COBOL object program.

- SEND statement—which causes data associated with the COBOL object program to be passed to one or more MCS output queues.

- ACCEPT MESSAGE COUNT statement—which causes the MCS to return to the COBOL object program the number of complete messages in the specified MCS input queues associated with the queue structure.

The RECEIVE, ACCEPT MESSAGE COUNT, and SEND statements control information flow between the COBOL object program and the MCS, while the ENABLE and DISABLE statements control message traffic between the MCS and the communications network. (Note that information flow between the COBOL object program and the MCS is _not_ affected by the ENABLE or DISABLE statements.)

Any completed messages existing in an MCS input queue prior to the disabling of that queue may still be accessed by RECEIVE and ACCEPT MESSAGE COUNT statements directed to that queue; however, any messages subsequently directed to that queue will be discarded.

Any completed messages existing in an MCS output queue at the time the associated symbolic destination is disabled will not be transmitted until the destination is reenabled; new messages may continue to be directed to a disabled destination and will be transmitted when the destination is reenabled.

The relationship between the COBOL object program, the MCS, and the remote stations is shown in Figure 79.

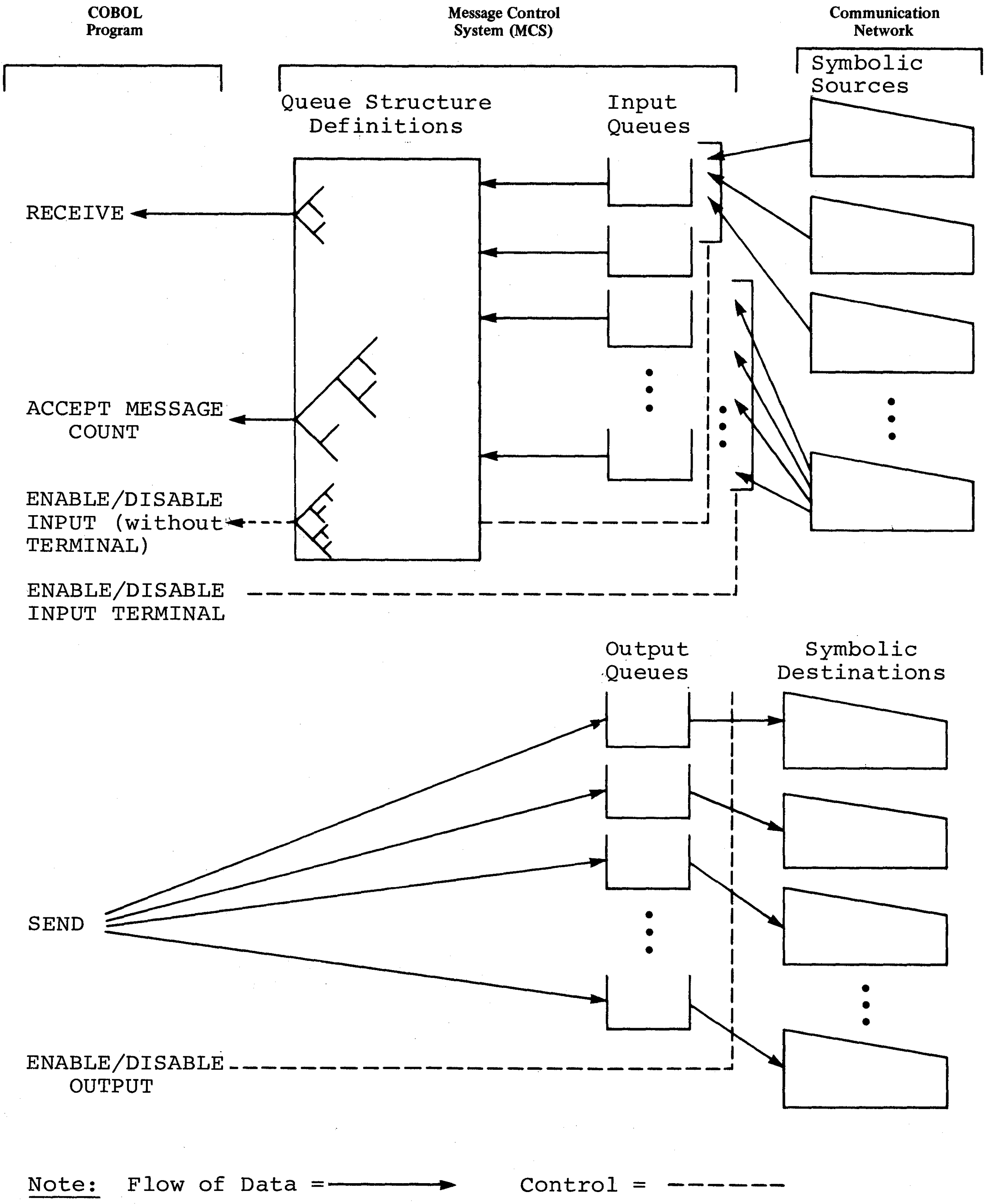

COBOL Program
Message Control System (MCS)
Communication Network
Symbolic Sources
Queue Structure Definitions
Input Queues
RECEIVE
ACCEPT MESSAGE COUNT
ENABLE/DISABLE INPUT (without TERMINAL)
ENABLE/DISABLE INPUT TERMINAL
Output Queues
Symbolic Destinations
SEND
ENABLE/DISABLE OUTPUT
Note: Flow of Data = ⟶ Control = ------

Figure 79. Communication Feature Relationships

OBJECT PROGRAM SCHEDULING

A COBOL communication program may be scheduled for execution through Job Control Language (JCL). It may also be scheduled for execution by the MCS. (In the latter case, the COBOL program is invoked only when there are messages available for it to process.)

A COBOL program can be written so that its object program can operate with either mode of scheduling. To determine which method of scheduling was used to invoke the COBOL object program, the following technique can be used:

- One CD FOR INITIAL INPUT must be specified.
- Procedure Division statements test the initial value of the symbolic queue and sub-queue names for that CD. If they are space-filled, the COBOL program was invoked by job control statements; if they contain data, the MCS invoked the COBOL program and placed the symbolic name of the queue containing messages into the symbolic queue and sub-queue name fields.

MESSAGE AND MESSAGE SEGMENT CONCEPTS

A message is an arbitrary amount of information whose start and end are defined or implied.

A message can be logically subdivided into message segments, which are delimited by end of segment indicators (ESI). A message is delimited from the next message by an end of message indicator (EMI). A group of messages can also be delimited from another group by an end of group indicator (EGI). The presence of these logical indicators is recognized and specified both by the MCS and the COBOL program; however, no indicators are included in the message text processed by COBOL programs. For incoming messages, the associated end indicators are identified in the CD entry area. For outgoing messages, the COBOL program specifies the end indicators to be associated with the message.

COMMUNICATION—DATA DIVISION

The Communication Section of a COBOL program must be specified if the program is to utilize the communication features of COBOL. The Communication Section, through the definition of Communication Description (CD) entries, establishes the interface between the COBOL object program and the MCS.

General Format

[COMMUNICATION SECTION.

{communication description entry

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

The Communication Section is identified by, and must begin with, the section header COMMUNICATION SECTION. The header is followed by Communication Description (CD) entries. Specification of the CD entry causes an implicitly defined data area to be created; that is, the generated data area has a fixed format. Level-01 record description entries may optionally follow the CD entry; these record description entries implicitly redefine the fixed data areas of the CD.

When it is specified, the Communication Section should contain at least one CD entry. A single CD entry is sufficient if messages are only of one type, that is, only FOR INPUT or only FOR OUTPUT. If the COBOL program is to both receive and send messages, then at least two CD entries are required—one FOR INPUT and one FOR OUTPUT. However, multiple input and/or output CD entries may be specified.

The CD entry is valid only in the Communication Section.

CD ENTRY

The CD entry is the highest level of organization in the Communication Section, and provides information about messages being processed by the COBOL object program.

Format 1—Option 1

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

Format 1—Option 2

```
CD  cd-name FOR [INITIAL] INPUT
    [data-name-1 data-name-2 ... data-name-11].
```

Format 2

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

The CD entry serves as a storage area through which the COBOL program and the MCS communicate. The COBOL programmer moves information about the operation to be performed into the CD before initiating any request. The MCS, after acting upon the request, returns through the same CD information pertaining to the operation.

The CD entry is defined in such a way that any number of message queues may be accessed through the same CD entry. Conversely, different portions of one message may be accessed through multiple CD entries in the same program or in different COBOL subprograms residing in the same region or partition. Thus, any COBOL communication program needs to specify only one input CD entry and/or one output CD entry. Rules controlling the accessing of MCS queues are specified in the detailed descriptions of both input (Format 1) and output (Format 2) CD entries.

The level indicator CD identifies the beginning of a Communication Description entry, and must appear in Area A. It must be followed in Area B by cd-name. cd-name follows the rules for formation of a data-name. cd-name may be followed by a series of optional independent clauses (as shown in Format 1 and Format 2).

Except for the INITIAL clause, the optional clauses of the CD entry may be written in any order. Because the data areas of both the input CD and the output CD have implicit definitions, the optional clauses are necessary only to assign user names for those areas to which the COBOL program will refer. However, if all the options of either format are omitted, then a level-01 record description entry must follow the CD entry.

The optional clauses may be followed by one or more level-01 record description entries; these record description entries implicitly redefine the fixed data area described by the CD entry. The total length of each record description entry must be the same as the fixed data descriptions of the CD entry. (Examples are given later in this section.)

If more than one such redefining record description entry is written, only the first may contain VALUE clauses.

Subordinate data description entries within the record description need not be the same length as the implicit data items described in the CD entry. However, the MCS always refers to this data area according to the implicit CD entry descriptions. That is, for an input CD, the contents of positions 1 through 12 are always used as the symbolic queue, the contents of positions 13 through 24 are always used as symbolic sub-queue-1, and so forth.

Format 1, Options 1 and 2—Input CD Entry

This format is required if the CD entry is FOR INPUT. At least one input CD entry must be specified if symbolic input queues or sources are to be referred to in the COBOL object program. Any number of queues or sources can be referred to through one input CD entry; to do this, the user simply places different queue or source names into the input CD.

Conversely, different portions of one message from the same symbolic queue may be accessed through different CD entries. Thus, CD entries in the same or different COBOL subprograms in the same run unit may be used to access different portions of the same message. The same CD entry may be used to access a message from another queue before the first message is completed. The following restrictions apply:

- Only one region (or partition) can have access to any particular queue at one time.

- The data in a queue must be accessed sequentially. That is, a second message in any queue cannot be accessed until the entire first message in that queue is accessed. However, a second message from another queue may be accessed before the entire message in the first queue is accessed.

The specification of an input CD entry results in a record whose implicit description is shown in Figure 80.

```
Equivalent COBOL Record Description    Description of Use

01  data-name-0.

     02  data-name-1  PICTURE X(12).   Symbolic Queue
     02  data-name-2  PICTURE X(12).   Sub-queue-1
     02  data-name-3  PICTURE X(12).   Sub-queue-2
     02  data-name-4  PICTURE X(12).   Sub-queue-3
     02  data-name-5  PICTURE 9(6).    Message Date
     02  data-name-6  PICTURE 9(8).    Message Time
     02  data-name-7  PICTURE X(12).   Symbolic Source
     02  data-name-8  PICTURE 9(4).    Text Length
     02  data-name-9  PICTURE X.       End Key
     02  data-name-10 PICTURE XX.      Status Key
     02  data-name-11 PICTURE 9(6).    Message Count
```

Figure 80. Input CD Entry—Implicit Description

For each input CD entry, a record area of 87 contiguous Standard Data Format characters is always generated, and implicitly defined as previously specified. The data names corresponding to the various fields of the CD record area may be explicitly defined, through the use of the optional clauses.

If MCS scheduling of the COBOL object program is desired, one CD FOR INITIAL INPUT must be specified; only one is allowed. When the MCS schedules the program, the SYMBOLIC QUEUE and (if applicable) SYMBOLIC SUB-QUEUE fields are updated to contain the names of the queue structure that caused the invocation. In all other cases, these fields of this input CD entry are initialized to spaces. These actions are completed before execution of the first Procedure Division statement.

Subsequent RECEIVE statement execution for this cd-name returns the message that caused this program to be scheduled. Only at this time is the remainder of the CD FOR INITIAL INPUT updated.

A CD FOR INITIAL INPUT must not be specified in a called subprogram that contains USING identifiers.

Format 1—Option 1

Data-names corresponding to the fields of the CD area may be explicitly defined. In one CD entry, each data-name must be unique.

SYMBOLIC QUEUE and SUB-QUEUE Clauses: These clauses define data-name-1, data-name-2, data-name-3, and data-name-4 as the names of alphanumeric data items each of 12 characters in length, and occupying character positions within the record as follows:

- data-name-1 occupies character positions 1 through 12
- data-name-2 occupies character positions 13 through 24
- data-name-3 occupies character positions 25 through 36
- data-name-4 occupies character positions 37 through 48

(See the chapter on System Dependencies.)

Before executing a statement that refers to a queue structure (RECEIVE, ACCEPT MESSAGE COUNT, ENABLE INPUT (without TERMINAL), or DISABLE INPUT (without TERMINAL)), the contents of data-name-1 must be set to the name of the queue structure, and the contents of data-name-2 through data-name-4 must be set to the names of any required sub-queues or to spaces. (Note that the compiler initializes the contents of data-name-1 through data-name-4 to spaces.) When a sub-queue name is specified, all higher-level sub-queue names must also be specified.

After execution of the ACCEPT MESSAGE COUNT, ENABLE INPUT (without TERMINAL), and DISABLE INPUT (without TERMINAL) statements, the contents of data-name-1 through data-name-4 are unchanged.

After execution of the RECEIVE statement, the contents of data-name-1 through data-name-4 are updated to contain the complete queue structure identification corresponding to the MCS input queue from which the message or message segment was retrieved. Thus, if message retrieval is incomplete, execution of subsequent RECEIVE statements using the same input CD will return successive portions of the same message. When a message from a less specific source is needed, the contents of data-name-1 through data-name-4 must be respecified.

For example: Referring to Figure 78, assume that data-name-1 contains A and data-name-2 through data-name-4 contain spaces before execution of a RECEIVE statement. After execution of the RECEIVE statement (which would access message 1), data-name-1 would contain A, data-name-2 would contain B, and data-name-3 would contain D.

MESSAGE DATE Clause: This clause defines data-name-5 as the name of an unsigned 6-digit integer data item, occupying character positions 49 through 54 of the record.

Data-name-5 has the format YYMMDD (year, month, day). Its contents represent the date on which the MCS received this message.

The contents of data-name-5 are updated by the MCS as part of the execution of each RECEIVE statement.

MESSAGE TIME Clause: This clause defines data-name-6 as the name of an unsigned 8-digit integer data item, occupying character positions 55 through 62 of the record.

Data-name-6 has the format HHMMSSTT (hours, minutes, seconds, hundredths of a second). Its contents represent the time of day the message was received in the system by the MCS.

The contents of data-name-6 are updated by the MCS as part of the execution of each RECEIVE statement.

SYMBOLIC SOURCE Clause: This clause defines data-name-7 as the name of an elementary alphanumeric data item of 12 characters, occupying character positions 63 through 74 of the record.

During execution of a RECEIVE statement, the MCS provides in data-name-7 the symbolic name of the source of this message. (The symbolic names the MCS uses are one through eight characters in length; the remaining characters are set to spaces.) However, if the symbolic name of the source is not known to the MCS, the contents of data-name-7 are set to spaces.

Before executing an ENABLE INPUT TERMINAL or DISABLE INPUT TERMINAL statement, the COBOL programmer must set the contents of data-name-7 to the symbolic name of the source to be enabled or disabled.

TEXT LENGTH Clause: This clause defines data-name-8 as the name of an unsigned 4-digit integer data item, occupying character positions 75 through 78 of the record.

The MCS indicates through the contents of data-name-8 the number of main storage bytes, if any, of the user's work area filled as a result of the execution of a RECEIVE statement.

END KEY Clause: This clause defines data-name-9 as the name of a 1-character elementary alphanumeric data item, occupying character position 79 of the record.

The MCS sets the contents of data-name-9, as part of the execution of each RECEIVE statement, according to the following rules:

- When RECEIVE MESSAGE is specified, then the contents of data-name-9 are:

  3 if end-of-group has been detected

  2 if end-of-message has been detected

  0 if less than a message has been moved into the user-specified area

- When RECEIVE SEGMENT is specified, then the contents of data-name-9 are:

  3 if end-of-group has been detected

  2 if end-of-message has been detected

  1 if end-of-segment has been detected

  0 if less than a message segment has been moved into the user-specified area.

- When more than one of the above conditions is satisfied simultaneously, the rule first satisfied in the order listed determines the contents of data-name-9. An end of group is a logical end of file condition caused by a user request in the MCS. In general, depending on the size of the work unit and the work area provided, End keys may be associated with a text length of 0. This is always the case for end of group.

Note: The MCS removes the end of transmission (EOT) line control character before presenting a message to a COBOL object program. Because the EBCDIC representation of EOT is hexadecimal "37", the last data character of a message should never be hexadecimal "37".

STATUS KEY Clause: This clause defines data-name-10 as the name of a 2-character elementary alphanumeric data item, occupying character positions 80 and 81 of the record.

The contents of data-name-10 indicate the status condition of the previously executed communication statement. The program should, therefore, check the contents of data-name-10 immediately after each communication operation to determine the success or failure of the operation. Note that the proper updating of other input CD fields and/or message input areas may depend on successful statement execution.

Figure 81 indicates the possible values that the STATUS KEY field (for both input and output CD entries) may contain at the completion of execution for each statement. An X on a line in a statement column indicates that the associated code on that line is possible for that statement.

STATUS KEY Value	Meaning	RE-CEIVE	SEND	ACCEPT MES-SAGE COUNT	ENABLE INPUT (without TERMI-NAL)	ENABLE INPUT (with TERMI-NAL)	ENABLE OUT-PUT	DISABLE INPUT (without TERMI-NAL)	DISABLE INPUT (with TER-MINAL)	DIS-ABLE OUTPUT
00	No error detected. Request completed.	X	X	X	X	X	X	X	X	X
10	1 or more destinations disabled. Action completed.		X							
20	1 or more destinations unknown. Action completed for known; none taken for unknown. ERROR KEY indicates known or unknown.		X				X			X
20	1 or more queues or subqueues unknown. No action taken.	X		X	X			X		
20	Source unknown. No action taken.					X			X	
30	DESTINATION COUNT content invalid. No action taken.		X				X			X
40	Password invalid. No action taken.				X	X	X	X	X	X
50	Character count greater than sending field. No action taken.		X							
60	Partial segment with 0 character count or no sending area specified. No action taken.		X							

Standard COBOL

STATUS KEY Value	Meaning	RE-CEIVE	SEND	ACCEPT MES-SAGE COUNT	ENABLE INPUT (without TERMI-NAL)	ENABLE INPUT (with TERMI-NAL)	ENABLE OUT-PUT	DISABLE INPUT (without TERMI-NAL)	DISABLE INPUT (with TER-MINAL)	DIS-ABLE OUTPUT
21	Insufficient storage for control blocks and/or buffers. Request canceled.	X	X	X	X	X	X	X	X	X
22	Required DD statement missing. Request canceled.		X		X	X	X	X	X	X
27	Queue full. Request canceled. Retry.		X							
28	Quick close down of communication network. Request canceled.		X							
29	I/O error has occurred. Request canceled.	X	X	X	X	X	X	X	X	X

IBM Extensions

Figure 81. STATUS KEY Field—Possible Values

MESSAGE COUNT Clause: This clause defines data-name-11 as the name of an unsigned 6-digit integer data item, occupying character positions 82 through 87 of the record.

The contents of data-name-11 indicate the number of complete messages that exist in an input queue structure. The MCS updates the contents of data-name-11 only as part of the execution of an ACCEPT MESSAGE COUNT statement.

Format 1—Option 2

The second option of Format 1 allows the programmer to specify data-name-1 through data-name-11 without the descriptive clauses. If any data-names are to be omitted, the word FILLER must be substituted for each omitted name, except that FILLER need not be specified for any data-name that occurs after the last name to be referred to.

For example, if the programmer wishes to refer to the SYMBOLIC QUEUE as QUEUE-IN and to the MESSAGE DATE as DATE-IN, the input CD entry can be written as follows:

```
CD  INPUT-AREA FOR INPUT
   QUEUE-IN FILLER FILLER FILLER DATE-IN.
```

In this case, data-name-6 through data-name-11 can be omitted without FILLER being written in their place.

The same input CD entry can be written as follows (in this case, an optional level-01 record description entry redefining the data areas is also included):

```
CD   INPUT-AREA FOR INPUT
     SYMBOLIC QUEUE IS QUEUE-IN
     MESSAGE DATE IS DATE-IN.
01   INAREA-RECORD.
     05  FILLER       PICTURE X(78).
     05  ENDKEY-CODE  PICTURE X(1).
         88  PARTIAL-SEGMENT  VALUE "0."
         88  END-SEGMENT      VALUE "1."
         88  END-MESSAGE      VALUE "2."
         88  END-GROUP        VALUE "3."
     05  FILLER       PICTURE X(8).
```

By naming the SYMBOLIC QUEUE and MESSAGE DATE fields of the CD the programmer can refer to these data areas within the program without further defining them. By redefining the END KEY data area, the programmer can use condition-names to refer to the values contained in that area.

Format 2—Output CD Entry

This format is required if the CD entry is FOR OUTPUT. At least one output CD entry must be specified if operations involving symbolic destinations are to be performed. A number of output CD entries in the same program or in different subprograms in the same run unit may be used to send different portions of the same message, so that parts of one message may be transferred to the MCS using different CD entries.

Until the transfer of a message for a specific destination from the COBOL object program to the MCS is complete, the transfer of a second message for the same destination must not begin; however, the transfer of messages for different destinations may proceed independently.

The specification of an output CD entry always results in a record whose implicit description is shown in Figure 82.

All data-names must be unique within the CD entry.

DESTINATION COUNT Clause: The DESTINATION COUNT clause defines data-name-1 as the name of an unsigned 4-digit integer data item, occupying character positions 1 through 4 of the record.

When a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement is executed, data-name-1 specifies the number of destinations to be used from the Destination Table. If the number is not in the range from 1 through integer-2, an error occurs, and execution of the statement is terminated; it is the user's responsibility to ensure that the value is valid.

```
Equivalent COBOL Record Description         Description of Use

01  data-name-0.
    02  data-name-1         PICTURE 9(4).       Destination Count
    02  data-name-2         PICTURE 9(4).       Text Length
    02  data-name-3         PICTURE XX.         Status Key
    02  data-name OCCURS    integer-2 TIMES.    Destination Table
        03  data-name-4     PICTURE X.          Error Key
        03  data-name-5     PICTURE X(12).      Symbolic
                                                Destination
```

Figure 82. Output CD Entry—Implicit Description

If the contents of data-name-1 are n, the symbolic destinations in ascending order are the first through nth occurrences of data-name-5 (the symbolic destination).

If the DESTINATION TABLE clause is omitted, the contents of data-name-1 must be 1, and one ERROR KEY and one SYMBOLIC DESTINATION are assumed. In this case, when referring to these items, subscripting or indexing must not be used.

TEXT LENGTH Clause: This clause defines data-name-2 as the name of an unsigned 4-digit integer data item, occupying character positions 5 through 8 of the record.

In the execution of a SEND statement, MCS interprets the contents of data-name-2 as the number of leftmost bytes of main storage of the identifier named in the SEND statement to be transferred (see SEND statement).

STATUS KEY Clause: This clause defines data-name-3 as the name of a 2-character elementary alphanumeric data item, occupying character positions 9 and 10 of the record.

The contents of data-name-3 indicate the status condition of the previously executed communication statement. The program should, therefore, check the contents of data-name-3 immediately after each communications operation. The values data-name-3 can contain, and their meanings, are defined in Figure 81.

DESTINATION TABLE Clause: This clause specifies that the items identified by the ERROR KEY and SYMBOLIC DESTINATION clauses are repeated integer-2 times.

The INDEXED BY option provides index-name-1, index-name-2, and so forth, as a means of referring to one specific ERROR KEY and SYMBOLIC DESTINATION. (For a description of the INDEXED BY option, see the chapter on Table Handling.)

When the DESTINATION TABLE clause is specified, character positions 11 through 23 and every following set of 13 characters form table items. When the DESTINATION TABLE clause is omitted, it is assumed that there is one ERROR KEY and one SYMBOLIC DESTINATION for this output CD entry.

ERROR KEY Clause: This clause defines data-name-4 as the name of a 1-character elementary alphanumeric item.

During execution of a SEND, ENABLE OUTPUT, or DISABLE OUTPUT statement, the MCS updates the contents of each occurrence of data-name-4, as follows:

1 If the corresponding symbolic destination in data-name-5 is unknown to the MCS

0 If this occurrence of the symbolic destination is known to the MCS

Note: The programmer should not examine the ERROR KEY unless the STATUS KEY field is set to '20'.

SYMBOLIC DESTINATION Clause: This clause defines data-name-5 as the name of a 12-character elementary alphanumeric data item.

Each occurrence of data-name-5 contains a symbolic destination. Character positions 1 through 8 must have been previously defined to the MCS.

Note: When a message is being sent to a remote station, the MCS adds the proper line control characters.

OUTPUT CD ENTRY EXAMPLE: The following example illustrates an output CD entry, with an optional level-01 data description entry redefining the data areas:

```
CD  OUTPUT-AREA FOR OUTPUT
    TEXT LENGTH IS MSG-LENGTH
    SYMBOLIC DESTINATION IS Q-OUT.
01  OUTAREA-RECORD.
    05  FILLER          PICTURE X(10).
    05  ERRKEY-CODE     PICTURE X.
        88  KNOWN         VALUE "0".
        88  UNKNOWN       VALUE "1".
    05  FILLER          PICTURE X(12).
```

By naming the TEXT LENGTH and SYMBOLIC DESTINATION fields of the CD entry, the programmer can refer to those data areas within the program without further defining them. By redefining the ERROR KEY data area, the programmer can use condition-names to refer to the values contained in that area.

COMMUNICATION—PROCEDURE DIVISION

The Procedure Division for the Communication Feature contains statements that direct data communication functions. These statements are: ENABLE, DISABLE, ACCEPT MESSAGE COUNT, RECEIVE, and SEND.

ENABLE STATEMENT

The ENABLE statement notifies the MCS to allow data transfer between specified sources/destinations and specified input/output queues.

Format

```
          ┌ INPUT [TERMINAL] ┐
ENABLE   <                    >  cd-name
          | OUTPUT           |
          └                  ┘
             ┌ identifier ┐
   WITH KEY <              >
             | literal    |
             └            ┘
```

The ENABLE statement provides a logical connection between the MCS and the specified sources or destinations. If the logical connection already exists, or is handled outside the COBOL program, the ENABLE statement is not required.

The logical path for data transfer between the COBOL program and the MCS is not affected by the ENABLE statement.

When INPUT is specified, the cd-name must specify a CD FOR INPUT. When OUTPUT is specified, the cd-name must specify a CD FOR OUTPUT.

When INPUT TERMINAL is specified, the logical paths between the specified source and all previously-enabled MCS input queues is activated. In the input CD specified by cd-name, only the SYMBOLIC SOURCE field (data-name-7) is meaningful to the MCS; this field must contain the name of the symbolic source before the ENABLE statement is executed.

When INPUT without TERMINAL is specified, the logical paths between the MCS input queues corresponding to the specified queue structure and all previously-enabled sources are activated. Before the ENABLE statement is executed, the COBOL programmer must specify the queue structure (or portion thereof) to be activated. To do this, the programmer places the queue structure name in the SYMBOLIC QUEUE field (data-name-1) of the input CD, and, if necessary, places any needed sub-queue names in the SYMBOLIC SUB-QUEUE fields (data-name-2 through data-name-4). When a sub-queue name is specified, all higher-level subqueue names in that branch of the queue structure must also be specified. Any SYMBOLIC SUB-QUEUE fields for which a sub-queue name has not been specified must be set to spaces.

Note that the logical path between an MCS input queue and a symbolic source is enabled only if both the queue and the source are enabled.

When OUTPUT is specified, the logical path(s) between the destination(s) specified in the SYMBOLIC DESTINATION field(s) for this output CD and the corresponding MCS output queue(s) are activated. The symbolic destination names must be placed in all required occurrences of the destination table (data-name-5) of the output CD before the ENABLE statement is executed.

For the KEY phrase, literal or identifier must be defined as alphanumeric. When the ENABLE statement is executed, the value in literal or identifier is matched with the system password (which is from 1 through 10 characters in length). If the values match, ENABLE statement execution is completed. If the values do not match, the STATUS KEY in this CD entry is updated to "40". (See Figure 81 for status key meanings.)

DISABLE STATEMENT

The DISABLE statement notifies the MCS to prevent data transfer between specified sources/destinations and specified input/output queues.

Format

```
         ┌                    ┐
         | INPUT [TERMINAL]   |
DISABLE  <                    >  cd-name
         | OUTPUT             |
         └                    ┘

             ┌              ┐
             | identifier   |
   WITH KEY  <              >
             | literal      |
             └              ┘
```

The DISABLE statement logically disconnects the MCS and the specified sources or destinations. If the MCS and the source/destination have already been disconnected, or if the action is to be handled outside the COBOL program, the DISABLE statement is not required.

The logical path for data transfer between the COBOL program and the MCS is not affected by the DISABLE statement.

When INPUT is specified, the cd-name must specify a CD FOR INPUT. When OUTPUT is specified, the cd-name must specify a CD FOR OUTPUT.

When INPUT TERMINAL is specified, the logical paths between the specified source and all MCS input queues and sub-queues is deactivated. In the input CD specified by cd-name, only the SYMBOLIC SOURCE field (data-name-7) is meaningful to the MCS; this field must contain the name of the symbolic source before the DISABLE statement is executed.

When INPUT without TERMINAL is specified, the logical paths between the MCS input queues corresponding to the specified structure and all sources are deactivated. Before the DISABLE statement is executed, the COBOL programmer must specify the queue structure (or portion thereof) to be deactivated. To do this, the programmer places the queue structure name in the SYMBOLIC QUEUE field (data-name-1) of the input CD, and, if necessary, places any needed sub-queue names in the SYMBOLIC SUB-QUEUE fields (data-name-2 through data-name-4). When a sub-queue name is specified, all higher-level subqueue names in that branch of the queue structure must also be specified. Any SYMBOLIC SUB-QUEUE fields for which a sub-queue name has not been specified must be set to spaces.

Note that the logical path between an MCS input queue and a symbolic source is disabled if either the queue or the source is disabled.

When OUTPUT is specified, the logical path(s) between the destination(s) specified in the SYMBOLIC DESTINATION field(s) for this output CD and the corresponding MCS output queue(s) are deactivated. The symbolic destination names must be placed in all required occurrences of data-name-5 of this output CD before the DISABLE statement is executed.

For the KEY phrase, literal or identifier must be defined as alphanumeric. When the DISABLE statement is executed, the value in literal or identifier is matched with the system password (which is from 1 through 10 characters in length). If the values match, DISABLE statement execution is completed. If the values do not match, the STATUS KEY in this CD entry is updated to "40". (See Figure 81 for status key meanings.)

ACCEPT MESSAGE COUNT STATEMENT

The ACCEPT MESSAGE COUNT statement returns the number of complete messages in a given input queue structure.

Format

ACCEPT cd-name MESSAGE COUNT

The cd-name must specify an input CD entry.

Before the ACCEPT MESSAGE COUNT statement is executed, the COBOL programmer must specify the queue structure (or portion thereof) for which a message count is required. To do this, the programmer places the queue structure name in the SYMBOLIC QUEUE field (data-name-1) of the input CD, and, if necessary, places any needed sub-queue names in the SYMBOLIC SUB-QUEUE fields (data-name-2 through data-name-4). When a sub-queue name is specified, all higher-level sub-queue names in that branch of the queue structure must also be specified. Any SYMBOLIC SUB-QUEUE fields for which a sub-queue name has not been specified must be set to spaces.

When the statement is executed, the STATUS KEY field (data-name-10) and the MESSAGE COUNT field (data-name-11) of this input CD entry are updated.

Execution of the ACCEPT MESSAGE COUNT statement causes the MESSAGE COUNT field of the named input CD to be updated with the number of complete messages present in the input queue structure. Thus, the COBOL program can check a queue structure for a predetermined message count before executing a specific communication processing function.

When the ACCEPT MESSAGE COUNT statement is executed, the STATUS KEY field of the named input CD is updated as shown in Figure 81. When a STATUS KEY other than "00" is returned, the MESSAGE COUNT field is unchanged.

RECEIVE STATEMENT

The RECEIVE statement makes available to the COBOL program a message, message segment, or a portion of a message or message segment, and pertinent information about that message from an input queue maintained by the MCS.

Format

```
                    ┌ MESSAGE ┐
RECEIVE cd-name    <           >  INTO identifier
                    └ SEGMENT ┘

    [NO DATA imperative-statement]
```

The cd-name must specify an input CD entry.

Before the RECEIVE statement is executed, the COBOL programmer must specify the queue structure (or portion thereof) from which a message segment is required. To do this, the programmer places the queue structure name in the SYMBOLIC QUEUE field (data-name-1) of the input CD, and, if necessary, places any needed sub-queue names in the SYMBOLIC SUB-QUEUE fields (data-name-2 through data-name-4). When a sub-queue name is specified, all higher-level sub-queue names in that branch of the queue structure must also be specified. Any SYMBOLIC SUB-QUEUE fields for which a sub-queue name has not been specified must be set to spaces.

Upon execution of the RECEIVE statement, data is transferred to the receiving character positions of identifier, aligned to the left without any space fill and without any data format conversion. The following data items in the input CD are appropriately updated when the RECEIVE statement is executed: SYMBOLIC SUB-QUEUE-1 through SYMBOLIC SUB-QUEUE-4, MESSAGE DATE, MESSAGE TIME, SYMBOLIC SOURCE, TEXT LENGTH, END KEY, and STATUS KEY (see Figure 81).

A complete message need not be received before another MCS queue is accessed. Because of this, messages from different MCS queues may be processed at the same time by a COBOL program. (Note, however, that a message is not made available to the COBOL program until it is completely received by the MCS and placed in a queue.)

A single execution of a RECEIVE statement never returns more than a single message (when the MESSAGE phrase is used) or a single segment (when the SEGMENT phrase is used), regardless of the size of the receiving area.

When the MESSAGE phrase is used, the end-of-segment condition, if present, is ignored. (This occurs only when the user, through the MCS, segments the message, and the COBOL program uses MESSAGE mode to RECEIVE the message.)

Note: For the treatment of the end-of-segment condition (when the MESSAGE phrase is used) as previously implemented, see Appendix A.

The following rules apply to the data transfer:

- If a message is the same size as identifier, the message is stored in identifier.

- If a message size is smaller than identifier, the message is aligned to the leftmost character position of identifier with no space fill.

- If a message size is larger than identifier, the message fills identifier left to right, starting with the leftmost character of the message. The remainder of the message can be transferred to identifier with subsequent RECEIVE statements referring to the same queue. Either the MESSAGE or the SEGMENT option may be specified for the subsequent RECEIVE statements.

When the SEGMENT phrase is used, the end-of-segment condition, if present (or the end-of-message condition, if present), determines the end of data transfer. The following rules apply to the data transfer:

- If a segment is the same size as identifier, the segment is stored in identifier.

- If a segment size is smaller than identifier, the segment is aligned to the leftmost character position of identifier with no space fill.

- If a segment size is larger than identifier, the segment fills identifier left to right, starting with the leftmost character of the segment. The remainder of the segment can be transferred to identifier with subsequent RECEIVE statements referring to the same queue. Either the MESSAGE or the SEGMENT option may be specified for the subsequent RECEIVE statements.

- If the message text has an end-of-message or end-of-group indicator associated with it, an associated end-of-segment indicator is implied. Thus, the text is treated as a message segment.

After 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 portions of the message to be returned.

After the execution of a program termination statement that ends this run unit, the disposition of the remaining portions of any message only partially obtained is not defined.

During execution of the RECEIVE statement, when the MCS finds that a completed message is not available in the requested queue structure:

- If NO DATA is specified, the RECEIVE statement execution ends with the appropriate input CD fields updated; the NO DATA imperative-statement is then executed.

- If NO DATA is not specified, execution is suspended (that is, the program is placed in wait status) until a completed message becomes available in the requested queue structure.

- If any errors occur during execution of a RECEIVE statement, the STATUS KEY is updated and control passes to the next executable statement, whether or not the NO DATA option is specified.

SEND STATEMENT

The SEND statement causes a message, a message segment, or a portion of a message or message segment to be released to the MCS.

Format 1

SEND cd-name FROM identifier-1

Format 2

```
                                         ┌ identifier-2 ┐
                                         │ ESI          │
   SEND cd-name [FROM identifier-1] WITH <              >
                                         │ EMI          │
                                         │ EGI          │
                                         └              ┘

                          ┌  ┌ ┌ identifier-3 ┐ ┌ LINE  ┐ ┐  ┐
                          │  < <              > │ LINES │ >  │
     ┌ BEFORE ┐           │  │ │ integer      ┘ └       ┘ ┘  │
[    <        > ADVANCING <  └ └                             > ]
     │ AFTER  │           │                                  │
     └        ┘           │   ┌ mnemonic-name ┐              │
                          │   <               >              │
                          │   │ PAGE          ┘              │
                          └   └                              ┘
```

Messages may be transferred to the MCS as segments, as complete messages, or as parts of segments or messages. However, data is never transmitted to the named destination until a complete message has been transferred to the MCS.

The cd-name must specify an output CD entry.

Before a SEND statement is executed, this output CD entry must contain:

- In the DESTINATION COUNT field, the number of destinations to be used from the destination table.

- In the TEXT LENGTH field, the number of leftmost bytes of contiguous data to be transferred to the MCS from identifier-1.

- In the SYMBOLIC DESTINATION field(s), the name(s) of the symbolic destination(s) that are to receive the message or message segment. These names must have been previously defined to the MCS.

After execution of the SEND statement, data is transferred from identifier-1 to the MCS output queue(s) corresponding to the symbolic destination name(s) contained in the SYMBOLIC DESTINATION field(s).

Once a SEND statement has released a portion of a message to the MCS, only subsequent SEND statement executions in the same run unit can release the remaining portion of the message.

When a receiving device (printer, display screen, card punch, and so forth) has a fixed line size:

- Each message or message segment begins at the leftmost character position of the physical line.

- Any message or message segment smaller than the physical line size is transmitted so as to appear space-filled to the right.

- Any message or message segment larger than the physical line size is not truncated. Instead, the entire physical line is filled with characters and transmitted to the device; the excess characters are continued on the next line.

When a receiving device (paper tape punch, another computer, and so forth) can handle variable length messages, each message or message segment begins at the next available character position of the device.

As part of the execution of the SEND statement, the MCS interprets the contents of the TEXT LENGTH field to be the user's indication of the number of leftmost character positions of identifier-1 from which data is to be transferred.

If the contents of the TEXT LENGTH field are 0, no characters of data are transferred from identifier-1. (A 0 TEXT LENGTH field is valid only with the Format 2 SEND statement.)

If the contents of the TEXT LENGTH field are outside the range of 0 through the size of identifier-1 inclusive, an error is indicated in the STATUS KEY field, and no data is transferred. (See Figure 81, STATUS KEY Field—Possible Values.)

If the user causes special control characters to be embedded as data characters within the message, these control characters are enqueued with the message, and it is the user's responsibility to ensure that these characters function as intended.

The disposition of a portion of a message not terminated by an associated EMI or EGI is undefined. (However, such a message portion will not be transmitted to the destination.)

Format 1 Considerations

A single execution of a Format 1 SEND statement releases only a single portion of a message or message segment to the MCS.

Format 2 Considerations

The WITH phrase of this format allows the user to specify whether or not an end indicator is associated with the message or message segment.

A single execution of a Format 2 SEND statement never releases to the MCS more than a single message or message segment, as indicated by identifier-2, ESI, EMI, or EGI.

When the FROM identifier-1 option is omitted, then an end indicator is associated with the data enqueued by a previous SEND statement, if any.

Note that a message or message segment need not contain text data but may be comprised solely of an end indicator.

Identifier-2 must specify a 1-character unsigned integer. The contents of identifier-2 indicate that the contents of identifier-1 have an end indicator associated with them according to the codes shown in Figure 83.

Any character other than 1, 2, or 3 is interpreted as 0.

If the contents of identifer-2 are other than 1, 2, or 3, and identifer-1 is not specified, or the output CD TEXT LENGTH field is 0, then an error is indicated in the STATUS KEY field of the associated CD entry, and no data is transferred.

The hierarchy of end indicators, and their meanings, is as follows:

If identifier-2 contains:	Then identifier-1 has associated with it:	Which means:
0	No indicator	No indicator
1	ESI	End of Segment Indicator
2	EMI	End of Message Indicator
3	EGI	End of Group Indicator

Figure 83. End Indicator Codes

EGI End of Group Indicator—the group of messages to be transmitted is complete. For this implementation, the EGI has no special significance; however, because it does imply the presence of both EMI and ESI, it may be used to indicate completion of a message. If EGI is specified without accompanying message text immediately following the end of the preceding message, it is treated as comments (that is, it is ignored).

EMI End of Message Indicator—the message to be transmitted is complete.

ESI End of Segment Indicator—the segment to be transmitted is complete.

An EGI need not be preceded by an EMI or ESI. An EMI need not be preceded by an ESI.

ADVANCING OPTION: For devices for which such positioning is applicable, this option allows control of the vertical positioning of each message or message segment. If vertical positioning is not applicable for the device, the MCS ignores the ADVANCING option.

If WITH identifier-2 is specified, and identifier-2 does not contain 1, 2, or 3, the ADVANCING option is ignored by the MCS.

If vertical positioning is applicable, the following rules apply:

- When BEFORE ADVANCING is specified, the data is presented on the device before vertical repositioning.

- When AFTER ADVANCING is specified, the data is presented on the device after vertical repositioning.

- If identifier-3 or integer is specified, the device is repositioned downward the number of lines specified in identifier-3 or integer.

- When mnemonic-name is specified, a skip to channels 1 through 9, 10 through 12, or space takes place. Mnemonic-name must be equated with function-name-1 in the SPECIAL-NAMES paragraph (valid function-names are listed in the System Dependencies chapter).

- If PAGE is specified, the device is repositioned to the next page. If PAGE has no meaning for this device, then the device is repositioned 1 line downward.

- If the ADVANCING option is omitted, automatic advancing is provided as if the user had specified AFTER ADVANCING 1 LINE.

DEBUGGING FEATURES

The Debugging features specify the conditions under which data items or procedures are to be monitored during program execution.

COBOL source language debugging statements and background symbolic debugging (for which no source language changes are needed) are provided. In addition, the Federal Information Processing Standard (FIPS) Flagger can be specified; the FIPS Flagger identifies source clauses and statements that do not conform to the Federal standard. Only the COBOL source language debugging statements and the FIPS Flagger are described in this chapter.

The user decides what to monitor and what information to retrieve for debugging purposes. The COBOL debugging features merely provide access to pertinent information.

COBOL SOURCE LANGUAGE DEBUGGING

COBOL language elements that implement the Debugging feature are: a compile-time switch (WITH DEBUGGING MODE), an object-time switch, a USE FOR DEBUGGING Declarative, the special register DEBUG-ITEM, and debugging lines (which can be written in the Environment, Data, and Procedure Divisions).

COMPILE-TIME SWITCH

In the SOURCE-COMPUTER paragraph of the Configuration Section, the WITH DEBUGGING MODE clause acts as a compile-time switch.

Format

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

The WITH DEBUGGING MODE clause serves as a compile-time switch for the debugging statements written in the source program.

When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled as specified in this chapter.

When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as documentation.

OBJECT-TIME SWITCH

The object-time switch dynamically activates the debugging code generated when WITH DEBUGGING MODE is specified.

When debugging mode is specified, through the object-time switch, all the debugging sections and debugging lines compiled into the object program are activated.

When debugging mode is suppressed, through the object-time switch, any USE FOR DEBUGGING declarative procedures are inhibited. However, all debugging lines remain in effect.

Recompilation of the source program is not required to activate or deactivate the object-time switch.

When WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph, the object-time switch has no effect on execution of the object program.

USE FOR DEBUGGING DECLARATIVE

The USE FOR DEBUGGING sentence identifies the items in the source program that are to be monitored by the associated debugging Declarative procedure.

Format

```
section-name SECTION [priority-number].

USE FOR DEBUGGING

            ┌                                    ┐
            │ cd-name-1                          │
            │ [ALL REFERENCES OF] identifier-1   │
        ON  < file-name-1                        >
            │ procedure-name-1                   │
            │ ALL PROCEDURES                     │
            └                                    ┘

            ┌                                    ┐
            │ cd-name-2                          │
            │ [ALL REFERENCES OF] identifier-2   │
            │ file-name-2                        │ ...
            │ procedure-name-2                   │
            │ ALL PROCEDURES                     │
            └                                    ┘
```

When specified, all debugging sections must be written together immediately after the DECLARATIVES header.

Except for the USE FOR DEBUGGING sentence itself, within the debugging procedure there must be no reference to any nondeclarative procedure.

Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

A debugging section is not executed for a specific operand more than once as the result of the execution of a single statement, no matter how many times the operand is explicitly specified. An exception to this rule is that each specification of a subscripted or indexed identifier will cause invocation of the debugging declarative. For a PERFORM statement that causes repeated execution of a procedure, any associated procedure-name debugging declarative section is executed once for each repetition.

For debugging purposes, each separate occurrence of an imperative verb within an imperative statement begins a separate statement.

Statements appearing outside the debugging sections must not refer to procedure-names defined within the debugging sections.

Except for the USE FOR DEBUGGING sentence itself, statements within a debugging declarative section may refer only through the PERFORM statement to procedure-names defined in a different USE procedure.

Procedure-names within debugging declarative sections must not appear in any USE FOR DEBUGGING sentence.

Figure 84 shows, for each valid option, the points during program execution when the USE FOR DEBUGGING procedures are executed.

In Figure 84 on page 338, cd-name-n, identifier-n, file-name-n, and procedure-name-n refer to the first and all subsequent specifications of that type of operand in one USE FOR DEBUGGING sentence.

Any given cd-name, identifier, file-name, or procedure-name may appear in only one USE FOR DEBUGGING sentence, and only once in that sentence.

An identifier in a USE FOR DEBUGGING sentence:

IBM Extension

- Must not refer to any item in the Report Section except sum counters.

End of IBM Extension

- If it contains an OCCURS clause or is subordinate to an entry containing an OCCURS clause, must be specified without the subscripting or indexing normally required. (A SEARCH or SEARCH ALL statement that refers to such an identifier will not invoke the USE FOR DEBUGGING procedures.)

- Must not be a special register.

When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, procedure-name-1, procedure-name-2, and so forth, must not be specified in any USE FOR DEBUGGING sentence. The ALL PROCEDURES option may be specified only once in a program.

When a USE FOR DEBUGGING operand is used as a qualifier, such a reference in the program does not activate the debugging procedures.

References to the DEBUG-ITEM special register may only be made from within a debugging declarative procedure.

USE FOR DEBUGGING operand	**Upon execution of the following, the USE FOR DEBUGGING procedures are executed immediately:**
cd-name-n	after ACCEPT/DISABLE/ENABLE/SEND cd-name-n after RECEIVE cd-name-n and NO DATA option not executed
identifier-n	before REWRITE/WRITE identifier-n and after FROM option move, if applicable before RELEASE identifier-n and after FROM option move (if applicable) after each initialization, modification, or evaluation of identifier-n in PERFORM/VARYING/ AFTER/UNTIL identifier-n after any other COBOL statement explicitly referring to identifier-n that changes its contents (see Note)
ALL REFERENCES OF identifier-n	before GO TO DEPENDING ON identifier-n control is transferred, and before any associated debugging section for procedure-name is executed before REWRITE/WRITE identifier-n and after FROM option move, if applicable before RELEASE identifier-n and after FROM option move (if applicable) after each initialization, modification, or evaluation of identifier-n in PERFORM/VARYING/ AFTER/UNTIL identifier-n after any other COBOL statement explicitly referring to identifier-n (see Note)
file-name-n	after CLOSE/DELETE/OPEN/START file-name-n after READ file-name-n and AT END/INVALID KEY not executed
procedure-name-n	before each execution of the named procedure after execution of an ALTER statement referring to the named procedure
ALL PROCEDURES	before each execution of every nondebugging procedure after execution of every ALTER statement (except ALTER statements in Declarative procedures)

Note: Operands acted upon but not explicitly named in such statements as ADD, MOVE, or SUBTRACT CORRESPONDING never cause activation of a USE FOR DEBUGGING procedure when such statements are executed. If identifier-n is specified in a phrase that is not executed, the associated debugging section is not executed.

Figure 84. Execution of Debugging Declaratives

DEBUG-ITEM Special Register

The DEBUG-ITEM special register provides information for a debugging declarative procedure. DEBUG-ITEM has the following implicit description.

Format

```
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 DEBUG-ITEM special register provides information about the conditions causing debugging section execution.

Before each debugging section is executed, DEBUG-ITEM is filled with spaces. The contents of the DEBUG-ITEM subfields are then updated according to the rules for the MOVE statement, with one exception: DEBUG-CONTENTS is updated as if the move were an alphanumeric to alphanumeric elementary move without conversion of data from one form of internal representation to another. After updating, each field contains:

DEBUG-LINE: The source-statement sequence-number or the compiler-generated card number, depending on the compiler option chosen.

DEBUG-NAME: The first 30 characters of the name causing debugging section execution. Any qualifiers are separated by the word "OF". (Subscripts or indexes are not entered in DEBUG-NAME.)

DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3: If the DEBUG-NAME is subscripted or indexed, the occurrence number of each level is entered in the respective DEBUG-SUB-n. If the item is not subscripted or indexed, these fields remain spaces.

DEBUG-CONTENTS: Data is moved into DEBUG-CONTENTS as shown in Figure 85.

Item causing debug section execution	DEBUG-LINE contains number of COBOL statement referring to	DEBUG-NAME contains	DEBUG-CONTENTS contains
cd-name-n	cd-name-n	cd-name-n	contents of cd-name-n area
identifier-n	identifier-n	identifier-n	contents of identifier-n when control passes to debug section
file-name-n	file-name-n	file-name-n	for READ; contents of record retrieved. Other references; spaces
procedure-name-1 ALTER reference	ALTER statement	procedure-name-1	procedure-name-n in TO PROCEED TO phrase
GO TO procedure-name-n	GO TO statement	procedure-name-n	spaces
procedure-name-n in SORT/MERGE INPUT/OUTPUT PROCEDURE	SORT/MERGE statement	procedure-name-n	"SORT INPUT" "SORT OUTPUT" "MERGE OUTPUT" as applicable
PERFORM statement transfer of control	this PERFORM statement	procedure-name-n	"PERFORM LOOP"
procedure-name-n in a USE procedure	statement causing USE procedure execution	procedure-name-n	"USE PROCEDURE"
implicit transfer from previous sequential procedure	previous statement executed in previous sequential procedure (see note)	procedure-name-n	"FALL THROUGH"
first execution of first nondeclarative procedure	line number of first nondeclarative procedure-name	first nondeclarative	"START PROGRAM"

Figure 85. DEBUG-ITEM Subfield Contents

Note to Figure 85: If this procedure is preceded by a section header and control is passed through the section header, the statement number refers to the section header.

DEBUGGING LINES

A debugging line is any line in a source program with a "D" coded in column 7 (the continuation area). If a debugging line contains nothing but spaces in Area A and Area B, it is considered a blank line.

Each debugging line must be written so that a syntactically correct program results whether the debugging lines are compiled into the program, or treated as documentation.

Successive debugging lines are permitted. Debugging lines may be continued; however, each continuation line must contain a "D" in column 7, and character-strings must not be broken across two lines.

Debugging lines may be specified only after the OBJECT-COMPUTER paragraph.

When the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph, all debugging lines are compiled as part of the object program.

When the WITH DEBUGGING MODE clause is omitted, all debugging lines are treated as documentation.

FIPS FLAGGER

The FIPS Flagger—depending on the compiler option chosen—identifies source statements and clauses that do not conform to either the current 1975 FIPS COBOL or the superseded 1972 FIPS COBOL.

1975 FIPS COBOL—or Federal Information Processing Standard COBOL, December 1975—is a compatible subset of American National Standard COBOL, X3.23-1974. 1975 FIPS COBOL is subdivided into four levels: full, high-intermediate, low-intermediate, and low. Any program written to conform to 1975 FIPS COBOL must conform to one of these levels of 1975 FIPS COBOL processing. (Figure 86 shows the 1974 Standard COBOL processing modules included in each of the levels of 1975 FIPS COBOL.)

1974 Standard Module	Full FIPS Module	High Intermediate FIPS Module	Low Intermediate FIPS Module	Low FIPS Module
2 NUC 1,2 (Nucleus)	2 NUC 1,2	2 NUC 1,2	1 NUC 1,2	1 NUC 1,2
2 TBL 1,2 (Table Handling)	2 TBL 1,2	2 TBL 1,2	1 TBL 1,2	1 TBL 1,2
2 SEQ 1,2 (Sequential I-O)	2 SEQ 1,2	2 SEQ 1,2	1 SEQ 1,2	1 SEQ 1,2
2 REL 0,2 (Relative I-O)	2 REL 0,2	2 REL 0,2	1 REL 0,2	—
2 INX 0,2 (Indexed I-O)	2 INX 0,2	—	—	—
2 SRT 0,2 (Sort-Merge)	2 SRT 0,2	1 SRT 0,2	—	—
1 RPW 0,1 (Report Writer)	—	—	—	—
2 SEG 0,2 (Segmentation)	2 SEG 0,2	1 SEG 0,2	1 SEG 0,2	—
2 LIB 0,2 (Library)	2 LIB 0,2	1 LIB 0,2	1 LIB 0,2	—
2 DEB 0,2 (Debug)	2 DEB 0,2	2 DEB 0,2	1 DEB 0,2	—
2 IPC 0,2 (Inter-program Communication)	2 IPC 0,2	2 IPC 0,2	1 IPC 0,2	—
2 COM 0,2 (Communication)	2 COM 0,2	2 COM 0,2	—	—

Figure 86. The 1974 Standard and 1975 FIPS COBOL

The superseded Federal Information Processing Standard for COBOL was Federal Information Processing Standard COBOL, May 1972—or 1972 FIPS COBOL—a compatible subset of American National Standard COBOL, X3.23-1968. 1972 FIPS COBOL, too, was subdivided into four levels. Any program written to conform to 1972 FIPS COBOL must conform to one of those levels of 1972 FIPS COBOL processing. (Figure 87 shows the 1968 Standard COBOL processing modules included in each of the levels of 1972 FIPS COBOL.)

The following lists identify COBOL source elements flagged for each level of both 1975 FIPS COBOL and 1972 FIPS COBOL.

1968 Standard Module	Full FIPS Module	High Intermediate FIPS Module	Low Intermediate FIPS Module	Low FIPS Module
2 NUC 1,2 (Nucleus)	2 NUC 1,2	2 NUC 1,2	2 NUC 1,2	1 NUC 1,2
3 TBL 1,3 (Table Handling)	3 TBL 1,3	2 TBL 1,3	2 TBL 1,3	1 TBL 1,3
2 SEQ 1,2 (Sequential Access)	2 SEQ 1,2	2 SEQ 1,2	2 SEQ 1,2	1 SEQ 1,2
2 RAC 0,2 (Random Access)	2 RAC 0,2	2 RAC 0,2	2 AC 0,2	—
2 SRT 0,2 (Sort)	2 SRT 0,2	1 SRT 0,2	—	—
2 RPW 0,2 (Report Writer)	—	—	—	—
2 SEG 0,2 (Segmentation)	2 SEG 0,2	1 SEG 0,2	1 SEG 0,2	—
2 LIB 0,2 (Library)	2 LIB 0,2	1 LIB 0,2	1 LIB 0,2	—

Figure 87. The 1968 Standard and 1972 FIPS COBOL

1975 FIPS COBOL FLAGGING

When flagging for elements not included in 1975 FIPS COBOL is specified (through a compiler option), the following elements of the COBOL source, if specified, are identified. Each level of flagging is listed separately.

1975 Full FIPS COBOL Flagging

When flagging for the full FIPS level is specified, the following elements, if specified, are identified.

GLOBAL ITEMS: as follows:

Apostrophe used as quote
Floating-point literals and data items
Suppress option of COPY statement
Special register CURRENT-DATE
Special register LABEL-RETURN
Special register LINE-COUNTER
Special register PAGE-COUNTER
Special register RETURN-CODE
Special register SORT-CORE-SIZE
Special register SORT-FILE-SIZE
Special register SORT-MESSAGE
Special register SORT-MODE-SIZE
Special register SORT-RETURN
Special register TALLY
Special register TIME-OF-DAY
Special register WHEN-COMPILED

IDENTIFICATION DIVISION: as follows:

ID DIVISION abbreviation
Program-name in quotes
Paragraphs not in order in Identification Division
Hyphen in continuation area of Identification Division
REMARKS paragraph

ENVIRONMENT DIVISION: as follows:

Omission of Configuration Section
Clauses not in order in I-O-CONTROL paragraph
C,D,I,R, or W organization in system-name
ASSIGN TO system-name-1 clause OR system-name clause
ASSIGN TO integer system-name clause
Data-name instead of literal in FILE-LIMIT(S) IS (ARE) clause
Multiple extents in FILE-LIMIT(S) IS (ARE) clause
ACTUAL KEY IS clause
Short form of RERUN ON clause
APPLY clause
Mnemonic-name without alphabetic character
NO or RESERVE ALTERNATE AREAS clause
FILE-LIMIT(S) clause
PROCESSING MODE clause
MULTIPLE REEL or UNIT clause
NOMINAL clause in SELECT sentence
TRACK AREA or LIMIT clause in SELECT
PASSWORD clause in SELECT sentence
AS organization in system name
C01 through C12 function-names in SPECIAL-NAMES paragraph
CSP function-name in SPECIAL-NAMES paragraph
S01 to S02 function-names in SPECIAL-NAMES paragraph
UPSI-0 to UPSI-7 function-names in SPECIAL-NAMES paragraph

DATA DIVISION: as follows:

Unequal level numbers at same group level
0 in BLOCK CONTAINS or RECORD CONTAINS clause
RECORDING MODE IS clause
LABEL RECORDS clause in sort-file
Data-name option of LABEL RECORDS clause
TOTALING and TOTALED AREA option of LABEL RECORDS clause
REPORTS clause in file-description
Use of VALUE clause as comment in other than condition-name entries
SYNCHRONIZED clause at 01 level
SYNCHRONIZED clause with USAGE IS INDEX
DISPLAY-ST option of USAGE clause
Omission of integer-1 to option of OCCURS DEPENDING ON clause
Nesting of OCCURS DEPENDING ON clauses
Omission of DEPENDING phrase in format-2 of OCCURS clause
0 as integer-1 in format-2 of OCCURS DEPENDING ON clause
Subsequent entries not subordinate in OCCURS DEPENDING ON clause
Asterisk as zero suppression symbol and BLANK WHEN ZERO clause in same entry
Numeric literal without sign in VALUE clause
Numeric literal in VALUE clause for edited items
REPORT SECTION
COMP-1 option of the USAGE clause
COMP-2 option of the USAGE clause
COMP-3 option of the USAGE clause
COMP-4 option of the USAGE clause
COMPUTATIONAL-1 option of the USAGE clause
COMPUTATIONAL-2 option of the USAGE clause
COMPUTATIONAL-3 option of the USAGE clause
COMPUTATIONAL-4 option of the USAGE clause

PROCEDURE DIVISION: as follows:

SKIP1
SKIP2
SKIP3
Mnemonic-name without alphabetic character
THEN used to separate sentences
FROM SYSIN option of ACCEPT statement
FROM CONSOLE option of ACCEPT statement
WITH POSITIONING option of CLOSE statement

WITH DISP option of CLOSE statement
DEBUG statement
UPON CONSOLE option of DISPLAY statement
UPON SYSPUNCH option of DISPLAY statement
UPON SYSOUT option of DISPLAY statement
ENTRY statement
EXAMINE statement
EXHIBIT statement
GENERATE statement
GO TO MORE-LABELS statement
GOBACK statement
INITIATE statement
NOTE statement
ON statement
LEAVE option of OPEN statement
REREAD option of OPEN statement
DISP option of OPEN statement
OTHERWISE in IF statement
Use of negative indexes in PERFORM statement
READY/RESET TRACE statement
SEEK statement
Semicolon in SORT statement
USING KEY option of START statement
Signed numeric literal in STOP statement
TERMINATE statement
TRANSFORM statement
USE BEFORE REPORTING sentence
GIVING option of USE statement
LABEL PROCESSING option of USE statement
AFTER POSITIONING option of WRITE statement

1975 High-Intermediate FIPS COBOL Flagging

When flagging for the high-intermediate FIPS level is specified, all elements in the preceding list are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS: as follows:

REPLACING option of COPY statement
OF or IN option of COPY statement
Pseudo-text option of COPY statement
REPLACING literal BY option

ENVIRONMENT DIVISION: as follows:

SEGMENT-LIMIT clause
SORT or SORT-MERGE option of SAME clause
RECORD KEY clause
INDEXED ORGANIZATION clause in SELECT sentence
ALTERNATE RECORD KEY clause

PROCEDURE DIVISION: as follows:

All sections with the same priority not together
All sections with priority numbers 00 thru 49 not together
MERGE statement
KEY option of READ statement
Nondeclarative portion of program may contain only SORT statement and STOP RUN
if SORT statement is used
Use of more than one SORT statement
COLLATING SEQUENCE in SORT statement

1975 Low-Intermediate FIPS COBOL Flagging

When flagging for the low-intermediate FIPS level is specified, all elements in the preceding lists are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS: as follows:

Comma or semicolon as punctuation
Continuation of words or numeric literals
Figurative constant ALL literal
Figurative constant HIGH-VALUES
Figurative constant LOW-VALUES
Figurative constant QUOTES
Figurative constant SPACES
Figurative constant ZEROES
Figurative constant ZEROS

IDENTIFICATION DIVISION: as follows:

DATE-COMPILED paragraph

ENVIRONMENT DIVISION: as follows:

RESERVE clause in SELECT sentence
OPTIONAL in SELECT sentence
RECORD option of SAME clause
MULTIPLE FILE TAPE clause
Literal phrase in alphabet-name clause
ACCESS MODE IS DYNAMIC

DATA DIVISION: as follows:

SD level indicator
One digit level number
Level number greater than 10
Data-name beginning with nonalphabetic character
66 or 88 number indicator
Integer-1 TO option of BLOCK CONTAINS clause
Data-name option of VALUE OF clause
ASCENDING or DESCENDING KEY option of OCCURS clause
DEPENDING ON option of OCCURS clause
LINAGE clause
Nesting of REDEFINES clauses
Communication Section

PROCEDURE DIVISION: as follows:

Qualification of data-names and paragraph-names
CORRESPONDING option
Unary plus operator
Use of AND OR and NOT in conditional relation
Use of condition-name
Use of arithmetic and relational symbols (+, -, *, **, /, >, <, =)
Sign condition
ON OVERFLOW statement
FROM in ACCEPT statement
DAY DATE or TIME in ACCEPT statement
CD MESSAGE COUNT in ACCEPT statement
Multiple results in ADD statement
GIVING series in ADD statement
Multiple operands in ALTER statement
CALL identifier statement
CANCEL statement
WITH NO REWIND option of CLOSE statement
CLOSE FOR REMOVAL statement
COMPUTE statement
Series in COMPUTE statement
DISABLE statement
UPON option of DISPLAY statement
REMAINDER in DIVIDE statement
INTO or GIVING series of DIVIDE statement
ENABLE statement

GO TO statement with no object
IF statement nesting
Series in INSPECT statement
BY or GIVING series in MULTIPLY statement
REVERSED option of OPEN statement
WITH NO REWIND option of OPEN statement
Multiple file-names in OPEN statement
EXTEND option of OPEN statement
UNTIL option of PERFORM statement
VARYING option of PERFORM statement
NEXT option of READ statement
RECEIVE statement
RELEASE statement
FROM option of RELEASE statement
RETURN statement
INTO option of RETURN statement
SEARCH statement
SEND statement
SORT statement
START statement
STRING statement
Multiple results in SUBTRACT statement
GIVING series in SUBTRACT statement
UNSTRING statement
EXTEND option of USE statement
ALL IDENTIFIER option of USE FOR DEBUGGING sentence
Non-integer number of lines in ADVANCING option of WRITE statement
EOP or END-OF-PAGE option of WRITE statement
File-name series in USE statement

1975 Low FIPS COBOL Flagging

When flagging for the low FIPS level is specified, all elements in the preceding lists are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS: as follows:

Debug items
COPY statement
D in continuation area

ENVIRONMENT DIVISION: as follows:

RANDOM option of ACCESS MODE IS clause
WITH DEBUGGING MODE clause
RELATIVE ORGANIZATION clause in SELECT sentence
RELATIVE KEY clause

DATA DIVISION: as follows:

Linkage Section

PROCEDURE DIVISION: as follows:

USING clause
Priority number on section header
CALL statement
Multiple file-names in CLOSE statement
WITH LOCK option of CLOSE statement
Segment numbers in Declaratives
DELETE statement
EXIT PROGRAM statement
INVALID KEY option of READ statement
INVALID KEY option of REWRITE statement
USE FOR DEBUGGING statement

1972 FIPS COBOL FLAGGING

When flagging for elements not included in 1972 FIPS COBOL is specified (through a compiler option), the following elements of the COBOL source, if specified, are identified. Each level of flagging is listed separately.

1972 Full FIPS COBOL Flagging

When flagging for the full FIPS level is specified, the following elements, if specified, are identified.

GLOBAL ITEMS: as follows:

* comment line
Apostrophe used as quote
Floating-point literals and data items
Debug items
Slash (/) in continuation area
SUPPRESS option of COPY statement
D in continuation area
OF or IN option of COPY statement
Pseudo option of COPY statement
REPLACING literal by option
Two contiguous quotes
Space preceding right parentheses
Space preceding period comma or semicolon
Space following left parentheses
Omission of space before left parentheses
Special register CURRENT-DATE
Special register LABEL-RETURN
Special register LINE-COUNTER
Special register PAGE-COUNTER
Special register RETURN-CODE
Special register SORT-CORE-SIZE
Special register SORT-FILE-SIZE
Special register SORT-MESSAGE
Special register SORT-MODE-SIZE
Special register SORT-RETURN
Special register TALLY
Special register TIME-OF-DAY
Special register WHEN-COMPILED

IDENTIFICATION DIVISION: as follows:

ID DIVISION abbreviation
Program-name in quotes
Paragraphs not in order in Identification Division

ENVIRONMENT DIVISION: as follows:

Omission of Configuration Section
Clauses not in order in SELECT sentence
Clauses not in order in I-O-CONTROL paragraph
ACTUAL KEY clause with sequential access to direct file
C, D, I, R, or W organization in system-name
Omission of IS in ACCESS MODE IS or ACTUAL KEY IS clauses
FILE STATUS clause in SELECT sentence
Short form of RERUN ON clause
APPLY clause
COPY statement in SELECT sentence
WITH DEBUGGING MODE clause
PROGRAM COLLATING SEQUENCE clause
Alphabet-name clause
Literal phrase in alphabet-name clause
L slash (/) or equals (=) in CURRENCY SIGN clause
ACCESS MODE IS DYNAMIC
RECORD KEY clause
ORGANIZATION clause in SELECT sentence
RELATIVE ORGANIZATION clause in SELECT sentence
INDEXED ORGANIZATION clause in SELECT sentence
ALTERNATE RECORD KEY clause
RELATIVE KEY clause

NOMINAL clause in SELECT sentence
TRACK AREA or LIMIT clause in SELECT sentence
PASSWORD clause in SELECT sentence
AS organization in system-name
C01 through C12 function-names in SPECIAL-NAMES paragraph
CSP function-name in SPECIAL-NAMES paragraph
S01-S02 function-names in SPECIAL-NAMES paragraph
UPSI-0 to UPSI-7 function-names in SPECIAL-NAMES paragraph

DATA DIVISION: as follows:

Unequal level numbers at same group level
0 in BLOCK CONTAINS or RECORD CONTAINS clauses
RECORDING MODE IS clause
LABEL RECORDS clause in sort file
TOTALING and TOTALED area option of LABEL RECORDS clause
REPORTS clause in file-description
Use of VALUE clause as comment in other than
condition-name entries
SYNCHRONIZED clause at 01 level
SYNCHRONIZED clause with USAGE IS INDEX
DISPLAY-ST option of USAGE clause
SIGN clause
Omission of integer-1 TO option of OCCURS
DEPENDING ON clause
Nesting of OCCURS DEPENDING ON clauses
77 items after 01 items
B in alphabetic PICTURE character-string
Slash (/) as editing character
LINAGE clause
Linkage Section
Communication Section
Report Section
CODE-SET IS alphabet-name clause
COPY statement for level 01 or 77
COMP-1 option of the USAGE clause
COMP-2 option of the USAGE clause
COMP-3 option of the USAGE clause
COMP-4 option of the USAGE clause
COMPUTATIONAL-1 option of the USAGE clause
COMPUTATIONAL-2 option of the USAGE clause
COMPUTATIONAL-3 option of the USAGE clause
COMPUTATIONAL-4 option of the USAGE clause

PROCEDURE DIVISION: as follows:

USING option
SKIP1
SKIP2
SKIP3
Omission of section header at beginning of
Procedure Division
Unary plus operator
Omission of blank following unary minus operator
Zero paragraphs in section
Zero sentences in paragraph
Omission of TO in EQUAL TO or relation condition
ON OVERFLOW option
THEN used to separate sentences
FROM SYSIN option of ACCEPT statement
FROM CONSOLE option of ACCEPT statement
DAY, DATE, or TIME in ACCEPT statement
CD MESSAGE COUNT in ACCEPT statement
GIVING series in ADD statement
CALL statement
CALL identifier statement
CANCEL statement
WITH POSITIONING option of CLOSE statement
WITH DISP option of CLOSE statement
CLOSE FOR REMOVAL statement
Series in COMPUTE statement
Use of COPY statement in Procedure Division with other
than section or paragraph name
Segment numbers in Declaratives

DEBUG statement
DELETE statement
DISABLE statement
UPON CONSOLE option of DISPLAY statement
UPON SYSPUNCH option of DISPLAY statement
UPON SYSOUT option of DISPLAY statement
Signed literal operands of DISPLAY statement
INOT or GIVING series of DIVIDE statement
ENABLE statement
ENTRY statement
EXHIBIT statement
EXIT PROGRAM statement
GENERATE statement
GO TO MORE-LABELS statement
Omission of TO in GO TO statement
GOBACK statement
INITIATE statement
INSPECT statement
Series in INSPECT statement
MERGE statement
BY or GIVING series in MULTIPLY statement
ON statement
LEAVE option of OPEN statement
REREAD option of OPEN statement
DISP option of OPEN statement
EXTEND option of OPEN statement
OTHERWISE in IF statement
Use of negative indexes in PERFORM statement
Omission of INVALID KEY or AT END option in READ statement
NEXT option of READ statement
KEY option of READ statement
READY/RESET TRACE statement
RECEIVE statement
REWRITE statement
INVALID KEY option of REWRITE statement
SEND statement
Negative literal in format-2 of SET statement
USING file-name series in SORT statement
COLLATING SEQUENCE in SORT statement
START statement
USING KEY option of START statement
STRING statement
GIVING series in SUBTRACT statement
TERMINATE statement
TRANSFORM statement
UNSTRING statement
USE BEFORE REPORTING sentence
GIVING option of USE sentence
EXTEND option of USE sentence
EXCEPTION option of USE sentence
File-name series in USE sentence
USE FOR DEBUGGING sentence
ALL IDENTIFIERS option of USE FOR DEBUGGING sentence
AFTER POSITIONING option of WRITE statement
EOP or END-OF-PAGE option of WRITE statement
BEFORE or AFTER PAGE phrase of WRITE statement
LINE instead of LINES

1972 High-Intermediate FIPS COBOL Flagging

When flagging for the high-intermediate FIPS level is specified, all elements in the preceding 1972 list are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS: as follows:

REPLACING option of COPY statement

ENVIRONMENT DIVISION: as follows:

SEGMENT-LIMIT clause
SORT option of SAME clause

DATA DIVISION: as follows:

ASCENDING or DESCENDING KEY option of OCCURS clause
DEPENDING ON option of OCCURS clause

PROCEDURE DIVISION: as follows:

All sections with the same priority not together
All sections with priority numbers 00 through 49 not together
FROM option of RELEASE statement
INTO option of RETURN statement
SEARCH statement
Nondeclarative portion of program may contain only SORT statement and STOP RUN IF SORT statement is used
Use of more than one SORT statement

1972 Low-Intermediate FIPS COBOL Flagging

When flagging for the low-intermediate FIPS level is specified, all elements in the preceding 1972 lists are flagged, plus the following additional COBOL source elements:

ENVIRONMENT DIVISION: as follows:

ASSIGN TO system-name-1 clause OR system-name clause

DATA DIVISION: as follows:

SD level indicator

PROCEDURE DIVISION: as follows:

RELEASE statement
RETURN statement
SORT statement

1972 Low FIPS COBOL Flagging

When flagging for the low FIPS level is specified, all elements in the preceding 1972 lists are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS: as follows:

Comma or semicolon as punctuation
Continuation of words or numeric literals
COPY statement
Figurative constant ALL literal
Figurative constant HIGH-VALUES
Figurative constant LOW-VALUES
Figurative constant QUOTES
Figurative constant SPACES
Figurative constant ZEROES
Figurative constant ZEROS

IDENTIFICATION DIVISION: as follows:

DATE-COMPILED paragraph

ENVIRONMENT DIVISION: as follows:

RESERVE clause in SELECT sentence
OPTIONAL in SELECT sentence
Data-name instead of literal in FILE-LIMIT(S) IS (ARE) clause
Multiple extents in FILE-LIMIT(S) IS (ARE) clause
RANDOM option of ACCESS MODE IS clause
ACTUAL KEY IS clause
RECORD option of SAME clause
MULTIPLE FILE TAPE clause

DATA DIVISION: as follows:

One digit level number
Level number greater than 10
Data-name beginning with nonalphabetic character
66 or 88 level number
Integer-1 TO option of BLOCK CONTAINS clause
Data-name option of LABEL RECORDS clause
Data-name option of VALUE OF clause
Nesting of REDEFINES clauses
Multiple index-names in OCCURS clause

PROCEDURE DIVISION: as follows:

Priority number on section header
Qualification of data-names and paragraph-names
Use of multiple subscripts
CORRESPONDING option
Use of AND OR and NOT in conditional relation
Use of condition-name
Use of arithmetic and relational symbols (+, -, *, **, /,], [, =)
FROM in ACCEPT statement
Multiple results in ADD statement
Multiple operands in ALTER statement
Use of multiple file-names in CLOSE statement
WITH LOCK option of CLOSE statement
WITH NO REWIND option of CLOSE statement
COMPUTE statement
DECLARATIVES sentence
END DECLARATIVES sentence
UPON option of DISPLAY statement
REMAINDER in DIVIDE statement
GO TO statement with no object
IF statement nesting
REVERSED option of OPEN statement
WITH NO REWIND option of OPEN statement
Multiple file-names in OPEN statement
UNTIL option of PERFORM statement
VARYING option of PERFORM statement
INTO option of READ statement
INVALID KEY option of READ statement
SEEK statement
UP BY or DOWN BY option of SET statement
Use of multiple index-names or identifiers in SET statement
Multiple results in SUBTRACT statement
USE sentence
FROM option of WRITE statement
Noninteger number of lines in ADVANCING option of WRITE statement
Sign condition

PART 6. SYSTEM DEPENDENCIES

SYSTEM DEPENDENCIES

This chapter documents system-dependent COBOL language elements.

Throughout this manual, IBM COBOL-oriented terms are used to describe elements of the sequential, indexed, and relative I-O modules. These terms, together with their OS/VS equivalents, are listed in Figure 88.

IBM COBOL-oriented Term	OS/VS System Equivalent
physical sequential	QSAM (queued sequential access method)
VSAM sequential	VSAM ESDS (VSAM entry sequenced file)
VSAM indexed	VSAM KSDS (VSAM keyed sequential file)
VSAM relative	VSAM RRDS (VSAM relative record file)

Figure 88. Equivalent I-O Terms—IBM COBOL and OS/VS

In this chapter, language elements are arranged in alphabetic order; if within a language element clauses must be written in a specified order, they are listed in that order; if no specific order is required, the clauses are listed in alphabetic order.

CD ENTRY FOR INPUT

In the SYMBOLIC QUEUE and SUB-QUEUE clauses, the first 8 characters of the data-name must match the DD name of the DD statement for the queue or sub-queue.

FD ENTRY

The following considerations apply.

IBM Extension

BLOCK CONTAINS 0 CHARACTERS CLAUSE

This clause may be specified for physical sequential files; the block size is determined at object time from the DD parameters or the data set label.

If the RECORD CONTAINS 0 CHARACTERS clause is specified, and BLOCK CONTAINS 0 CHARACTERS option is used (or if the BLOCK CONTAINS clause is omitted), then the block size is determined at object time from the DD parameters or the data set label of the file.

RECORD CONTAINS 0 CHARACTERS CLAUSE

This clause may be specified for input physical sequential files; the record size is determined at object time from the DD parameters or the data set label. If at object time the actual record is larger than the 01 record description, only the 01 record length is available. If the actual record is shorter, only the actual record length may be referred to or results will be unpredictable.

Note: If the BLOCK CONTAINS 0 or the RECORD CONTAINS 0 clause is specified, then the SAME AREA or SAME RECORD AREA clause may not be specified.

End of IBM Extension

Recording Mode

For physical sequential files, the COBOL compiler scans each record description entry to determine the recording mode. The recording mode may be fixed (F), variable (V), or spanned (S).

RECORDING MODE F: All the records in a file are the same length and each is wholly contained within one block. Blocks may contain more than one record, and there is usually a fixed number of records per block. In this mode, there are no record-length or block-descriptor fields.

RECORDING MODE V: The records may be either fixed or variable in length, and each must be wholly contained in one block. Blocks may contain more than one record. Each data record includes a record-length field, and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

RECORDING MODE S: The records may be either fixed or variable in length, and may be larger than a block. If a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block (or blocks, if required). Only complete records are made available to the user. Each segment of a record in a block, even if it is the entire record, includes a segment-descriptor field, and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

For a given physical sequential file, the compiler determines the recording mode as follows:

F

if all the records are defined as being the same size, and the size is smaller than or equal to the block size.

For blocked F-mode records, the BLOCK CONTAINS clause is required.

V

if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records, and the longest record is smaller than or equal to the block size.

For V-mode records, the BLOCK CONTAINS clause is required.

S

if the maximum block size is smaller than the largest record size.

For S-mode records, the BLOCK CONTAINS CHARACTERS clause is required.

FILE-CONTROL ENTRY

The following considerations apply.

ASSIGN Clause

The assignment-name has the following format:

[comments-][S-]name	physical sequential files
[comments-]AS-name	VSAM sequential files
[comments-]name	VSAM indexed or relative files

The comments field is treated as documentation. If specified, it must end with a hyphen. The comments field is useful for documenting the device and device class to which a file is assigned.

For physical sequential files, the S- (organization) field may be omitted.

For VSAM sequential files the AS- (organization) field must be specified.

For VSAM indexed and relative files the organization field must be omitted.

The name field is required. It is a 1- to 8-character field that specifies the external name for this file. It must be the name specified in the DD statement for this file. It must conform to the rules for formation of a program name; it must not be a COBOL reserved word.

PASSWORD Clause

IBM Extension

For indexed files, if the file has been completely predefined to VSAM, then, at file creation time only the PASSWORD data item for the RECORD KEY must contain the valid password before the file can be successfully opened. For any other type of file processing (including dynamic invocation at file creation time through a COBOL object-time subroutine), every PASSWORD data item for this file must contain a valid password before the file can be successfully opened, whether or not all paths to the data are used in this object program.

End of IBM Extension

RESERVE Clause

The integer must not exceed 255. If the RESERVE clause is omitted, two buffers are reserved; if both RESERVE and SAME AREA are omitted, the number of buffers at execution time is taken from the DD statement for the file; if none is specified, two buffers are reserved.

I-O-CONTROL ENTRY

SAME Clause

Restrictions on specifying the SAME clause are:

IBM Extension

The SAME AREA clause must not be specified when the BLOCK CONTAINS 0 CHARACTERS or RECORD CONTAINS 0 CHARACTERS is specified.

The SAME RECORD AREA clause must not be specified when the RECORD CONTAINS 0 CHARACTERS clause is specified.

End of IBM Extension

MULTIPLE FILE TAPE CLAUSE

This clause is treated as documentation; the function is performed by the system through the LABEL parameter of the DD statement.

INPUT/OUTPUT STATEMENTS

The following considerations apply for the OPEN, WRITE, and CLOSE statements.

OPEN Statement

When OPEN REVERSED is specified, the recording mode may be fixed (F).

The OPEN INPUT and OPEN I-O options are valid for files which have not yet been loaded; note, however, that in this case any input/output request except a WRITE statement for an I-O file results in an error condition.

WRITE ADVANCING Statement

A compile-time option determines whether or not the first character in the record must be reserved for the ADVANCING control character.

IBM Extension

CLOSE Statement

If a CLOSE statement for an OPEN file is not executed before a GOBACK statement in a main program, results are unpredictable. (See Program Termination Considerations in the following Subprogram Linkage description.)

End of IBM Extension

REPORT WRITER—RECORD CONTAINS CLAUSE

IBM Extension

No matter which compile-time control option is specified, the RECORD CONTAINS clause must always include the WRITE ADVANCING control character as part of the record.

End of IBM Extension

SD ENTRY FOR SORT/MERGE

In the SD Entry, the LABEL RECORDS clause is accepted and treated as documentation.

The minimum acceptable record length is 18 bytes; the maximum acceptable record length is 32752 bytes.

SOURCE PROGRAM LIBRARY

For the COPY statement, the OF/IN option is treated as documentation.

SPECIAL-NAMES PARAGRAPH

function-name-1: can be chosen from the list shown in Figure 89.

Function-Name	Meaning	Used In
SYSIN	system logical input unit	ACCEPT statement
SYSOUT	system logical output unit	DISPLAY statement
CONSOLE	console typewriter	ACCEPT and DISPLAY statements
C01 through C12	skip to channel 1 through 12, respectively	WRITE ADVANCING and SEND ADVANCING statement
CSP	suppress spacing	WRITE ADVANCING and SEND ADVANCING statement
S01,S02	pocket select 1 or 2 on punch devices	WRITE ADVANCING statement
1-character nonnumeric literal*	**report writer report code**	**CODE clause**

Figure 89. Valid Names of Function-Name-1

*This is also an IBM Extension.

function-name-2: UPSI-0 through UPSI-7 are COBOL names that identify program switches defined outside the COBOL program, in the communications region. Their contents are considered to be alphanumeric.

IBM Extension

SPECIAL REGISTERS

The following special registers are available to the COBOL user.

RETURN-CODE

RETURN-CODE has the implicit description PICTURE S9999 COMP, and can be set by the user to pass a return code to the invoking program or the system before executing a STOP RUN, EXIT PROGRAM, or GOBACK statement. (See the following paragraphs on Subprogram Linkage.) When control is returned to the invoking program, the code passed by the called program is stored in the calling program's RETURN-CODE special register. The compiler initializes the field to zero (0), which is the normal return code for successful completion; other values returned are conventionally in multiples of 4. However, the maximum value the field can contain is 4095.

The return code can be used to determine subsequent job or job step execution flow.

SORT-CORE-SIZE

Placing the following value in SORT-CORE-SIZE takes advantage of the "maximum" storage parameter of the Sort/Merge program:

> +999999 specifies that Sort/Merge should use all main storage available to it.

Negative integer specification tells COBOL to use the absolute value as the number of bytes to reserve for data management routines. (In this case, Sort/Merge uses the CORE parameter it was installed with, including the maximum main storage parameter.)

SORT-MESSAGE

SORT-MESSAGE has the implicit description PICTURE X(8) DISPLAY. If Sort/Merge is installed to route messages to the printer, then the user can specify the ddname in SORT-MESSAGE to which the Sort/Merge program is to route messages in place of SYSOUT. If SORT-MESSAGE is not modified during the program, SYSOUT is the default value.

For example: If MOVE "SORTDDNM" TO SORT-MESSAGE is executed before the Sort/Merge is begun, messages will be routed to SORTDDNM instead of to SYSOUT.

Note: Use of the SORT-MESSAGE special register is strongly recommended. If SORT-MESSAGE is not used in the COBOL program and if the system opens SYSOUT and the COBOL program then attempts to use it, unpredictable results can occur; in addition, Sort/Merge messages are then interspersed with other unrelated messages from this and other programs. When SORT/MESSAGE is used, all Sort/Merge messages can be sent to an alternative destination and can then be printed in one separate easily-identified group.

SORT-RETURN

If there is no reference to SORT-RETURN in the program, and the Sort/Merge terminates abnormally, a message is displayed on the console. The operator can continue or cancel the job.

The SORT-RETURN special register can also be used to terminate the OS Sort/Merge Program Product operation. The programmer can place the value 16 in this special register at any point during an Input or Output Procedure to terminate the Sort or Merge immediately after execution of the next RELEASE or RETURN statement.

WHEN-COMPILED

WHEN-COMPILED is a 20-byte alphanumeric field with the format:

hour.minute.secondMONTH DAY, YEAR
(hh.mm.ssMMM DD, YYYY)

WHEN-COMPILED makes available to the object program the date-and-time-compiled constant carried in the object module; it is valid only as the sending field in a MOVE statement.

If compilation began at 4:31 p.m. on June 10, 1981, WHEN-COMPILED would contain the value 16.31.00JUN 10, 1981

Note: WHEN-COMPILED may not be moved directly to a user-defined edited field for further editing. So inserting a blank between the 'second' and 'MONTH' fields, for example, is accomplished by setting up two fields:

```
FIELDA       PIC X(20)
FIELDB       PIC X(8)BX(12)
```

WHEN-COMPILED is first moved into FIELDA, which is then moved into FIELDB for the editing.

End of IBM Extension

STATUS KEY VALUE 96

A value of 96 means that no DD statement has been specified for this VSAM file.

SUBPROGRAM LINKAGE

The following considerations for Subprogram Linkage apply.

Linkage Section

The total number of entries in the Linkage Section (01-level and 77-level combined) must not exceed 255.

CALL Statement

The mode of the CALL statement (static or dynamic) is specified through the EXEC job control statement; the default option is static mode.

At object time, when the dynamic CALL statement is used, the COBOL Library Management Facility must be used by the main program and all subprograms in one region/partition. Otherwise, multiple copies of library subroutines may be resident at one time and cause unpredictable results.

Called subprograms that are invoked at object time by the dynamic CALL statement must be members of the system link library or of a user-supplied private library.

The static call statement results in the subprogram so invoked being link-edited with the main program into one load module. Thus, the program-name and the alternate entry point can be specified in any order in the main program's CALL statements.

The dynamic CALL statement results in the dynamic invocation of a separate load module at execution time. In this case, for the CANCEL statement to work properly, alternative entry points for one subprogram should not be specified unless an intervening CANCEL statement has been executed.

Static and dynamic CALL statements may both be specified in the same program. The CALL literal-2 statement results, in this case, in the subprogram so invoked being link-edited with the main program into one load module. The CALL identifier-3 statement results in the dynamic invocation of a separate load module. When a dynamic CALL statement and a static CALL statement to the same subprogram are issued within one program, a second copy of the subprogram is loaded. Therefore, care must be used to avoid duplicate load modules.

Note: Linking two load modules together results logically in a single program with a primary entry point and an alternate entry point, each with its own name. (Each name by which a subprogram is to be dynamically invoked must be known to the system; each such name must be specified in linkage editor control statements as either a NAME or an ALIAS of the load module containing the subprogram.) Only if user modules are link-edited with the attribute of nonreentrant and nonserially-reusable will a CANCEL statement guarantee a fresh copy of the subprogram upon a subsequent CALL.

If a called subprogram has more than one entry point, differing entry points can be called without an intervening CANCEL only if the module has been link-edited with an attribute of either reentrant or reusable. In those cases, the module being called will be in its last-used state.

USING Option

If the called subprogram is written in a language other than COBOL, a CALL statement USING identifier may be a file-name for a physical sequential file, or a data-item defined in the file, Working-Storage, or Linkage Section of the calling program. When a file-name is specified, the file it identifies must be opened in the calling program.

At execution time, the USING option may be used to pass parameters from the EXEC job control statement to a main COBOL program. In this case, a USING option on the Procedure Division header of a main program may contain identifier-1 as its only operand. Information from the PARM field of the EXEC statement is then available in the Linkage Section at the location specified as identifier-1. The first two bytes of identifier-1 contain a count of the number of bytes of information in the PARM field; the two bytes are set to 0 if the PARM field was omitted. This 2-byte field is binary and should be defined with PIC S9(4) COMP. Immediately following these two bytes is the information in the PARM field. The maximum length of the field to be passed is 100 bytes.

When the first two bytes of identifier-1 contain a count of 0, no storage is allocated for the field following the count bytes and references to it may cause unpredictable results.

Note: When a valid COBOL execution-time option (for example, /FLOW=20) is specified at the end of the PARM field, the count of the number of bytes of information in the PARM field is decreased to exclude the slash and all characters to its right before the PARM field is passed to a main COBOL program. If the PARM field contains any slashes, an additional slash should be added to the end, and the PARM field will be decreased from this last slash. If the data after the last slash is not a valid execution-time option, it is removed from the PARM and an error message is issued.

Program Termination Statements

A main program is the highest level COBOL program invoked in a step. A subprogram is a COBOL program invoked by another COBOL program. (Other language programs that follow COBOL linkage conventions are considered COBOL programs in this sense.)

Figure 90 shows the action taken for each program termination statement in both a main program and a subprogram.

Termination Statement	Main Program	Subprogram
EXIT PROGRAM	Nonoperational	Return to invoking program
STOP RUN	Return to invoker* (may be system and cause end of job step)	Return directly to invoker of main program* (may be system and end job step)
GOBACK	Return to invoker* (may be system and cause end of job step)	Return to invoking program

* If a main program is called by a program written in another language that does not follow COBOL linkage conventions, return will be to this calling program.

Figure 90. Program Termination Statements—Actions Taken

PART 7. SUPPLEMENTARY MATERIAL

- APPENDIX A. IBM OS Full ANS COBOL Items (IBM Extension)
- APPENDIX B. ASCII Considerations
- APPENDIX C. System/370 Unit Record Processing
- APPENDIX D. Intermediate Results
- APPENDIX E. Sample File-Processing Programs
- APPENDIX F. OS/VS COBOL Reserved Word List
- APPENDIX G. EBCDIC and ASCII Collating Sequences
- APPENDIX H. COBOL Statements Flagged by Specifying MIGR
- COBOL Glossary

APPENDIX A. IBM OS FULL ANS COBOL ITEMS—(IBM EXTENSION)

IBM Extension

This appendix contains language elements of IBM OS Full American National Standard COBOL supported by OS/VS COBOL—language that is, in effect an extension to American National Standard COBOL, X3.23-1974. These language extensions are supported for compatibility purposes only.

The IBM OS Full American National Standard COBOL compilers are designed according to the specifications for the highest level of all eight modules of American National Standard COBOL, X3.23-1968, as well as IBM extensions to that standard.

The IBM OS/VS COBOL compiler incorporates all language elements from IBM OS Full American National Standard COBOL, with two exceptions: In OS/VS COBOL the MESSAGE COUNT clause replaces the QUEUE DEPTH clause, and the ACCEPT MESSAGE COUNT statement replaces the IF MESSAGE statement.

OS/VS COBOL allows the user at compile time to request an "old" or "new" object module through a compiler option:

- The "old" compiler option tells the compiler to interpret certain COBOL language elements according to the 1968 Standard.

- The "new" compiler option tells the compiler to interpret the same COBOL elements according to the 1974 Standard.

The three sections of this appendix document three types of extensions:

- Section I—input/output language elements that either support the 1968 standard or are extensions to both the 1968 and 1974 standards and for which either the "old" or the "new" compiler option may be specified

- Section II—other language elements that either support the 1968 standard or are extensions to both the 1968 and 1974 standards and for which either the "old" or the "new" compiler option may be specified

- Section III—language elements that support the 1968 standard for which the "old" compiler option must be specified

This appendix describes language elements within each category. Sections II and III document any differences between the old compiler and the new compiler.

SECTION I—I/O LANGUAGE EXTENSIONS

The input/output language extensions listed in this section can be used with either the "new" or the "old" compiler options.

FILE PROCESSING CAPABILITIES

Within a given access method, old and new language elements may be used interchangeably, if such elements are available. That is, for a physical sequential file, both the FILE STATUS clause and the RECORDING MODE clause can be specified for one file. (Note that old indexed, direct and relative file processing language is the only language valid for ISAM, BSAM, and BDAM files. Conversely, only new file processing language is valid for VSAM indexed files.) OS Full American National Standard COBOL file processing capabilities are shown in Figure 91.

Data Management Technique	Device Type	Access	Organization
QSAM	Reader	[SEQUENTIAL]	Standard sequential
QSAM	Punch	[SEQUENTIAL]	Standard sequential
QSAM	Printer	[SEQUENTIAL]	Standard sequential
QSAM	Tape	[SEQUENTIAL]	Standard sequential
QSAM	Direct access storage	[SEQUENTIAL]	Standard sequential
BSAM	Direct access storage	[SEQUENTIAL]	Direct
BDAM	Direct access storage	RANDOM	Direct
QISAM	Direct access storage	[SEQUENTIAL]	Indexed
BISAM	Direct access storage	RANDOM	Indexed
BSAM	Direct access storage	[SEQUENTIAL]	Relative
BDAM	Direct access storage	RANDOM	Relative

Figure 91. OS ANS COBOL File Processing Capabilities

FILE CONTROL PARAGRAPH

ACCESS MODE SEQUENTIAL or ACCESS MODE RANDOM may be specified with the clauses in this section. The FILE-LIMITS clauses and PROCESSING MODE clauses are accepted and treated as documentation.

ASSIGN Clause

The ASSIGN clause is used to assign a file to an external medium. The FOR MULTIPLE REEL/UNIT option is accepted and treated as documentation.

Assignment-name has the following format and capabilities:

Format

SYSnnn-class-device-S [-name]

The class and device fields are treated as documentation.

Organization is a 1-character field that indicates the file organization. The following characters must be used:

S for files with standard sequential organization.

D for files with direct organization.

W for files with direct organization when REWRITE is used. When the file is opened as INPUT or OUTPUT, however, W is the equivalent of D.

R for files with relative organization.

I for files with indexed organization.

Name is a 1- to 8-character field specifying the external-name by which the file is known to the system. It is the name that appears in the name field of the DD card for the file.

ACTUAL KEY Clause

When creating or retrieving records from a randomly accessed file, the programmer is responsible for providing the ACTUAL KEY for each record to be processed.

Format

ACTUAL KEY IS data-name

Data-name may be any fixed item from 5 through 259 bytes in length. It must be defined in the File, Working-Storage, or Linkage Section. However, if data-name is specified in the File Section, it may not be contained in the file for which it is the key. The following considerations apply:

- The first four bytes of data-name are the track identifier and must be defined as a 5-integer binary data item whose maximum value does not exceed 65535.
- The remainder of data-name—1 through 255 bytes in length—represents the record identifier. It is the user's responsibility to select from 1 through 255 bytes for the symbolic portion of the ACTUAL KEY field. Data within these bytes will be treated exactly as specified.

NOMINAL KEY Clause

The NOMINAL KEY clause specifies a search argument for indexed or relative files.

Format

NOMINAL KEY IS data-name

INDEXED FILES: Data-name may specify any fixed-length Working-Storage item from 1 through 255 bytes in length.

Data-name must be at a fixed displacement from the beginning of the record description in which it appears; that is, it may not appear in the entry subsequent to an OCCURS DEPENDING ON clause.

In random processing, the symbolic identity of the record must be placed in data-name before execution of the READ, WRITE, or REWRITE statement.

The symbolic identity is used when retrieving or updating a record to locate the logical record with a matching RECORD KEY or, when adding a record, to create the key that will be associated with the record.

In sequential processing, a NOMINAL KEY must be specified if a Format-1 START statement is used. When the START statement is executed, the contents of data-name are used to locate the record at which processing is to begin. The next READ statement accesses this record.

RELATIVE FILES: Data-name may specify any 8-integer binary item in Working-Storage whose maximum value does not exceed 15728640.

Data-name must be at a fixed displacement from the beginning of the record description in which it appears; that is, it may not appear in the entry subsequent to an OCCURS DEPENDING ON clause.

The relative record number must be placed in data-name before the execution of the READ, WRITE, or REWRITE statement.

RECORD KEY Clause

A RECORD KEY is used to access an indexed file. It specifies the item within the data record that contains the key for the record.

Format

```
RECORD KEY IS data-name
```

The RECORD KEY clause must be specified for an indexed file.

Data-name may be any fixed-length item within the record. It must be less than 256 bytes in length.

When two or more record descriptions are associated with a file, a similar field must appear in each description, and must be in the same relative position from the beginning of the record, although the same data-name need not be used for both fields.

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

- Files with unblocked records
- Files from which records are to be deleted
- Files whose keys might start with a delete-code character (HIGH-VALUE)

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

TRACK-AREA Clause

This clause may optionally be used when records are to be added to an indexed file in the random access mode. Efficiency in adding a record is improved when the TRACK-AREA clause is specified.

Format

```
                 ┌ data-name ┐
TRACK-AREA IS   <             >  CHARACTERS
                 └ integer   ┘
```

When records are to be added to random access files with indexed organization, this clause specifies either an area (data-name) or the size of an area (integer). The area is used to hold all the blocks on a track, including their count and key fields, plus one logical record.

The area defined by the TRACK-AREA clause must be a multiple of 8 and must not exceed 65272 bytes.

When the integer option is specified, an area of integer bytes is obtained from the system when the file is opened. It is released to the system when the file is closed.

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

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

TRACK-LIMIT Clause

Format

```
                         ┌ TRACK  ┐
TRACK-LIMIT IS integer   │        │
                         │ TRACKS │
                         └        ┘
```

This clause does not cause track allocation, which is the function of a DD card parameter.

SEQUENTIAL ACCESS: When used in conjunction with ACCESS IS SEQUENTIAL and a file is opened as OUTPUT, if the last relative track number used by the file when it is closed is less than that specified in the TRACK-LIMIT clause, the unused portion(s) of the track(s) is filled either with capacity records (mode U, V, or S) or with dummy records (mode F). If the last relative track number used by the file is equal to or greater than that specified in the TRACK-LIMIT clause, or if the clause is omitted, a capacity record or dummy records are written for the current track of the file, and the remaining allocated tracks are not initialized. Note that, because the first relative track is track 0, at least integer plus one track will be initialized.

RANDOM ACCESS: When used in conjunction with ACCESS IS RANDOM, the TRACK-LIMIT clause specifies the last relative track number to be initialized at open time; the tracks are initialized with dummy (mode F) or capacity (modes U, V, or S) records. This defines the total size of the file; that is, no additional tracks may be used by the file, and any references to tracks outside this area will result in an INVALID KEY condition. If this clause is omitted, the number of tracks initialized is determined from the SPACE and VOLUME count parameters of the DD card. The first volume will be initialized according to the primary allocation quantity, and succeeding volumes (if any) will be initialized from the secondary quantity (one quantity per volume).

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph defines some of the special techniques to be used in the program. For example, it specifies when checkpoints are to be taken, the storage areas to be shared by different files, and the location of files on a multiple file reel. The I-O-CONTROL paragraph and its associated clauses are optional.

RERUN Clause

Current formats for the RERUN clause are valid for "old" file organizations. The following notes apply.

END-OF-REEL/UNIT OPTION: This option is valid for sequentially accessed files with any organization.

RECORDS OPTION: This option is valid for sequentially accessed or randomly accessed files with any organization. The value of the integer must not exceed 16777215.

MULTIPLE FILE TAPE Clause

"New" implementation of the MULTIPLE FILE TAPE clause is valid for tape files with "old" sequential organization.

APPLY Clause

There are several options of the APPLY clause. More than one of each option may appear.

Format for Option 1

APPLY WRITE-ONLY ON file-name-1 [file-name-2] ...

This option is used to make optimum use of buffer and device space allocated when creating a file whose recording mode is V. Usually, a buffer is truncated when there is not enough space remaining in it to accommodate the maximum size record. Use of this option will cause a buffer to be truncated only when the next record does not fit in the unused remainder of the buffer. This option has meaning only when the file is opened as OUTPUT.

The files named in this option must have standard sequential organization.

Every WRITE statement associated with the file must use the WRITE record-name FROM identifier option. None of the subfields of record-name may be referred to in procedural statements, nor may any of the subfields be the object of an OCCURS DEPENDING ON clause.

However, if the same file is opened for INPUT or I-O, the subfields of the record-name may be referred to. When the same file is opened for I-O, the WRITE statement must not be used; the REWRITE statement must be used in its place.

Format for Option 2

APPLY CORE-INDEX ON file-name-1 [file-name-2] ...

This option may be specified only for an indexed file whose access mode is random. It is used to specify that the highest level index is to be processed in main storage. The area will be obtained at open time and released at close time.

Format for Option 3

APPLY RECORD-OVERFLOW ON file-name-1 [file-name-2] ...

If the record overflow feature is available for the direct access storage device being used, the amount of unused space on a volume may be reduced by specifying this option for files on that volume. If the option is used, a block that does not fit on the track is partially written on that track and continued on the next available track.

This option may be specified only for a standard sequential file (with F, U, or V mode records) assigned to a direct access storage device, or a direct file with fixed-length records.

Format for Option 4

APPLY REORG-CRITERIA TO data-name ON file-name

If the "reorganization criteria" feature was specified on the DD card for an indexed file when it was created, this option may be specified for the file when ACCESS IS RANDOM is specified.

The reorganization statistics maintained by the system will be placed in data-name when a CLOSE statement is executed for the file. Data-name must be composed of three COMPUTATIONAL items of 2, 2, and 4 bytes in length, respectively.

The first 2 bytes will contain the number of cylinder overflow areas that are full. The second 2 bytes will contain the number of tracks (partial or whole) remaining in the independent overflow area. The last 4 bytes will contain the number of READ or WRITE statements that accessed overflow records that are not the first in the chain of such records.

DATA DIVISION

FD ENTRY

Most clauses in the FD Entry have the same effect in both the old and the new standards. The following clauses, from the IBM American National Standard COBOL Compilers, are accepted by the OS/VS and DOS/VS Compilers.

LABEL RECORDS Clause

The LABEL RECORDS clause with the data-name option, indicates the presence of either user standard labels or nonstandard labels.

Format

```
        ┌             ┐ ┌                                             ┐
        | RECORD IS   | | OMITTED                                     |
LABEL  <               >| STANDARD                                    |
        | RECORDS ARE |< data-name-1 [data-name-2] ... [TOTALING AREA >
        └             ┘ |    IS data-name-3 TOTALED AREA IS           |
                        |    data-name-4]                             |
                        └                                             ┘
```

The STANDARD option must be specified for files with indexed organization.

In the discussion that follows, all references to data-name-1 apply equally to data-name-2.

The data-name-1 option indicates either the presence of user labels in addition to standard labels, or the presence of nonstandard labels. Data-name-1 specifies the name of a user label record. Data-name-1 must appear as the subject of a record description entry associated with the file, and must not appear as an operand of the DATA RECORDS clause for the file.

If user labels are to be processed, data-name-1 may be specified only for direct files, relative files, or for standard sequential files with the exception of files assigned to unit-record devices.

If nonstandard labels are to be processed, data-name-1 may be specified only for standard sequential files, with the exception of files assigned to unit-record devices. The length of a nonstandard label may not exceed 4095 character positions.

All Procedure Division references to data-name-1, or to any item subordinate to data-name-1, must appear within label processing declaratives.

Note: In the discussion that follows, the term volume applies to all input/output devices. Treatment of a direct access storage device in the sequential access mode is logically equivalent to the treatment of a tape file.

The TOTALING and TOTALED AREA option may be specified when the programmer wants to create a sequential file with user labels. With this option, the programmer is able to obtain exact information about each volume of a multivolume file, so that the information can be recorded in the user trailer label each time a volume switch occurs.

Data-name-3, the TOTALING AREA, is defined in the Working-Storage Section. Data-name-3 is used by the programmer to store information to be used in constructing the user labels—information such as accumulated totals for records, identification fields within the current record, and so forth. Before each WRITE statement is issued, the programmer must store information associated with the current record in data-name-3. There are two exceptions: If the SAME RECORD AREA and/or the APPLY WRITE-ONLY clause is specified, then the programmer must store the current record information in data-name-3 after issuing the WRITE statement. The information in data-name-3 is always associated (by the system) with the current WRITE statement.

Data-name-4, the TOTALED AREA, must be defined at the 01 level in the Linkage Section, and must contain fields described as identical with those within data-name-3. The system allocates the space for data-name-4 and uses it to save user label information (obtained from data-name-3) associated with the most recent record actually written on the current volume. Thus, when a volume switch occurs, data-name-4 contains the user label information for the last record actually written on the current volume, and the programmer can use data-name-4 to construct an accurate trailer label for the current volume, and an accurate header label for the next.

For both data-name-3 and data-name-4, the user must define the first two bytes of each record for use by the system.

The TOTALING and TOTALED AREA option may not be specified for S mode records.

RECORDING MODE Clause

The RECORDING MODE clause specifies the format of the logical records in the file.

Format

<u>RECORDING</u> MODE IS mode

Mode is specified as: F, V, U, or S.

F mode (fixed-length format) may be specified when all the logical records in a file are the same length and each is wholly contained within one physical block. This implies that no OCCURS DEPENDING ON clause is associated with an entry in any record description for the file. If more than one record description entry is given following the FD entry, all record lengths calculated from the record descriptions must be equal.

V mode (variable-length format) may be specified for any combination of record descriptions if each record is wholly contained in one physical block. A mode V logical record is preceded by a control field containing the length of the logical record. Blocks of variable-length records include a block-descriptor control field. V mode may not be specified for files with indexed or relative organization.

U mode (unspecified format) may be specified for any combination of record descriptions, if each record is wholly contained in one physical block, and the block contains only one physical record. It is comparable to V mode except that U mode records are not blocked and have no preceding control field. U mode may not be specified for files with indexed or relative organization.

S mode (spanned format) may be specified for any combination of record descriptions. If a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block (or blocks, if required). Only complete records are made available to the user. Each segment of a record in a block, even if it is the entire record, includes a segment-descriptor field, and each block includes a block-descriptor field. These fields are not desribed in the Data Division; provision is automatically made for them. These fields are not available to the user. S mode may be specified for standard sequential or direct files.

When the RECORDING MODE clause is not used to specify the recording mode of the records in the file, the COBOL compiler scans each record description entry to determine it. The recording mode may be F (fixed), U (unspecified), V (variable), or S (spanned).

RECORDING MODE F: All the records in a file are the same length and each is wholly contained within one block. Blocks may contain more than one record, and there is usually a fixed number of records per block. In this mode, there are no record-length or block-descriptor fields.

RECORDING MODE U: The records may be either fixed or variable in length. However, there is only one record per block. There are no record-length or block-descriptor fields.

RECORDING MODE V: The records may be either fixed or variable in length, and each must be wholly contained in one block. Blocks may contain more than one record. Each data record includes a record-length field and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

RECORDING MODE S: The records may be either fixed or variable in length, and may be larger than a block. If a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block (or blocks, if required). Only complete records are made available to the user. Each segment of a record in a block, even if it is the entire record, includes a segment-descriptor field, and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

For standard sequential files, the compiler determines the recording mode for a given file to be:

F if all the records are defined as being the same size and the size is smaller than or equal to the block size.

V if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records and the longest record is less than or equal to the maximum block size.

S if the maximum block size is smaller than the largest record size.

For direct files, the compiler determines the recording mode for a given file to be:

F if all the records are defined as being the same size and the size is smaller than or equal to the block size.

U if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records and the longest record is less than or equal to the maximum block size.

S if the maximum block size is smaller than the largest record size.

Files with indexed organization must have F mode records.

Note: ASCII considerations for compiler calculation of recording mode are given later in this Appendix.

PROCEDURE DIVISION DECLARATIVES

In segmented programs, priority numbers must not be specified in Declarative procedures.

Label Declaratives—Format 1

Label Declaratives allow the user to process labels.

Format 1

```
      [ BEFORE ]            [ BEGINNING ]  [ REEL ]
USE   <        >  STANDARD  |           |  | FILE |
 .    [ AFTER  ]            [ ENDING    ]  [ UNIT ]
 .
 .
 .                            [ {file-name} . . .  ]
 .                            | OUTPUT             |
 .      LABEL PROCEDURE ON    < INPUT              >
 .                            | I-O                |
 .                            | EXTEND             |
                              [                    ]
```

When BEFORE is specified, it indicates that nonstandard labels are to be processed. Nonstandard labels may be specified only for tape files.

When AFTER is specified, it indicates that user labels follow standard file labels, and are to be processed.

Note: ASCII considerations for user label-handling procedures are given later in this appendix.

The labels must be listed as data-names in the LABEL RECORDS clause in the file description entry for the file, and must be described as level -01 data items subordinate to the file entry.

If neither BEGINNING nor ENDING is specified, the designated procedures are executed for both beginning and ending labels.

If UNIT, REEL, or FILE is not included, the designated procedures are executed for REEL or UNIT, whichever is appropriate, and for FILE labels. The REEL option is not applicable to direct access storage files. The UNIT option is not applicable to files in the random access mode, since only FILE labels are processed in this mode.

If FILE is specified, the designated procedures are executed at beginning-of-file (on first volume) and/or at end-of-file (on last volume) only. If REEL or UNIT is specified, the designated procedures are executed at beginning-of-volume (on each volume but the first) and/or at end-of-volume (on each volume but the last). Both BEGINNING and ENDING label processing is executed if BEGINNING or ENDING has not been specified.

The same file-name may appear in different specific arrangements of Format 1. However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

If the file-name option is used, the file description entry for file-name must not specify a LABEL RECORDS ARE OMITTED clause.

The file-name must not represent a sort-file.

The EXTEND option is valid for sequential files.

The user label procedures are executed as follows when the OUTPUT, INPUT, or I-O options are specified:

- When OUTPUT is specified, only for files opened as OUTPUT
- When INPUT is specified, only for files opened as INPUT
- When I-O is specified, only for files opened as I-O

If the INPUT, OUTPUT, or I-O option is specified, and an input, output, or input/output file, respectively, is described with a LABEL RECORDS ARE OMITTED clause, the USE procedures do not apply.

The standard system procedures are performed:

- Before or after the user's beginning or ending input label check procedure is executed

- Before the user's beginning or ending output label is created

- After the user's beginning or ending output label is created, but before it is written on tape

- Before or after the user's beginning or ending input/output label check procedure is executed

Within the procedures of a USE declarative in which the USE sentence specifies an option other than file-name, references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but does not appear in any data record of this program. Such items must have identical descriptions and positions within each label record.

The exit from a Format 1 declarative section is inserted by the compiler following the last statement in the section. All logical program paths within the section must lead to the exit point.

There is one exception: A special exit may be specified by the statement GO TO MORE-LABELS. When an exit is made from a Format 1 declarative section by means of this statement, the system will do one of the following:

- Write the current beginning or ending label and then reenter the USE section at its beginning for further creating of labels. After creating the last label, the user must exit by executing the last statement of the section.

- Read an additional beginning or ending label, and then reenter the USE section at its beginning for further checking of labels. When processing user labels, the section will be reentered only if there is another user label to check. Hence, there need not be a program path that flows through the last statement in the section. For nonstandard labels, the compiler does not know how many labels exist. Therefore, the last statement in the section must be executed to terminate nonstandard label processing.

If a GO TO MORE-LABELS statement is not executed for a user label, the declarative section is not reentered to check or create any immediately succeeding user labels.

When reading nonstandard header labels, it is the user's responsibility to read any tape marks that are used to terminate labels. The GO TO MORE-LABELS exit must be used, and the declarative must recognize that a tapemark rather than data is being read. The final exit from the declarative must not be taken until the file is positioned just before the first data record.

The programmer must set special register LABEL-RETURN to nonzero if the nonstandard header label of an input file is not correct.

LABEL-RETURN is an alphanumeric item whose PICTURE is X. It may be used to indicate the validity of nonstandard labels. At the completion of a USE BEFORE STANDARD LABEL PROCEDURE for an input file, the programmer must set LABEL-RETURN to indicate the validity of the nonstandard label.

After the nonstandard trailer labels are processed, the system determines from the DD statement if another reel is to be read or if the current reel is the end of the file. If the current reel is the last one for the file, the statement executed is the one specified in the AT END phrase of the READ statement that detected the end-of-reel condition. If the current reel is not the last, a volume-switch takes place, the header label is processed, and the first record on the reel is read.

Error Declaratives—Format 2

Error Declaratives specify procedures to be followed if an input/output error occurs.

Format 2

```
USE AFTER STANDARD ERROR PROCEDURE
          ┌                                  ┐
          | file-name-1  [file-name-2] ...   |
          | INPUT                            |
   ON     <                                  >
          | OUTPUT                           |
          | I-O                              |
          └                                  ┘
            GIVING data-name-1 [data-name-2].
```

USE declaratives that specify error handling procedures are activated when an input/output error occurs during execution of a READ, WRITE, REWRITE, or START statement.

Automatic system error routines are executed before user-specified procedures.

Within the error procedure, the allowable statements that may be executed depend on the organization and access specified for the file in error.

When the file-name option is used, error handling procedures are executed for input/output errors occurring for the named file(s) only.

A file-name must not be referred to, implicitly or explicitly, in more than one Format 2 USE sentence.

The user error procedures are executed, when the INPUT, OUTPUT, or I-O option is specified and an input/output error occurs, as follows:

- When INPUT is specified, only for files opened as INPUT

- When OUTPUT is specified, only for files opened as OUTPUT

- When I-O is specified, only for files opened as I-O

An exit from this type of declarative section can be effected by executing the last statement in the section (normal return), or by means of a GO TO statement.

The normal return from an error declarative is to the statement following the input/output statement that caused the error.

User error handling procedures are executed for invalid key conditions if the INVALID KEY option is not specified in the statement causing the condition.

When the GIVING option is used and an error declarative section is entered, data-name-1 will contain the information shown in Figure 92 for non-VSAM files only.

The GIVING option may be specified with multiple file-names, or any of the INPUT/OUTPUT/I-O options.

Data-name-1 must be a 136-byte item. It must be defined in the Working-Storage Section. For additional information, see the appropriate Programmer's Guide.

If specified, data-name-2 contains the block in error for a READ operation, if data was transmitted. For a WRITE, REWRITE, or START operation, data-name-2 must not be referred to.

Data-name-2 must be an item large enough to hold the largest physical block which exists or which will be processed. It must be defined in the Working-Storage or Linkage Section. If data-name-2 is defined in the Linkage Section, the block in error is referenced in the buffer area; no storage need be defined for the error block.

Byte	Information		
0-7	System use		
08-13	Output Operation: always blanks Input Operation: Bytes 8-11: Input Buffer Address (used for data-name-2), or blanks Bytes 12-13: Number of bytes transmitted (size or error block) or blanks		
14-48	Blanks		
49	,		
50-57	Jobname		
58	:		
59-66	Stepname		
67	,		
68-70	Unit address		
71	,		
72-73	Device-type		
74	,		
75-82	ddname		
83	,		
84-89	Operation attempted		
90	,		
91-105	Error description		
106	,		
107-127	The contents of this field depend on the type of input/output device in use, as follows:		
	For unit record,	107-120	Asterisks
		121	,
		122-127	Access method
	For magnetic tape,	107-113	Block count (in decimal)
		114	,
		115-119	Access method
		120-127	Blanks
	For direct access storage,	107-120	Last actual address, in the form BBCCHHR (in hexadecimal)
		121	,
		122-127	Access method
128-135	System use		

Figure 92. Information Supplied With the GIVING Option When an Error Declarative is Entered

Notes for Figure 92

1. If data was transmitted during the input operation, bytes 8 through 13 contain binary data. If no data was transmitted, bytes 8 through 13 are blank.

2. Bytes 49 through 127, unless otherwise indicated, are in printable EBCDIC representation.

3. Bytes 91 through 105 (Error Description) contain a brief description of the type of error that occurred. The description is provided by the system error analysis procedures, and is placed into bytes 91 through 105 upon entry into the Error Declarative.

   For example, if the FD for an input file described 120-character records, and if at execution time the file actually contained 100-character records, then when a read operation was attempted an error would occur, and the system would return the message "WRN.LEN.RECORD".

INPUT/OUTPUT STATEMENTS

The statements described in this section give results identical with IBM OS Full American National Standard COBOL.

Valid clauses and statements for each type of file processing are summarized as follows:

- Figure 93—Standard Sequential Files
- Figure 94—Direct Files
- Figure 95—Indexed Files
- Figure 96—Relative Files

Figure 93. Standard Sequential Files—Required and Optional Entries

Device Type	Required Entries					Optional Entries						
	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY[3]	RESERVE	ACCESS	Other ENVIRONMENT DIVISION Clauses	BLOCK CONTAINS[4]	RECORDING MODE	USE
Reader	UR [-xxxx]-S-name	OMITTED	INPUT	[LOCK]	READ [INTO] AT END		{integer NO}	SEQUENTIAL	SAME [RECORD] AREA RERUN	[n TO] m	{F U V}	ERROR
Punch	UR [-xxxx]-S-name	OMITTED	OUTPUT	[LOCK]	WRITE [FROM] [{BEFORE AFTER} ADVANCING] [AFTER POSITIONING]	WRITE-ONLY (V-mode only)	{integer NO}	SEQUENTIAL	SAME [RECORD] AREA RERUN	[n TO] m	{F U V}	ERROR
Printer	UR [-xxxx]-S-name	OMITTED	OUTPUT	[LOCK]	WRITE [FROM] [{BEFORE AFTER} ADVANCING] [AFTER POSITIONING]	WRITE-ONLY (V-mode only)	{integer NO}	SEQUENTIAL	SAME [RECORD] AREA RERUN	[n TO] m	{F U V}	ERROR REPORTING
					[END-OF-PAGE]		NO					
Tape	UT [-xxxx]-S-name	{STANDARD OMITTED data-name [TOTALING- TOTALED]}	INPUT [REVERSED NO REWIND] [LEAVE REREAD DISP]	[REEL] [LOCK NO REWIND POSITIONING DISP]	READ [INTO] AT END	WRITE-ONLY (V-mode only)	{integer NO}	SEQUENTIAL	SAME [RECORD] AREA RERUN MULTIPLE FILE TAPE	[n TO] m	{F U V S}	LABEL ERROR
			OUTPUT [NO REWIND] [LEAVE REREAD DISP]	[REEL] [LOCK NO REWIND POSITIONING DISP]	WRITE[1] [FROM] [{BEFORE AFTER} ADVANCING] [AFTER POSITIONING]							LABEL ERROR REPORTING
Direct Access Storage	{UT DA}[-xxxx]-S-name	{STANDARD OMITTED data-name [TOTALING- TOTALED]}	INPUT	[UNIT] [LOCK]	READ [INTO] AT END	RECORD-OVERFLOW (not for S-mode)	{integer NO}	SEQUENTIAL	SAME [RECORD] AREA RERUN	[n TO] m	{F U V S}	AFTER LABEL ERROR
			OUTPUT	[UNIT] [LOCK]	WRITE[1] [FROM] INVALID KEY WRITE[1] [FROM] [{BEFORE AFTER} ADVANCING] [AFTER POSITIONING]	WRITE-ONLY (V-mode only)						AFTER LABEL ERROR REPORTING
			I-O	[LOCK]	READ [INTO] AT END WRITE[2] [FROM] INVALID KEY REWRITE[2] [FROM]							AFTER LABEL ERROR

[1] Create

[2] Update

[3] These APPLY clauses have meaning only for OUTPUT files; however, the compiler accepts them when the same file is opened for INPUT or I-O.

[4] Not for U-mode

Figure 94. Direct Files (Direct Access Storage Devices Only)—Required and Optional Entries

Required Entries							Optional Entries			
ACCESS	KEY	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	RECORDING MODE	APPLY[5]	Other ENVIRONMENT DIVISION Clauses	USE
[SEQUENTIAL]	[ACTUAL]	DA [-xxxx]-D-name	{STANDARD data-name}	INPUT	[UNIT] [LOCK]	READ [INTO] AT END	{F U V S}		SAME [RECORD] AREA RERUN	AFTER LABEL ERROR
[SEQUENTIAL]	ACTUAL	DA [-xxxx]-D-name	{STANDARD data-name}	OUTPUT	[UNIT] [LOCK]	WRITE[1] [FROM] INVALID KEY	F	RECORD-OVERFLOW	SAME [RECORD] AREA TRACK-LIMIT RERUN	AFTER LABEL ERROR
							{U V S}			
RANDOM	ACTUAL	DA [-xxxx]-D-name	{STANDARD data-name}	INPUT	[LOCK]	SEEK READ [INTO] INVALID KEY	{F U V S}		SAME [RECORD] AREA RERUN ON RECORDS	AFTER LABEL ERROR
				OUTPUT	[LOCK]	SEEK WRITE[1] [FROM] INVALID KEY	F	RECORD-OVERFLOW	SAME [RECORD] AREA TRACK-LIMIT RERUN ON RECORDS	
							{U V S}			
				I-O	[LOCK]	SEEK READ [INTO] INVALID KEY WRITE[2] [FROM] INVALID KEY	{F U V S}		SAME [RECORD] AREA RERUN ON RECORDS	
RANDOM	ACTUAL	DA [-xxxx]-W-name	{STANDARD data-name}	INPUT	[LOCK]	SEEK READ [INTO] INVALID KEY	{F U V S}		SAME [RECORD] AREA RERUN ON RECORDS	AFTER LABEL ERROR
				OUTPUT	[LOCK]	SEEK WRITE[1] [FROM] INVALID KEY	F	RECORD-OVERFLOW	SAME [RECORD] AREA TRACK-LIMIT RERUN ON RECORDS	
							{U V S}			
RANDOM	ACTUAL	DA [-xxxx]-W-name	{STANDARD data-name}	I-O	[LOCK]	SEEK READ [INTO] INVALID KEY WRITE[3] [FROM] INVALID KEY REWRITE[4] [FROM] [INVALID KEY]	{F U V S}		SAME [RECORD] AREA RERUN ON RECORDS	AFTER LABEL ERROR

[1] Create

[2] Update and add

[3] Add

[4] Update

[5] These APPLY clauses have meaning only for OUTPUT files; however, the compiler accepts them when the same file is opened for INPUT or I-O.

Figure 95. Indexed Files (Direct Access Storage Devices Only)—Required and Optional Entries

Required Entries							Optional Entries					
ACCESS	KEY	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY	RESERVE	Other ENVIRONMENT DIVISION Clauses	RECORDING MODE	BLOCK CONTAINS	USE
[SEQUENTIAL]	RECORD	DA [-xxxx] -I-name	STANDARD	INPUT	[LOCK]	READ [INTO] AT END		{integer NO}	SAME [RECORD] AREA RERUN	F	m	ERROR
	RECORD NOMINAL					START [INVALID KEY]						
	RECORD [NOMINAL]					START USING KEY [INVALID KEY]						
	RECORD			OUTPUT	[LOCK]	WRITE[1] [FROM] INVALID KEY						
	RECORD			I-O	[LOCK]	READ [INTO] AT END REWRITE[2] [FROM] [INVALID KEY]						
	RECORD NOMINAL					START [INVALID KEY]						
	RECORD [NOMINAL]					START USING KEY [INVALID KEY]						
RANDOM	RECORD NOMINAL	DA [-xxxx] -I-name	STANDARD	INPUT	[LOCK]	READ [INTO] INVALID KEY	REORG-CRITERIA[4] CORE-INDEX	NO	SAME [RECORD] AREA RERUN ON RECORDS	F	m	ERROR
				I-O	[LOCK]	READ [INTO] INVALID KEY WRITE[3] [FROM] INVALID KEY REWRITE[2] [FROM] [INVALID KEY]			SAME [RECORD] AREA TRACK-AREA RERUN ON RECORDS			

[1] Create

[2] Update

[3] Add

[4] This APPLY clause has meaning only for OUTPUT files; however, the compiler accepts it when the same file is opened for INPUT or I-O.

Required Entries							Optional Entries		
ACCESS	KEY	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	Other ENVIRONMENT DIVISION Clauses	RECORDING MODE	
[SEQUENTIAL]	[NOMINAL]	DA [-xxxx] -R-name	{STANDARD data-name}	INPUT	[UNIT] [LOCK]	READ [INTO] AT END	SAME [RECORD] AREA RERUN APPLY RECORD-OVERFLOW	F	ERROR AFTER LABELS
				OUTPUT	[UNIT] [LOCK]	WRITE[1] [FROM] INVALID KEY			
RANDOM	NOMINAL	DA [-xxxx] -R-name	{STANDARD data-name}	INPUT	[LOCK]	READ [INTO] INVALID KEY	SAME [RECORD] AREA RERUN ON RECORDS APPLY RECORD-OVERFLOW	F	ERROR AFTER LABELS
				I-O	[LOCK]	READ [INTO] INVALID KEY REWRITE[2] [FROM] [INVALID KEY]			

[1] Create [2] Update

Figure 96. Relative Files (Direct Access Storage Devices Only)—Required and Optional

OPEN Statement

The OPEN statement initiates the processing of files.

Format

```
                         ┌                  ┐  ┌        ┐
                         │ REVERSED         │  │ LEAVE  │
OPEN [INPUT {file-name   │                  │  │ REREAD │ }...]
                         │ WITH NO REWIND   │  │ DISP   │
                         └                  ┘  └        ┘
                                                  ┌        ┐
                                                  │ LEAVE  │
     [OUTPUT {file-name  [WITH NO REWIND]         │ REREAD │ }...]
                                                  │ DISP   │
                                                  └        ┘

     [I-O  {file-name}...]
```

Except for subsequent input/output statements, the OPEN statement does not make the buffer area available to the COBOL program.

If one is specified by a USE sentence in the Declaratives Section, the OPEN statement causes the user's beginning label subroutine to be executed.

If a sequential input file is designated with the OPTIONAL clause in the File Control paragraph of the Environment Division, the clause is treated as documentation. The desired effect is achieved by specifying the DUMMY or NULLFILE parameter in the DD statement for the file. If the parameter is specified, the first READ statement for this file causes control to be passed to the imperative statement after the key words AT END.

LEAVE, REREAD, and DISP may be specified only for standard sequential files. Because the positioning options are only applicable to tape files, they will be ignored if, at execution time, a direct access storage device is assigned to the file.

When the subsequent volume is not to be mounted on the same device, the LEAVE, REREAD, and DISP options define the positioning of volumes at end of volume in either of two cases:

- When automatic end of volume occurs (when an end-of-volume condition is detected during execution of a READ or WRITE statement).
- When execution of a CLOSE REEL/UNIT WITH POSITIONING statement causes forced end of volume.

The LEAVE option causes each affected volume to be positioned at the end of the file on the volume, unless the REVERSED option is also specified. If the REVERSED option is specified, the tape is positioned at the beginning (that is, the logical end) of the file on each volume affected.

Unless the REVERSED option is specified, the REREAD option causes each affected volume to be backspaced and positioned at the beginning of the file on the volume. If the REVERSED option is specified, the tape is repositioned at the end (that is, the logical beginning) of the file on each volume.

If the DISP option is specified, the action taken—such as rewind, unload, and so forth—is a function of the DISP parameter of the associated DD statement for the file. The action is the same, whether or not the REVERSED option is specified.

START Statement

The START statement initiates processing of a sequentially accessed indexed file at a specified full or generic key.

Format 1

```
START file-name [INVALID KEY imperative-statement]
```

Format 2

```
                                          ┌ EQUAL TO ┐
START file-name  USING KEY data-name     <            > identifier
                                          │    =     │
                                          └          ┘

      [INVALID KEY imperative-statement]
```

If processing is to begin at the first record in the file, a START statement is not required before the first READ statement.

FILE-NAME: The file-name must be defined by a file description entry in the Data Division.

FORMAT 1: When Format 1 is used, the contents of the NOMINAL KEY are used as the key value of the record at which processing is to begin. In this instance, this key value must be placed in the data-name specified by the NOMINAL KEY clause for this file before the START statement is issued.

When the INVALID KEY option is specified, control is passed to the imperative-statement following INVALID KEY when the contents of the NOMINAL KEY field are invalid. The key is considered invalid when the record is not found in the file.

In both Format 1 and Format 2, if the INVALID KEY option is not specified, an invalid key condition causes the execution of the USE AFTER STANDARD ERROR procedure, if specified, for the file. If neither is specified, abnormal termination may result.

FORMAT 2: When Format 2 is used, the programmer requests that processing begin with the first record of a specified generic key class.

Data-name must be the data-name specified in the RECORD KEY clause for the file.

Identifier contains the generic key value for the request, and may be any data item less than or equal in length to the RECORD KEY clause for the file. Identifier may not appear in the record description for the file.

The USAGE of data-name and identifier should be DISPLAY (USAGE IS DISPLAY).

When the USING KEY option is specified, then, before a START statement is issued, the user must place the desired value (the generic key) into identifier. When the START statement is executed, the contents of identifier are compared with the contents of the RECORD KEY data-name. The comparison is nonalgebraic, from left to right. The length of the comparison is controlled by the length of identifier. Sequential processing of the file begins at the first record whose RECORD KEY contains a match with the contents of identifier.

Identifiers of different lengths may be specified for different START statements for the same file.

Note that upon execution of a Format 2 START statement the contents of the NOMINAL KEY field associated with the file remain unchanged.

If identifier is greater in length than data-name, then the excess low-order characters of identifier are truncated.

In Format 2, when the INVALID KEY option is specified, and the contents of identifier are invalid, control is passed to the imperative-statement following INVALID KEY. Identifier is considered invalid when the generic key class it contains is not found in the file. However, if the first record of the specified key class has been deleted, retrieval begins at the next nondeleted record regardless of key class.

SEEK Statement

The SEEK statement serves as documentation.

Format

```
SEEK file-name RECORD
```

The SEEK statement is meant to initiate access to a direct access storage record for subsequent reading or writing. However, this compiler treats it as documentation.

The file-name must be defined by a file description entry in the Data Division.

A SEEK statement pertains only to direct files in the random access mode and may be executed prior to the execution of a READ or WRITE statement.

READ Statement

The READ statement makes available a logical record from a file.

Format

```
READ  file-name  RECORD [INTO identifier]
    [ AT END        ]
    <               >  imperative-statement
    | INVALID KEY   |
    L               J
```

AT END OPTION: If a DD card for a sequential file specifies the DUMMY or NULLFILE parameter, on the first READ for the file, control will be passed to the imperative statement in the AT END phrase. For purposes of language consistency, the OPTIONAL clause should be specified for this type of file.

INVALID KEY OPTION: If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL or NOMINAL KEY for the file must be set to the desired value before execution of the READ statement.

Only the track specified in the ACTUAL KEY is searched for the record being read.

If the desired record cannot be found on the specified track, the search can be extended to include a specific number of tracks or to include the entire file, with the LIMCT parameter on the DD card.

Control is passed to the imperative statement following INVALID KEY when the contents of the ACTUAL KEY or NOMINAL KEY field are invalid.

The key is considered invalid under the following conditions:

- For a direct file that is accessed randomly: when the record is not found within the search limits, or when the track address in the ACTUAL KEY field is outside the limits of the file.

- For an indexed file that is accessed randomly: when no record exists whose RECORD KEY field matches the contents of the NOMINAL KEY field.

- For a relative file that is accessed randomly: when the relative record number in the NOMINAL KEY field is outside the limits of the file.

When the execution of a READ statement for an indexed file causes an INVALID KEY condition, a REWRITE statement should not be executed for the record with that key.

WRITE Statement

The WRITE statement releases a record to a file.

Format 1

```
WRITE record-name  [FROM identifier-1]

                           ┌ identifier-2 ┐
  AFTER POSITIONING        <              >   LINES
                           | integer      |
                           └              ┘
      ┌ END-OF-PAGE ┐
  [AT <             >   imperative-statement]
      | EOP         |
      └             ┘
```

Format 2

```
WRITE record-name [FROM identifier-1]

    INVALID KEY imperative-statement
```

FORMAT 1: The POSITIONING option may be specified only for standard sequential files.

The POSITIONING option allows control of the vertical positioning of each record on the printed page. If the POSITIONING option is not used, automatic advancing is provided to cause single spacing. If the POSITIONING option is used, automatic advancing is overridden.

When the ADVANCING or POSITIONING option is written for a record in a file, every WRITE statement for records in the same file must also contain one of these options. The POSITIONING and ADVANCING options may not both be specified for a file.

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

In the AFTER POSITIONING option, identifier-2 must be described as a 1-character alphanumeric item; that is, with PICTURE X. Figure 97 shows the valid values that identifier-2 may assume and their interpretations.

Value of Identifier-2	Interpretation
b (blank)	Single-spacing
0	Double-spacing
-	Triple-spacing
+	Suppress spacing
1 - 9	Skip to channel 1 through 9, respectively
A, B, C	Skip to channel 10, 11, 12, respectively
V, W	Pocket select 1 or 2, respectively, on the IBM 1442 Card Reader, and P1 or P2 on the IBM 2540 Card Read Punch

Figure 97. Values of Identifier-2 and Interpretations—POSITIONING Option

The POSITIONING option integer must be unsigned, and must be the value 0, 1, 2, or 3. The values assume the meanings given in Figure 98.

Value of Integer	Interpretation
0	Skip to channel 1 of next page (carriage control)
1	Single-spacing
2	Double-spacing
3	Triple-spacing

Figure 98. Values of Integer and Interpretations—POSITIONING Option

If the AFTER POSITIONING option is used, the record is written after the printer page is advanced according to the preceding rules.

Note: DISPLAY, EXHIBIT, and WRITE AFTER POSITIONING statements all cause the printer to space before printing. However, a simple WRITE statement with no option given, or a WRITE BEFORE ADVANCING statement causes the printer to space after printing. Therefore, it is possible that mixed DISPLAY, EXHIBIT, and simple WRITE statements or WRITE BEFORE ADVANCING statements within the same program may cause overprinting.

END-OF-PAGE-OPTION: The END-OF-PAGE condition exists when the channel 12 punch on the carriage control tape is sensed by an online printer. The printer file must be defined as an unblocked single buffered file. The programmer should ensure that every WRITE statement in the program (whether using the ADVANCING or the POSITIONING option) advances the printer only one line at a time; otherwise, the channel 12 punch may not be sensed and results may be unpredictable.

When an END-OF-PAGE condition exists, the writing and spacing operations are completed before the END-OF-PAGE imperative statement is executed. The END-OF-PAGE statement will be executed only for an online printer.

FORMAT 2: This format is used for randomly or sequentially accessed direct access storage files.

For standard sequential files opened as OUTPUT, the WRITE statement can be specified only to create the file. For such files opened as I-O, a READ statement must be executed before the WRITE statement is issued; the WRITE statement updates the record retrieved by the previous READ statement.

For sequentially accessed direct files, the WRITE statement updates or creates a record for an OUTPUT file. If a record with the same ACTUAL KEY already exists, the WRITE statement updates that record; otherwise, it creates a new record.

For sequentially accessed indexed or relative files, the WRITE statement creates a record for an OUTPUT file.

If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL or NOMINAL KEY field for the file must be set to the desired value before the execution of a WRITE statement. For a direct file, the track specified in the ACTUAL KEY field is searched for space for the record to be written.

If the required space cannot be found or if the record is not found on the specific track, the search can be extended to include a specific number of tracks, or to include the entire file, with the LIMCT parameter on the DD card.

INVALID KEY OPTION: The INVALID KEY option must be specified for a file that resides on a direct access storage device; however, this IBM implementation will allow the user to omit this option. If the INVALID KEY option is not specified, an invalid key condition causes the execution of the USE AFTER STANDARD ERROR procedure if specified for the file. If no error processing declarative is specified for the file, the invalid key condition is ignored.

Control is passed to the imperative statement following INVALID KEY if the following conditions exist:

- For a direct access storage file in the sequential access mode and opened as OUTPUT: when no space is available in which to write the record.

- For a direct file in the random access mode and opened as I-O or OUTPUT: when a record is being added to the file, and any one of the following conditions occurs:

  - The track number specified in the ACTUAL KEY field is outside the limits of the file.

  - For files with mode F records, the figurative constant HIGH VALUE (or its equivalent) has been moved into the first character position of the symbolic portion of the ACTUAL KEY field.

- For a direct file in the random access mode, opened as I-O, and a record is being updated: when the record is not found, or when the track number in the ACTUAL KEY field is outside the limits of the file.

- For an indexed file in the sequential access mode, opened as OUTPUT, and either of the following conditions occurs:

  - The contents of the RECORD KEY field are not in ascending order when compared with the contents of the RECORD KEY field of the preceding record. The contents of the RECORD KEY field duplicate those of the preceding record.

- For an indexed file in the random access mode, opened as I-O, and a record is being added to the file: when the contents of the NOMINAL KEY field associated with the record to be added duplicate the contents of a RECORD KEY field already in the file.

RANDOMLY ACCESSED DIRECT FILES: For a direct file in the random access mode that is opened for I-O, the following considerations apply:

- If D is specified in the ASSIGN clause system-name, then:
  - A WRITE statement updates a record if the preceding READ statement was for a record with the same ACTUAL KEY.
  - A WRITE statement adds a new record to the file, whether or not a duplicate record exists, if the preceding READ statement was not for a record with the same ACTUAL KEY.
- If W is specified in the ASSIGN clause system-name, then:
  - A REWRITE statement searches for a record with a matching ACTUAL KEY, and updates it.
  - A WRITE statement adds a new record to the file, whether or not a duplicate key exists.

REWRITE Statement

The REWRITE statement replaces a logical record on a direct access storage file.

Format

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

The READ statement for a file must be executed before a REWRITE statement for the file can be executed, except for a direct file accessed randomly. A REWRITE statement can be executed only for files opened as I-O; any file organization is valid.

When the FROM option is used, the REWRITE statement is equivalent to the statement MOVE identifier TO record-name followed by the statement REWRITE record-name. Identifier should be defined in the Working-Storage Section, the Linkage Section, or in another FD.

For a randomly accessed direct file, control is passed to the imperative statement following INVALID KEY when the contents of the ACTUAL KEY field are invalid. The key is considered invalid when the record is not found, or the track address specified in ACTUAL KEY is outside the limits of the file.

For a randomly accessed relative file, control is passed to the imperative statement following INVALID KEY when the contents of the NOMINAL KEY field are invalid. The key is considered invalid when the relative record number in the NOMINAL KEY field is outside the limits of the file.

An INVALID KEY error will never be detected when updating a randomly accessed indexed file, and the results of a REWRITE statement are unpredictable. If, when randomly reading a record of an indexed file, an INVALID KEY condition occurs, the record should not be rewritten. If the INVALID KEY option is not specified, an invalid key condition will cause the execution of the USE AFTER STANDARD ERROR procedure, if specified for the file. If no error processing declarative is specified for the file, the invalid key condition will be ignored.

If ACCESS IS RANDOM is specified for the direct or relative file, the ACTUAL or NOMINAL KEY must be set to the desired value before the REWRITE statement is executed.

CLOSE Statement

Format 1

```
                     ┌ REEL ┐        ┌ NO REWIND ┐
CLOSE  file-name-1   │      │  [WITH <           >    ]
                     │ UNIT │        │ LOCK      │
                     └      ┘        └           ┘
                     ┌ REEL ┐        ┌ NO REWIND ┐
      [file-name-2   │      │  [WITH <           >    ]] ...
                     │ UNIT │        │ LOCK      │
                     └      ┘        └           ┘
```

Format 2

```
                                ┌ NO REWIND ┐
CLOSE   file-name-1    [WITH    < LOCK      >   ]
                                │ DISP      │
                                └           ┘
                                ┌ NO REWIND ┐
       [file-name-2    [WITH    < LOCK      >   ] ] ...
                                │ DISP      │
                                └           ┘
```

Format 3

```
                     ┌ REEL ┐          ┌ NO REWIND   ┐
CLOSE file-name-1    <      >  [WITH   < LOCK        >  ]
                     │ UNIT │          │ POSITIONING │
                     └      ┘          └             ┘
                     ┌ REEL ┐          ┌ NO REWIND   ┐
     [file-name-2    <      >  [WITH   < LOCK        >   ]] ...
                     │ UNIT │          │ POSITIONING │
                     └      ┘          └             ┘
```

A file may be closed more than once, but each CLOSE statement (without the REEL/UNIT option) must be preceded by an OPEN statement for that file. A file that is opened within a run unit must be closed within that run unit.

The REEL, DISP, WITH POSITIONING, and WITH NO REWIND options are applicable only to tape files. The UNIT option is applicable only to direct access storage files in sequential access mode. Since device assignments can be specified at execution time, the words REEL and UNIT are interchangeable. If a file is assigned to a direct access storage device, the DISP, WITH POSITIONING, and NO REWIND options will be ignored.

For purposes of showing the effect of various CLOSE options as applied to various storage media, all input/output files are divided into the following categories:

- Unit record. A file whose input or output medium is such that rewinding, units, and reels have no meaning.

- Sequential single volume. A sequential file that is entirely contained on one volume. There may be more than one file on this volume.

- Sequential multivolume. A sequential file that may be contained on more than one volume.

- Random single volume. A file in the random access mode that may be contained on a single direct access storage volume.

- Random multivolume. A file in the random access mode that may be contained on more than one direct access storage volume.

Note: See also the file processing charts (Figure 90 through Figure 92) at the beginning of this section.

Sequential File Processing

The results of executing each CLOSE option for each type of file are summarized in Figure 99. The definitions of the symbols in the figure are given below. Where the definition of the symbol depends on whether the file is an input or output file, alternative definitions are given; otherwise, a definition applies to files opened as INPUT, OUTPUT, and I-O.

	File Types		
CLOSE Option	**Unit Record**	**Sequential Single-Volume**	**Sequential Multivolume**
CLOSE	C	C, G	C, G, A
CLOSE WITH LOCK	C, E	C, G, E	C, G, E, A
CLOSE WITH NO REWIND	X	C, B	C, B, A
CLOSE WITH DISP	X	C, J	C, J, A
CLOSE REEL	X	X	F, G
CLOSE REEL WITH LOCK	X	X	F, G, D
CLOSE REEL WITH NO REWIND	X	X	F, B
CLOSE REEL WITH POSITIONING	X	X	F, H
CLOSE UNIT	X	X	F
CLOSE UNIT WITH LOCK	X	X	F, D
CLOSE UNIT WITH POSITIONING	X	X	F

Figure 99. Sequential File Types and Options of the CLOSE Statement

A—Previous Volumes Unaffected
All volumes in the file prior to the current volume are processed according to standard volume switch procedures except those volumes controlled by a prior CLOSE REEL/UNIT statement. The standard switch procedure positions the volumes as specified by the volume positioning option of the OPEN statement.

B—No Rewind of Current Reel
The current volume is left in its current position.

C—Standard Close File

Files opened as INPUT and I-O: If the file is positioned at its end, and label records are specified, the standard ending label procedure and the user ending label procedure (if specified by the USE statement) are performed. The order of execution of these two procedures is specified by the USE statement. Standard system closing procedures are then performed.

If the file is positioned at its end, and label records are not specified for the file, standard system closing procedures are performed.

If the file is positioned other than at its end, the standard system closing procedures are performed. Even if label procedures are specified, no label processing is performed.

(An INPUT or I-O file is considered to be at its end if the AT END phrase of the READ statement has been executed, and no CLOSE statement has been executed.)

Files opened as OUTPUT: If label records are specified for the file, standard ending label procedures and user ending label procedures (if specified by the USE statement) are performed. The order of execution of these two procedures is specified by the USE statement. Standard system closing procedures are then performed.

If label records are not specified for the file, standard system closing procedures are performed.

D—Standard Reel/Unit Lock

This feature has no meaning in this system and is treated as comments.

E—Standard File Lock

The compiler ensures that this file cannot be opened again during this execution of the object program.

F—Standard Close Volume

Files Opened as INPUT and I-O: The following operations are performed:

- A volume switch.
- The standard beginning volume label procedure and the user's beginning volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.
- Makes the next data record on the new volume available to be read.

Files Opened as OUTPUT: The following operations are performed:

- The standard ending volume label procedure and the user's ending volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.
- A volume switch.
- The standard beginning volume label procedure and the user's beginning volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

G—Rewind

The current volume is positioned at its beginning.

H—POSITIONING of Current Reel
The current volume is positioned as specified by the volume positioning option of the OPEN statement.

J—DISP
The positioning of the current volume (such as rewind, unload, and so forth) is a function of the DISP parameter of the associated DD statement for the file. The action is the same, whether or not the file was opened as REVERSED.

X—Illegal
This is an illegal combination of the CLOSE option and a file type. The results at object time may be unpredictable.

General Considerations: A file is designated as optional by specifying the DUMMY or NULLFILE parameter on the DD card for the file. If an optional file is not present, standard end-of-file processing is not performed. For purposes of language consistency, the OPTIONAL phrase of the SELECT clause should be specified for this type of file.

If a CLOSE statement without the REEL or UNIT option has been executed for a file, the next input/output statement to be executed for that file must be an OPEN statement.

Random File Processing

The results of executing each CLOSE option for each type of file are summarized in Figure 100. The definitions of the symbols in the figure are given below. Where the definition depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to files opened as INPUT, OUTPUT, and I-O.

K—Standard Close File
The standard ending label procedure and the user ending label procedure (if specified by the USE statement) are performed. For I-O files and OUTPUT files, the labels are written. Standard system closing procedures are then performed.

L—Standard File Lock
The compiler ensures that this file cannot be opened again during this execution of this object program.

	File Types	
CLOSE Option	**Random Single-Volume**	**Random Multivolume**
CLOSE	K	K
CLOSE WITH LOCK	K, L	K, L

Figure 100. Random File Types and Options of the CLOSE Statement

ASCII FILE CONSIDERATIONS

For ASCII-encoded files using "old" sequential organization, there are special considerations, as follows.

ASSIGN Clause

The Environment Division ASSIGN clause must be specified as follows.

The assignment-name has the following format:

UT[-device]-C[-offset]-name

Device, if specified, must specify a magnetic tape device. If this field is omitted, the magnetic tape device must be specified through control cards at execution time.

C in the organization field specifies that an ASCII-encoded sequential file is to be processed, or that an ASCII-collated sort is to be performed.

Offset may be specified only for an ASCII file, and then only if a buffer offset in the range 01 through 99 exists. It is a 2-digit field, and may be specified as follows.

- 01 through 99 for an input file
- 04 for an output file (D-mode records only)

Name is a 1- to 8-character field specifying the external-name by which the file is known to the system. It is the name that appears in the name field of the DD card for the file.

For an ASCII-collated sort, assignment-name has the following format:

class[-device]-C-name

Note: For the sort assignment-name, the buffer offset field is not permitted.

RECORDING MODE Clause

For ASCII files, mode may be specified as F, U, or V. S mode may not be specified.

LABEL Procedure Declarative

Because the user may not specify nonstandard labels for an ASCII-encoded file, the BEFORE option of the LABEL PROCEDURE Declarative is not allowed.

Relation Conditions

If the ASCII character strings to be compared contain mixed alphabetic/ numeric characters and/or special characters, then the TRANSFORM verb can be used before the comparison is made to ensure a valid comparison.

SECTION II—"NEW" AND "OLD" LANGUAGE EXTENSIONS

The language extensions documented in this section may be specified in a program for which either the "new" or the "old" compiler option is specified. All are extensions to the 1974 standard.

Note: These extensions are included for compatibility purposes only. Therefore, whether the "new" or the "old" compiler option is chosen, they must be specified only in conjunction with 1968 Standard language or with IBM extensions to that Standard.

LANGUAGE CONSIDERATIONS

Floating-Point Numeric Literals

A floating-point numeric literal is an item whose potential range of value is too great for fixed-point representation. A floating-point literal must have the form:

[±] mantissa E [±] exponent

A floating-point literal must appear as a continuous string of characters with no intervening spaces. The plus-or-minus signs preceding the mantissa and exponent are the only optional characters within the format. The mantissa consists of from 1 through 16 digits with a required decimal point.

The exponent is represented immediately to the right of the mantissa by the symbol E, followed by a plus-or-minus sign (if a sign is given) and one or two digits. The magnitude of the number represented by a floating-point literal must not exceed 0.72 x (10^{76}). A zero exponent must be written as 0 or 00. It is assumed that an unsigned exponent is positive.

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

Special Registers

TALLY: is the name of a special register whose implicit description is that of an integer of five digits without an operational sign, and whose implicit USAGE is COMPUTATIONAL. The primary use of the TALLY register is to hold information produced by the EXAMINE statement. References to TALLY may appear wherever an elementary data item of integral value may appear.

CURRENT-DATE: is an 8-byte alphanumeric field, valid only as the sending field in a MOVE statement. The format of these eight bytes is MM/DD/YY (month/day/year).

TIME-OF-DAY: is a 6-byte external-decimal field, valid only as the sending field in a MOVE statement. The format is HHMMSS (hour, minute, second).

IDENTIFICATION DIVISION

Program-name may be specified within quotation marks.

The REMARKS paragraph allows a comment-entry to be specified. The comment-entry may consist of any characters within the EBCDIC set.

Level-Numbers

This compiler accepts nonstandard level-numbers which are not identical with others at the same level. For example, the compiler will accept the following COBOL descriptions as equivalent:

```
01 A.                               01 A.
   05  C-1.                            05  C-1.
       10  D PICTURE X.                    10  D PICTURE X.
       10  E PICTURE 999.                  10  E PICTURE 999.
   05  B-1.                            04  B-1.
       10  F PICTURE 9.                    08  F PICTURE 9.
       10  G PICTURE A(5).                 08  G PICTURE A(5).
```

REDEFINES Clause

For level-numbers that are subordinate to a record, the compiler allows the redefining area (data-name-1) to be smaller than the redefined area (data-name-2).

When multiple redefinition is used, this compiler accepts the data-name of the preceding entry as a valid redefinition. For example:

```
02  A  PICTURE 99V99.
02  B  REDEFINES A  PICTURE A(4).
02  C  REDEFINES B  PICTURE X(4).
```

PICTURE Clause

When the asterisk (*) zero suppression symbol and the BLANK WHEN ZERO clause are both specified for the same data item, the asterisk overrides the BLANK WHEN ZERO clause.

USAGE Clause—Floating-Point Items

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

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

EXTERNAL FLOATING-POINT ITEMS: These items have USAGE DISPLAY and a PICTURE character-string in the following form:

{±} mantissa E {±} exponent

where each element of the string is composed according to the following rules:

± A plus sign or a minus sign must immediately precede both the mantissa and the exponent in the PICTURE character string

+ indicates that a plus sign in the data represents positive values and that a minus sign represents negative values.

- indicates that a space character in the data represents positive values and that a minus sign represents negative values.

  The plus sign, the space character, and the minus sign each occupy one byte of storage.

mantissa The mantissa immediately follows the first sign character, and is represented using the following three symbols:

9 Each 9 in the mantissa character string represents a digit position into which a numeric character will be placed. From one to sixteen 9s may be present in the string. Each digit position occupies one byte of storage.

. indicates an actual decimal point. It does not take up any storage.

V indicates an assumed decimal point. It does not take up any storage.

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

E indicates the exponent, and immediately follows the mantissa. It occupies one byte of storage.

exponent The exponent immediately follows the second sign character. It is represented by two consecutive 9s. Each occupies one byte of storage.

The value of an external floating-point number is the mantissa multiplied by the power of 10 expressed by the exponent. The magnitude of a number represented by a floating-point item must be greater than 5.4 x (10^{-79}) but must not exceed 0.72 x (10^{76}). When used as a numeric operand, or as the sending item in a move statement, an external floating-point number is scanned at object time, and converted to its equivalent internal floating-point value. In this form, the number is used in arithmetic or move operations.

External data must conform to the representation specified in the PICTURE clause.

USAGE DISPLAY must be specified or implied for external floating-point items.

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

INTERNAL FLOATING-POINT ITEMS: These items are specified through the following options of the USAGE CLAUSE:

(USAGE COMPUTATIONAL-1) for short-precision internal

(USAGE COMP-1) floating-point (4 bytes long)

(USAGE COMPUTATIONAL-2) for long-precision internal

(USAGE COMP-2) floating-point (8 bytes long)

Note: Because of the size limitation for an internal floating-point item, there is an inherent inaccuracy in the conversion from decimal to hexadecimal. Because of this limitation, low order significance may be lost. For example, 0.18200 external decimal would convert to 0.1819999999 in internal decimal.

Internal floating-point items are equivalent to external floating-point items in capability and purpose.

The sign of the fraction (mantissa) is the leftmost bit (bit 0) in either format.

The exponent appears in bits 1 through 7.

The fraction (equivalent to the mantissa) appears in the rightmost bytes:

For COMPUTATIONAL-1, the fraction is 3 bytes long.

For COMPUTATIONAL-2, the fraction is 7 bytes long.

No PICTURE clause may be associated with an internal floating-point item.

If a VALUE clause is associated with an internal floating-point item, the literal must be a floating-point literal.

VALUE Clause

In the File Section, Linkage Section, and the Communication Section, the VALUE clause may be specified for level-88 items, and also for other items.

Using JUSTIFIED in a VALUE clause, under the "old" language options, will cause initialization to be right justified.

JUSTIFIED IN A VALUE CLAUSE: The "new" language options ignore the JUSTIFIED clause, and initialization will be left justified. In the latter case, it is treated as documentation.

PROCEDURE DIVISION

Section Headers

If a section header is missing at the beginning of the nondeclarative portion of the Procedure Division, the compiler processes the source program as if a section header had been written.

THEN Reserved Word

The word THEN may be used to separate a series of statements.

Floating-Point Numeric Operands

The following rules apply to the use of floating-point numeric operands:

RELATION CONDITION: For external floating-point operands, the rules for the comparison of external decimal operands apply; for internal floating-point operands, the rules for the comparison of internal decimal operands apply.

ACCEPT FROM DATE/DAY/TIME STATEMENT: The receiving field can be an external floating-point item.

DISPLAY STATEMENT: Internal floating-point items are converted to external floating-point items before printing.

MOVE STATEMENT: Both external and internal floating-point items are treated as numeric noninteger items.

ROUNDED OPTION: Rounding has no meaning when the result field is floating-point. When the result field is fixed-point, however, and at least one of the operands of an arithmetic statement is floating-point, rounding always takes place, whether or not ROUNDED is specified.

SEARCH STATEMENT: The identifier must not be a floating-point item.

SIZE ERROR OPTION: For internal floating-point items, only division by 0 causes the SIZE ERROR imperative-statement to be executed.

Arithmetic Expressions—Unary Operators

A unary operator need not be followed by a space.

Comparisons

The compiler allows a group item to be compared with a numeric item, even when the USAGE of the numeric item is not DISPLAY.

IF Statement

The following format is accepted by the compiler.

Format

```
                    [ statement-1    ]   [ ELSE      ]   [ statement-2    ]
IF condition THEN   <                >   <           >   <                >
                    | NEXT SENTENCE  |   | OTHERWISE |   | NEXT SENTENCE  |
                    L                J   L           J   L                J
```

THEN is accepted as a connective after the condition.

The word OTHERWISE is the equivalent of the word ELSE.

Execution of the IF statement, when these words are coded, is unaffected.

EXAMINE Statement

The EXAMINE statement is used to count the number of times a specified character appears in a data item and/or to replace a character with another character. **Note:** EBCDIC collating sequence is always used for comparisons, no matter which collating sequence is used by the program.

Format 1

```
                                  [ UNTIL FIRST ]
EXAMINE identifier TALLYING       < ALL         >   literal-1
                                  | LEADING     |
                                  L             J
          [REPLACING BY literal-2]
```

also see "INSPECT"
page 115

Format 2

```
                              ┌             ┐
                              │ ALL         │
                              │ LEADING     │
EXAMINE identifier REPLACING  < FIRST       > literal-1
                              │ UNTIL FIRST │
                              └             ┘
         BY literal-2
```

In all cases, the description of identifier must be such that its USAGE is DISPLAY (explicitly or implicitly).

When identifier represents a nonnumeric data item, examination starts at the leftmost character and proceeds to the right. Each character in the data item is examined in turn. For purposes of the EXAMINE statement, external floating-point items are treated as nonnumeric data items.

When identifier represents a numeric data item, this data item must consist of numeric characters, and may possess an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn.

If the letter 'S' is used in the PICTURE of the data item description to indicate the presence of an operational sign, the sign is ignored by the EXAMINE statement.

Each literal must consist of a single character belonging to a class consistent with that of the identifier; in addition, each literal may be any figurative constant except ALL. If identifier is numeric, each literal must be an unsigned integer or the figurative constant ZERO (ZEROES, ZEROS).

When Format 1 is used, an integral count is created which replaces the value of a special register called TALLY, whose implicit description is that of an unsigned integer of five digits (see "Language Considerations" in this Appendix).

- When the ALL option is used, this count represents the number of occurrences of literal-1.
- When the LEADING option is used, this count represents the number of occurrences of literal-1 prior to encountering a character other than literal-1.
- When the UNTIL FIRST option is used, this count represents all characters encountered before the first occurrence of literal-1.

Whether Format 2 is used, or the REPLACING option of Format 1, the replacement rules are the same. They are as follows:

- When the ALL option is used, literal-2 is substituted for each occurrence of literal-1.
- When the LEADING option is used, the substitution of literal-2 for each occurrence of literal-1 terminates as soon as a character other than literal-1 or the rightmost boundary of the data item is encountered.
- When the UNTIL FIRST option is used, the substitution of literal-2 terminates as soon as literal-1 or the far rightmost boundary of the data item is encountered.
- When the FIRST option is used, the first occurrence of literal-1 is replaced by literal-2.

NOTE Statement

Using the NOTE statement, the programmer can write commentary that will be produced on the source listing but not compiled.

Format

NOTE character string

Any combination of characters from the EBCDIC set may be included in the character string.

If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be part of the character string. Proper format rules for paragraph structure must be observed.

If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first occurrence of a period followed by a space.

MOVE Statement

When MOVE CORRESPONDING is specified, multiple receiving operands are allowed.

TABLE HANDLING FEATURE

There are special considerations for the OCCURS clause and for the USAGE IS INDEX clause when this feature is used.

OCCURS Clause

The OCCURS clause specifies the number of times a series of items with identical format is to be repeated; it also specifies keys and indexes associated with these items.

In all formats with the OCCURS DEPENDING ON clause, this IBM implementation allows any subordinate entry to be variable in length—that is, to contain an OCCURS DEPENDING ON clause.

Format 2

```
OCCURS lnteger-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] ... ]
```

Format 3

```
OCCURS 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] ... ]
```

An entry containing an OCCURS DEPENDING ON clause need not be the last in the record.

In Format 2, integer-1 represents the minimum number of occurrences. The value of integer-1 must be 0 or greater; it must be less than integer-2.

In Formats 2 and 3, integer-2 represents the maximum number of occurrences. The value in data-name-1 must not exceed that of integer-2.

USAGE IS INDEX Clause

When USAGE IS INDEX is specified for a data item, this IBM implementation allows the specification of the SYNCHRONIZED clause. The SYNCHRONIZED clause is treated as documentation.

SEARCH ALL Statement

In the WHEN option, the KEY data item may be either the subject or the object of the EQUAL TO relation-condition. However, if the object of the EQUAL TO is the KEY data item, then the subject identifier must not contain a literal subscript.

DEBUGGING LANGUAGE

Debugging statements may appear anywhere in a COBOL program or in a compile-time debugging packet.

Output for the TRACE and EXHIBIT statements is written on the system logical output device; a maximum logical record size of 120 characters is assumed. This assumed size can be overridden at compile time.

READY or RESET TRACE Statement

Format

```
┌ READY ┐
<       >    TRACE
│ RESET │
└       ┘
```

After a READY TRACE statement is executed, and each time execution of a paragraph or section begins, its name or compiler-generated card number—depending on the compiler option chosen—is displayed. The execution of a RESET TRACE statement terminates the functions of a previous READY TRACE statement.

EXHIBIT Statement

Format

```
               ┌ NAMED          ┐   ┌ identifier-1 ┐
EXHIBIT        < CHANGED NAMED  >   <              >
               │ CHANGED        │   │ literal-1    │
               └                ┘   └              ┘

    ┌                ┐
    │ identifier-2   │
    │                │ ...
    │ literal-2      │
    └                ┘
```

The execution of an EXHIBIT statement causes a formatted display of the identifiers (or literals) listed in the statement.

Identifiers listed in the statement cannot be any special register except TALLY.

Literals listed in the statement are followed by a blank when displayed.

The display of the operands is continued as described for the DISPLAY statement. A maximum logical record size of 120 characters is assumed.

EXHIBIT NAMED: Each time an EXHIBIT NAMED statement is executed, there is a formatted display of each identifier listed and its value. Because both the identifying name and the value of the identifier are displayed, a fixed columnar format is unnecessary. If the list of operands includes literals, they are displayed as remarks each time the statement is executed.

EXHIBIT CHANGED NAMED: Each time an EXHIBIT CHANGED NAMED statement is executed, there is a display of each identifier listed and its value (only if the value has changed since the previous time the statement was executed). The first time such a statement is executed, all values are considered changed and are displayed. If the list of operands includes literals, they are displayed as remarks each time the statement is executed.

Because both the identifying name and the value of each identifier are displayed, a fixed columnar format is unnecessary. If some of the identifiers have not changed in value, no space is reserved for them. If none of the identifiers have changed in value, blank line(s) will not be printed.

The format of the output for each identifier listed in the EXHIBIT NAMED and EXHIBIT CHANGED NAMED statement is:

original identifying name, including qualifiers if written (no more than 120 characters in length)

space

equal sign

space

value of identifier (for EXHIBIT CHANGED NAMED, no more than 256 bytes in length)

space

EXHIBIT CHANGED: Each time an EXHIBIT CHANGED statement is executed, there is a display of the current value of each identifier listed (only if the value has changed since the previous time the statement was executed). The first time the statement is executed, all values are considered changed and are displayed. If the list of operands includes literals, they are printed as remarks each time the statement is executed.

The format of the output for a specific EXHIBIT CHANGED statement presents each operand in a fixed columnar position. Because the operands are displayed in the order they are listed in the statement, the programmer can easily distinguish each operand.

The following considerations apply:

- If there are two or more identifiers as operands, and some, but not all, are changed from the previous execution of the statement, only the changed values are displayed. The positions reserved for a given operand are blank when the value of the operand has not changed.

- If none of the operands have changed in value from the previous execution of the statement, a blank line(s) will be printed.

- Variable-length identifiers are not permitted as operands.

- The storage reserved for any operand cannot exceed 256 bytes.

Note: The combined total length of all operands for all EXHIBIT CHANGED NAMED plus all EXHIBIT CHANGED statements in one program cannot exceed 32767 bytes.

If two distinct EXHIBIT CHANGED NAMED or two EXHIBIT CHANGED statements appear in one program, each specifying the same identifiers, the changes in value of those identifiers are associated with each of the two separate statements. Depending on the path of program flow, the values of the identifier saved for comparison may differ for each of the two statements.

ON Statement

The ON statement allows selective execution of the statements it contains.

Format

```
    ┌              ┐                   ┌              ┐
    | integer-1    |                   | integer-2    |
ON  <              >    [AND EVERY     <              >   ]
    | identifier-1 |                   | identifier-2 |
    └              ┘                   └              ┘
               ┌              ┐        ┌                      ┐
               | integer-3    |        | imperative-statement |
      [UNTIL   <              >  ]     <                      >
               | identifier-3 |        | NEXT SENTENCE        |
               └              ┘        └                      ┘
    ┌           ┐    ┌               ┐
    | ELSE      |    | statement...  |
    <           >    <               >
    | OTHERWISE |    | NEXT SENTENCE |
    └           ┘    └               ┘
```

All integers contained in the ON statement must be positive and no greater than 16777215.

The phrase ELSE/OTHERWISE NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

All identifiers must be fixed-point numeric items described as integers. Their values must be positive and no greater than 16777215.

At object time, each identifier must be initialized to a positive value before the first execution of the ON statement. Between executions of the ON statement, the values contained in the identifiers may be modified. The programmer's manipulation of these values in no way affects the compiler-generated counter associated with the ON statement.

In the discussion that follows, each reference to integer-1 applies equally to identifier-1. Similarly, each reference to integer-2 applies to identifier-2, and each reference to integer-3 applies to identifier-3.

The ON statement is evaluated and executed as follows:

- Each ON statement has a compiler-generated counter associated with it. The counter is initialized to 0 in the object program. Each time the path of program flow reaches the ON statement, the counter is incremented by one, and the count-condition (integer-1 AND EVERY integer-2 UNTIL integer-3) is tested. When the counter reaches 16777215, it is reset to 0 and counting resumes.

- If the count-condition is satisfied, the imperative-statement (or NEXT SENTENCE) preceding ELSE/OTHERWISE is executed. (Note that an imperative-statement may consist of a series of imperative statements.)

- If the count-condition is not satisfied, the statement(s) (or NEXT SENTENCE) following ELSE/OTHERWISE is executed. If the ELSE/OTHERWISE option does not appear, the next sentence is executed.

The count-condition is evaluated as follows:

- If only integer-1 has been specified, then the count-condition is satisfied only once: when the path of program flow has reached the ON statement integer-1 times—that is, when the value in the counter equals integer-1.
- When only integer-1 and integer-3 are specified, then the value of integer-2 is assumed to be one, and the count-condition is satisfied when the value in the counter is any value within the range from integer-1 through integer-3.
- If only integer-1 and integer-2 are specified, then the count-condition is satisfied each time the value in the counter is equal to integer-1 + (integer-2 * K), where K is any positive integer or 0. No upper limit for the execution of the ON statement is assumed.
- When all three integers are specified, then the count-condition is satisfied as in the last preceding case, except that an upper limit at which the count-condition cannot be satisfied is specified. The upper limit is integer-3.

Note: In called subprograms, reinitialization of the EXHIBIT statement and ON statement operands is the responsibility of the programmer.

COMPILE-TIME DEBUGGING PACKET

Debugging statements for a given paragraph or section in a program may be grouped together into a debugging packet. These statements will be compiled with the source language program and will be executed at object time. Each packet refers to a specified paragraph-name or section-name in the Procedure Division. Compile-time debugging packets are grouped together and are placed immediately following the source program. No reference to procedure-names in debug packets may be made in the body of the program.

DEBUG CARD

Each compile-time debugging packet is preceded by a DEBUG card.

Format

<u>DEBUG</u> location

The word DEBUG followed by location may appear anywhere within columns 1 through 72 on the card except in column 7 (because a D in column 7 specifies a debugging line). There must be no other text on the card. However, a sequence number followed by a space may appear in columns 1 through 7.

The location is the section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the packet is to be executed. Effectively, the statements in the packet are executed as though they were physically placed in the source program following the section-name or paragraph-name, but preceding the text associated with the procedure. The same location must not be used in more than one DEBUG control card. Location cannot be a paragraph-name within any debug packet.

A debug packet may consist of any procedural statements conforming to the requirements of American National Standard COBOL. The following considerations apply:

- A PERFORM or ALTER statement in a debug packet may refer to a procedure-name in any debug packet or in the main body of the Procedure Division.

- A GO TO statement in a debug packet may not refer to a procedure-name in another debug packet, but it may refer to a procedure-name in the main body of the Procedure Division.

SOURCE LISTING FORMAT CONTROL

The following statements control the spacing of the source program listings produced by the COBOL Compiler.

EJECT Statement

The EJECT statement specifies that the next source statement is to be printed at the top of the next page.

Format

<u>EJECT</u>

The word EJECT may be written anywhere within Area B and must be the only statement on the card. There must be no punctuation.

SKIP1/2/3 Statements

These statements specify lines to be skipped in the source listing.

Format

```
┌       ┐
│ SKIP1 │
│ SKIP2 │
│ SKIP3 │
└       ┘
```

SKIP1 specifies a single blank line (double spacing), SKIP2 specifies two blank lines (triple spacing), and SKIP3 specifies three blank lines (quadruple spacing).

SKIP1, SKIP2, or SKIP3 may be written anywhere within Area B and must be the only statement on the card. There must be no punctuation.

GLOBAL ITEMS

Quotation Mark Specification

✓

The double quotation mark (") conforms to the 1974 Standard, and is the installation option supplied with the compiler. If this option is chosen as the installation default, the single quotation mark (')—unless the default is overridden at compile time—must not be used, except as a character in a nonnumeric literal.

The single quotation mark (') is an IBM extension, and was the installation default supplied with the OS Full American National Standard COBOL compiler. If this option is chosen as the installation default, the double quotation mark (")—unless the default is overridden at compile time—must not be used, except as a character in a nonnumeric literal.

SECTION III—"OLD" LANGUAGE EXTENSIONS

The language extensions documented in this section require that the "old" compiler option be specified; that is, the language element in the 1974 Standard has a different interpretation from that in the 1968 Standard. Therefore, when these options are used, if 1968 Standard results are desired, the "old" compiler option must be coded.

Note: These extensions are included for compatibility purposes only. Therefore, they must be specified only in conjunction with 1968 Standard language or with IBM extensions to that Standard.

ENVIRONMENT DIVISION

SPECIAL-NAMES Paragraph

In the CURRENCY SIGN clause, the following additional characters may be specified: L /

SELECT OPTIONAL Clause

For files not necessarily present during each object program execution, OPTIONAL need not be specified; if specified, it is treated as documentation. The same function is performed by the system through the DUMMY or NULLFILE parameter on the DD statement.

RESERVE ALTERNATE AREAS Clause

Format

```
          ┌          ┐                ┌        ┐
          │ NO       │                │ AREA   │
RESERVE  <            >   ALTERNATE   │        │
          │ integer  │                │ AREAS  │
          └          ┘                └        ┘
```

This clause specifies that the number of buffers represented by integer is to be reserved for a physical sequential file or an indexed file that is accessed sequentially, in addition to the one required buffer which is automatically reserved.

This clause should not be specified for direct files; if specified, the clause is ignored and the one required buffer is reserved.

The number of additional buffers is represented by the value of integer. Integer must not exceed 254.

If NO is written, no additional buffers are reserved, aside from the standard minimum of one.

Note: When the optional word ALTERNATE is specified, the RESERVE clause is treated as old language, whether or not the "old" compiler option is specified.

DATA DIVISION

JUSTIFIED in a VALUE clause

Using JUSTIFIED in a VALUE clause, under the "old" language options, will cause initialization to be right justified.

NOT in Combined Relation Conditions

NOT is treated as a logical operator when only the subject is implied in an abbreviated combined relation condition. For example:

```
A = B AND NOT LESS THAN C OR D
```

is equivalent to

```
((A = B) AND (A NOT  <  C)) OR (A  <  D)
```

NOT is treated as part of the relational operator when both the subject and the relational operator are implied. For example:

```
A NOT > B AND C
```

is equivalent to

```
A NOT > B AND A NOT > C
```

MOVE Statement Scaling

If the sending field is a scaled integer (that is, the rightmost character in the PICTURE character-string is P), and if the receiving field is alphanumeric or numeric-edited, the trailing 0's are truncated.

PERFORM Statement in Segmented Programs

When a PERFORM statement in an independent segment refers to a procedure-name in a permanent segment, the independent segment is reinitialized upon each exit from the performed procedures.

UNSTRING Statement—DELIMITED BY ALL Option

When DELIMITED BY ALL is specified, two or more contiguous occurrences of any delimiter are treated as if they were only one occurrence; as much of the first occurrence as will fit is moved into the current delimiter receiving field (if specified); each additional occurrence is moved only if the complete occurrence will fit. Thus the current delimiter receiving field contains as many complete occurrences of the delimiter as are present or as the delimiter receiving field can hold, whichever is smaller.

SOURCE PROGRAM LIBRARY

COPY Statement

Figure 101 shows options of the COPY statement for which the "old" compiler option is specified. If option 1, 2, 3, or 4 is written, the entire entry is replaced by the information identified by library-name, except that data-name (if specified) replaces the corresponding data-name from the library.

Option 1 (within a File or Sort description entry, or within the Working-Storage Section or the Linkage Section):

```
01  data-name COPY statement.
```

Option 2 (with a Report Group):

```
01  [data-name]  COPY statement.
```

Option 3 (within the Working-Storage Section or the Linkage Section):

```
77  data-name COPY statement.
```

Option 4 (within the Working-Storage Section or the Linkage Section):

```
01  data-name-1 REDEFINES data-name-2 COPY statement.
77  data-name-1 REDEFINES data-name-2 COPY statement.
```

Figure 101. COPY Statement—"Old" Language Options

End of IBM Extension

COMMUNICATION FEATURE

RECEIVE MESSAGE Statement

When an end-of-segment indicator is present, and a RECEIVE MESSAGE statement is executed, the end-of-segment indicator is treated as a data character.

APPENDIX B. ASCII CONSIDERATIONS

The compiler supports the American National Standard Code for Information Interchange (ASCII). Thus the programmer can create and process tape files recorded in accordance with the following standards:

- American National Standard Code for Information Interchange, X3.4-1968
- American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1969
- American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI), X3.22-1967

ASCII encoded tape files, when read into the system, are automatically translated in the buffers into EBCDIC. Internal manipulation of data is performed exactly as if they were EBCDIC encoded files. For an output file, the system translates the EBCDIC characters into ASCII in the buffers before writing the file out on tape. Therefore, there are special considerations concerning ASCII encoded files when they are processed in COBOL. The following paragraphs discuss these considerations.

ENVIRONMENT DIVISION

In the Environment Division, the OBJECT-COMPUTER, SPECIAL-NAMES, and FILE-CONTROL paragraphs are affected.

OBJECT-COMPUTER and SPECIAL-NAMES Paragraphs

When at least one file in the program is an ASCII-encoded file, the alphabet-name clause of the SPECIAL-NAMES paragraph must be specified; the alphabet-name must be associated with STANDARD-1 (for ASCII collating sequence or code set).

When nonnumeric comparisons within the object program are to use the ASCII collating sequence, the PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph must be specified; the alphabet-name used must also be specified as an alphabet-name in the SPECIAL-NAMES paragraph, and associated with the STANDARD-1. For example:

```
OBJECT-COMPUTER.  IBM 370.
   PROGRAM COLLATING SEQUENCE IS ASCII-SEQUENCE.
SPECIAL-NAMES.  ASCII-SEQUENCE IS STANDARD.
```

When both clauses are specified, the ASCII collating sequence is used in this program to determine the truth value of the following nonnumeric comparisons:

- Those explicitly specified in relation conditions
- Those explicitly specified in condition-name conditions

IBM Extension

- Those implicitly specified in an RD entry CONTROL clause

End of IBM Extension

- Any nonnumeric sort or merge keys (unless the COLLATING SEQUENCE option is specified in the MERGE or SORT statement)

When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used for such comparisons.

FILE-CONTROL Paragraph

For ASCII files, the ASSIGN clause assignment-name has the following format:

```
[comments-] [S-]name
```

The file must be a physical sequential file assigned to a magnetic tape device. See the System Dependencies chapter.

I-O-CONTROL Paragraph

The assignment-name in a RERUN clause must not specify an ASCII-encoded file.

ASCII-encoded files containing checkpoint records cannot be processed.

DATA DIVISION

In the Data Division, there are special considerations for the FD entry and for data description entries.

FD Entry—CODE-SET Clause

The FD Entry for an ASCII-encoded file must contain a CODE-SET clause; the alphabet-name must be associated with STANDARD-1 (for the ASCII code set) in the SPECIAL-NAMES paragraph. For example:

```
SPECIAL-NAMES.  ASCII-SEQUENCE IS STANDARD-1
    .
    .
    .
FD  ASCII-FILE LABEL RECORDS STANDARD
  CODE-SET IS ASCII-SEQUENCE.
```

Data Description Entries

For ASCII files, the following data description considerations apply:

- PICTURE clause specifications for all five categories of data are valid.
- For signed numeric items, the SIGN clause with the SEPARATE CHARACTER option must be specified.
- For the USAGE clause, only the DISPLAY option is valid.

SORT/MERGE FEATURE

For ASCII-collated sort/merge operations, there are special considerations in the Environment Division, the Data Division, and in the SORT and MERGE statements.

Environment Division

If the PROGRAM COLLATING SEQUENCE clause specifies an ASCII collating sequence, and there is no COLLATING SEQUENCE option in the SORT or MERGE statement, any nonnumeric sort or merge keys are processed according to the ASCII collating sequence.

Checkpoint records for ASCII collated sort and merge operations are permitted. However, the assignment-name for the RERUN clause must not specify an ASCII-encoded file.

DATA DIVISION

There are special considerations for the FD entries for the input and output file, and for data description clauses.

The FD entries for the input and output files must specify the CODE-SET clause; alphabet-name must be associated with STANDARD-1 (ASCII code set) in the SPECIAL-NAMES paragraph.

In the data description entry, the considerations for the PICTURE, SIGN, and USAGE clauses are the same as for other ASCII-encoded files.

SORT and MERGE Statements

An ASCII collated sort/merge operation can be specified in two ways:

- Through the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph. (See the preceding section on the Environment Division.) In this case, all nonnumeric comparisons in the program are also performed using the ASCII collating sequence.
- Through the COLLATING SEQUENCE option of the SORT or MERGE statement; in this case, alphabet-name must be associated with STANDARD-1 (for ASCII collating sequence) in the SPECIAL-NAMES paragraph. In this case, only this sort/merge operation uses the ASCII collating sequence.

For this sort/merge operation, the COLLATING SEQUENCE option takes precedence over the PROGRAM COLLATING SEQUENCE clause.

If both the PROGRAM COLLATING SEQUENCE clause and the COLLATING SEQUENCE option are omitted (or if the one in effect specifies an EBCDIC collating sequence), the sort/merge is performed using the EBCDIC collating sequence.

APPENDIX C. SYSTEM/370 UNIT RECORD PROCESSING

This appendix describes special considerations for the following IBM System/370 unit record devices: 3505, 3525, and 3886.

3505/3525 CARD PROCESSING

The IBM 3505 Card Reader and the 3525 Card Punch are 80-column devices that offer more flexible processing capabilities than former card devices. The 3505 card reader can be used for sequential reading; it can also be used for Optical Mark Read (OMR) processing. Both the 3505 and the 3525 support Read Column Eliminate (RCE) processing. The 3525 card punch, when equipped with appropriate special features, can be used separately as a card reader, as a card punch, as an interpreting card punch, and as a printer (either 2-line or multiline printing is available); in addition, the read, punch, and print functions (any two or all three) can be combined, so that those functions specified are all performed during one pass of a card through the device.

The processing functions are all specified through new parameters of the DD statement. For OMR and RCE processing, format descriptor card(s) must also be included as the first card(s) of the data set. (For OMR processing, the format descriptor specifies those columns that are optically marked; for RCE processing, the format descriptor specifies those columns that are to be ignored.)

Note: The interpreting card punch is considered one function. It cannot be combined with the other functions, but is specified through the DD statement for the data set.

3505 OMR Processing

If the user wants to inspect the substitution character (hexadecimal "3F") placed in column 80 by the system for a defective optically marked card, a record description of 80 characters must be specified. (Note that the "3F" is placed in both card column 80 and the defective (unreadable) card column.)

3505/3525 RCE Processing

When RCE processing is specified for input, the user must not refer to the ignored columns (as specified by the format descriptor), or results are unpredictable.

When RCE processing is specified for output, any data in the COBOL record that corresponds to the ignored columns (as specified by the format descriptor) is not punched or printed.

RCE and OMR Format Descriptor

When the user specifies O (for Optical Mark Read) or R (for Read Column Eliminate) in the assignment-name then, at object time, a format descriptor must be provided as the first card(s) in the data deck. If the format descriptor is missing for such files, a message is issued to the operator, and the job is terminated.

The format descriptor must be the first card(s) in the data deck. Column 1 of the first card must be blank. The key word FORMAT must be punched in columns 2 through 7. Column 8 must be blank. Columns 9 through 71 can contain the parameters that specify which columns of the data cards are to be read in OMR or RCE mode. Continuation cards are valid. A continuation code must be placed in column 72 of the preceding card. Parameters may then be continued, beginning in column 16 of the continuation card. Comments, if used, must follow the last operand on each card by at least one blank space, and continuation card restrictions must be observed.

The format of the format descriptor is as follows:

```
Col.

12....7.9............
||    | |
||    | |
||    | |
VV    V V OUTPUT.
 FORMAT (N1,N2)[,(N3,N4)]...
```

N1, N2, N3, and N4 may be any decimal integers from 1 through 80. However, N2 must be greater than or equal to N1. N4 must be greater than or equal to N3. In addition, for OMR processing, N1 and N2 must be both even or both odd, N3 and N4 must be both even or both odd, and N3 - N2 must be greater than or equal to 2.

In OMR mode, the user establishes which columns are to be read in OMR mode. For example, if the user wants to read columns 1, 3, 5, 7, 9 and 70, 72, 74, 76, 78, 80 in OMR mode, the following descriptor format is valid:

```
FORMAT  (1,9),(70,80)
```

In RCE mode, the user specifies those columns that are <u>not</u> to be read. For example, if the user chooses to eliminate columns 20 through 30, and columns 52 through 73, the following format descriptor is valid:

```
FORMAT (20,30),(53,73)
```

ASSIGN CLAUSE SPECIFICATION

For the 3505 and 3525 devices, the ASSIGN clause assignment-name has the following form:

```
UR[-device]-organization-name
```

The optional device field can be specified to document whether this is a 3505 or 3525 device.

The organization field can be specified as follows:

O for 3505 OMR processing

R for 3525 RCE processing (including combined function files)

S for 3505 sequential reading (without OMR processing) and for 3525 combined function processing (without RCE processing)

The name field is the external name for this file; it must be the name specified in the DD statement for this file. It is a 1- to 8-character field; the first character must be alphabetic. It must conform to the rules for formation of a program-name.

<u>3525 COMBINED FUNCTION PROCESSING</u>

COBOL handles each of the separate functions to be combined as a separate logical file. Each such logical file has its own file structure and procedural processing requirements. However, because such combined function files refer to one physical unit, the user must observe certain restrictions during processing. The following sections explain the programming requirements for combined function processing.

The COBOL language does not define the files as being combined function files; instead, the combined functions are specified through parameters for the files' DD statements. (In this way, the user can process the same COBOL files as completely separate read, punch, and print files.) The necessary parameters are given in IBM System/360 Planning Guide for IBM 3505 Card Reader and IBM 3525 Card Punch on System/370.

ENVIRONMENT DIVISION CONSIDERATIONS

For each function, there must be a separate FILE-CONTROL entry written in the Environment Division. Each read function file and each punch function file must specify RESERVE 1 AREA.

SPECIAL-NAMES Paragraph

If stacker selection of punched output, or line control of printed output is desired, mnemonic-names for the purpose can be specified in the SPECIAL-NAMES Paragraph. The mnemonic-names may be equated with the function-names shown in Figure 102.

Function Name	Meaning
S01	Stacker 1
S02	Stacker 2
C01	Line 1
C02	Line 2
C03	Line 5
...	...
C12	Line 23

Figure 102. IBM 3525 Card Punch—Valid Function-Names

IBM Extension

When stacker selection is specified, RESERVE 1 AREA must also be specified.

End of IBM Extension

DATA DIVISION CONSIDERATIONS

For each logical file defined in the Environment Division for the combined function structure, there must be a corresponding FD entry and 01 record description entry in the File Section of the Data Division.

PROCEDURE DIVISION CONSIDERATIONS

Input/output operations must proceed in a specified order in the Procedure Division. In the 3525 device, the card passes first through the reading station, next through the punching station, and last through the printing station. Therefore, the combined functions may be specified only in the order shown in Figure 103.

Functions to be Combined	Order of Operations	Associated COBOL Statement
read/punch/print	read punch [print]	READ ... AT END WRITE ... ADVANCING WRITE ... AFTER ADVANCING
read/punch	read punch	READ ... AT END WRITE ... ADVANCING
read/print	read [print]	READ ... AT END WRITE ... AFTER ADVANCING
punch/print	punch [print]	WRITE ... ADVANCING WRITE ... AFTER ADVANCING

Figure 103. IBM 3525 Card Punch Combined Function Processing Order

All required operations on one card must be completed before the next card is obtained, or there is an abnormal termination of the job.

The following Procedure Division considerations in the COBOL source program apply.

OPEN Statement

For any specified function, an OPEN statement must be issued before the input/output operation for that function is attempted. The following additional considerations apply:

- For the read function, the file must be opened for INPUT.
- For the punch function and print function the file must be opened for OUTPUT.

READ Statement

For combined function files, the READ statement, if the function is specified, must be the first input/output operation specified. A second READ statement must not be issued before all necessary combined function operations for the same card have been completed, or abnormal termination of the job results.

WRITE Statement—Punch Function Files

When the punch function is used, the next input/output operation after the READ statement is issued must be a WRITE statement for the punch function file.

If the user wishes to punch additional data into some of the cards and not into others, a dummy WRITE statement must be issued for the null cards, first filling the output area with SPACES.

IBM Extension

If stacker selection for the punch function file is required, the user can specify the appropriate stacker function-names in the SPECIAL-NAMES paragraph, and then issue WRITE ADVANCING statements using the associated mnemonic-names.

End of IBM Extension

WRITE Statement—Print Function Files

After the punch function operations (if specified) are completed, the user can issue WRITE statement(s) for the print function file.

If the user wishes to print additional data on some of the data cards and not on others, the WRITE statement for the null cards may be omitted.

Any attempt to write beyond the limits of the card results in abnormal termination of the job; thus, the END-OF-PAGE option may not be specified.

Depending on the capabilities of the specific 3525 model in use, the print file may be either a 2-line print file or a multiline print file. Up to 64 characters may be printed on each line.

For a 2-line print file, the lines are printed on line 1 (top edge of card) and line 3 (between rows 11 and 12). Line control may not be specified. Automatic spacing is provided.

For a multiline print file, up to 25 lines of characters may be printed. Line control may be specified. If line control is not specified, automatic spacing is provided.

Line control is specified by issuing WRITE AFTER ADVANCING statements for the print function file. If line control is used for one such statement, it must be used for all other WRITE statements issued to the file. The maximum number of printable characters, including any space characters, is 64. Such WRITE statements must not specify space suppression.

Identifier and integer have the same meanings as they have for other WRITE AFTER ADVANCING statements. However, such WRITE statements must not increase the line position on the card beyond the card limits, or abnormal termination results.

The mnemonic-name option of the WRITE AFTER ADVANCING statement may also be specified. In the SPECIAL-NAMES paragraph, the function-names may be associated with the mnemonic-names as shown in Figure 104.

Function Name	Meaning
C02	Line 3
C03	Line 5
C04	Line 7
...	...
C12	Line 23

with WRITE AFTER ADVANCING Statement

Figure 104. IBM 3525 Card Punch Valid Function Names with WRITE AFTER ADVANCING Statement

See the System Dependencies chapter.

CLOSE Statement

When processing is completed, a CLOSE statement must be issued for each of the combined function files. After a CLOSE statement has been issued for any one of the functions, an attempt to perform processing for any of the functions results in abnormal termination.

3886 OCR PROCESSING

The IBM 3886 Optical Character Reader (OCR) Model 1 is a general purpose online unit record device that satisifies a broad range of data entry requirements. The 3886 OCR can significantly reduce time and cost factors, by eliminating input steps in both new and existing applications; a keying process is no longer necessary, since the 3886 OCR can read and recognize data created by numeric hand printing, high-speed computer printing, typewriters, and preprinted forms.

The IBM 3886 OCR uses several technologies which make it a compact, highly reliable, modular device. A powerful microprogrammed recognition and control processor performs all machine control and character recognition functions, and enables the 3886 OCR to perform sophisticated data and blank editing.

The 3886 OCR accepts documents from 3 x 3 to 9 x 12 inches in size. Under program control, it can read documents line-by-line, transmitting their contents line-by-line to the processor. Additional facilities, all under program control, include: document marking, line marking, document ejecting (with stacker selection), and line reading (of current line).

Support for the 3886 OCR is through an object-time subroutine in the COBOL library, invoked through COBOL CALL statements. By means of parameters passed to the subroutine, the following operations are provided: open and close the file, read a line, wait for read completion, mark a line, mark the current document, eject the current document, and load a format record. After each operation, a status indicator is passed back to the COBOL program, so that any exceptional condition can be tested.

Through a fixed format OCR file information area in the Working-Storage or Linkage-Section, the COBOL user defines storage for the OCR parameters. Of these parameters, the COBOL programmer is responsible for providing a file identifier, a format record identifier, an operation code, and (depending on the operation) a line number, line format number, mark code, and stacker number. After completion of each operation a status indicator is returned; after completion of a read operation, header and data records are also returned.

Two assembler-language macro instructions define OCR documents. The DFR macro instruction defines attributes common to a group of line types. The DLINT macro instruction defines specific attributes of an individual line type. The DFR and associated DLINT macro instructions are used in one assembly to build a format record module. The format record indicates the line types to be read, attributes of the fields in the lines, and the format of the data records to be processed.

The format record must be link-edited into a user library so that it can be loaded into the 3886 OCR when the file is to be processed.

Additional system information on the 3886 OCR can be found in OS/VS Program Planning Guide for IBM 3886 Optical Character Reader Model 1.

Information on the 3886 OCR device can be found in IBM 3886 Optical Character Reader General Information Manual and Input Document Design Guide and Specifications.

APPENDIX D. INTERMEDIATE RESULTS

This appendix describes the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results. The following abbreviations are used:

i is number of integer places carried for an intermediate result.

If the ROUNDED option is used, and if necessary for accuracy, 1 more integer may be added.

d is number of decimal places carried for an intermediate result.

dmax in a particular statement, the larger of either:

- The number of decimal places needed for the final result field, or
- The maximum number of decimal places defined for any operand, except

  exponents

  divisors

op1 is first operand in a generated arithmetic statement.

op2 is second operand in a generated arithmetic statement.

d1,d2 is number of decimal places defined for op1 or op2, respectively.

ir intermediate result field obtained from the execution of a generated arithmetic statement or operation. Ir1, ir2, and so forth, represent successive intermediate results. These intermediate results are generated either in registers or in storage locations. Successive intermediate results may have the same location.

In the case of an arithmetic statement containing only a single pair of operands, no intermediate results are generated, except when the TRUNC option is specified for COMPUTATIONAL items. Intermediate results are possible in the following cases:

- In an ADD or SUBTRACT statement containing multiple operands immediately following the verb
- In a COMPUTE statement specifying a series of arithmetic operations
- In arithmetic expressions contained in IF or PERFORM statements
- In the GIVING option with multiple result fields for the ADD, SUBTRACT, MULTIPLY, DIVIDE, or COMPUTE statements

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by

```
  **F             BY G          yielding ir1
  MULTIPLY B      BY C          yielding ir2
  DIVIDE E        INTO D        yielding ir3
  ADD A           TO ir2        yielding ir4
  SUBTRACT ir3    FROM ir4      yielding ir5
  ADD ir5         TO ir1        yielding Y
```

COMPILER CALCULATION OF INTERMEDIATE RESULTS

The number of integer places in an ir is calculated as follows:

- The compiler first determines the maximum value that the ir can contain by performing the statement in which the ir occurs.

  1. If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the PICTURE for the data-name (for example, PICTURE 9V99 has the value 9.99).

  2. If an operand is a literal, the literal's actual value is used.

  3. If an operand is an intermediate result, the value determined for the intermediate result in a previous arithmetic operation is used.

  4. If the operation is division:

     a. If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE for the data-name (for example, PICTURE 9V99 has the value 0.01).

     b. If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.

- When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

- The number of decimal places contained in an ir is calculated as:

Operation	**Decimal Places**
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	dmax
**	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

Note: The user must define the operands of any arithmetic statement with enough decimal places to give the desired accuracy in the final result.

Figure 105 indicates the action of the compiler when handling intermediate results.

Value of i + d	**Value of d**	**Value of i + dmax**	**Action Taken**
<30 =30	Any value	Any value	*i* integer and *d* decimal places are carried for ir
>30	<dmax	Any value	30 - *d* integer and *d* decimal places are carried for ir
	=dmax		
	>dmax	<30	*i* integer and 30 - *i* decimal places are carried for ir
		=30	
		>30	30 - dmax integer and dmax decimal places are carried for ir

Figure 105. Compiler Action on Intermediate Results

APPENDIX E. SAMPLE FILE-PROCESSING PROGRAMS

The programs in this appendix illustrate the fundamental programming techniques associated with each type of file organization. They are intended to illustrate the input/output statements necessary for certain access methods. Other COBOL features (the use of the ALTER statement and the PERFORM statement, for example) are used only incidentally. The programs are:

- Sequential File Creation
- Sequential File Updating and Extension
- Indexed File Creation
- Indexed File Updating
- Relative File Creation
- Relative File Updating
- Relative File Retrieval

SEQUENTIAL FILE CREATION

This program creates a sequential file of employee salary records. The input records are arranged in ascending order of employee number. The output file has the identical order.

Note: The program-id used by the system is CREATEOS.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CREATE-SEQUENTIAL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
SPECIAL-NAMES.  CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO SYS010-DISKETTE.
    SELECT OUTPUT-FILE ASSIGN TO SYS011-AS-MSSTORGE
         FILE STATUS IS SK.

DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE LABEL RECORD STANDARD.
01  INPUT-RECORD.
    05   INPUT-EMPLOYEE-NUMBER    PICTURE   9(6).
    05   INPUT-EMPLOYEE-NAME      PICTURE   X(28).
    05   INPUT-EMPLOYEE-CODE      PICTURE   9.
    05   INPUT-EMPLOYEE-SALARY    PICTURE   9(6)V99.
FD  OUTPUT-FILE    LABEL RECORDS  STANDARD.
01  OUTPUT-RECORD.
    05   OUTPUT-EMPLOYEE-NUMBER   PICTURE   9(6).
    05   OUTPUT-EMPLOYEE-NAME     PICTURE   X(28).
    05   OUTPUT-EMPLOYEE-CODE     PICTURE   9.
    05   OUTPUT-EMPLOYEE-SALARY   PICTURE   9(6)V99.
WORKING-STORAGE SECTION.
01  DISP-RECORD.
    05   OP-NAME    PICTURE X(5).
    05   FILLER     PICTURE XX VALUE SPACE.
    05   SK         PICTURE XX.
```

```
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT INPUT-FILE OUTPUT OUTPUT-FILE
         IF SK NOT = "00"
              MOVE "OPEN" TO OP-NAME
              PERFORM ERROR-OUT-1 THRU ERROR-OUT-2
              GO TO CLOSE-FILES-2.
BUILD-FILE.
    READ INPUT-FILE INTO OUTPUT-RECORD
        AT END GO TO CLOSE-FILES-1.
    WRITE OUTPUT-RECORD.
         IF SK NOT = "00"
         MOVE "WRITE" TO OP-NAME
         PERFORM ERROR-OUT-1 THRU ERROR-OUT-2.
    GO TO BUILD-FILE.
ERROR-OUT-1.
    DISPLAY DISP-RECORD UPON TYPEWRITER.
ERROR-OUT-2.
    EXIT.
CLOSE-FILES-1.
    CLOSE OUTPUT-FILE.
CLOSE-FILES-2.
    CLOSE INPUT-FILE.
    STOP RUN.
```

SEQUENTIAL FILE UPDATING AND EXTENSION

This program updates and extends the file created by the CREATE-SEQUENTIAL program. The INPUT-FILE and the I-O-FILE are each read. When a match is found between INPUT-EMPLOYEE-NUMBER and I-O-EMPLOYEE-NUMBER, the input record replaces the original mass storage record. After the I-O-FILE has been completely processed, new employee records are added at the end of the file.

Note: The program-id used by the system is UPDATEOS.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  UPDATE-SEQUENTIAL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-370.
OBJECT-COMPUTER.   IBM-370.
SPECIAL-NAMES.  CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO SYS010-UT-3350-S-DISKETTE.
    SELECT I-O-FILE   ASSIGN TO SYS011-AS-MSSTORG
      FILE STATUS IS SK.
DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE LABEL RECORD STANDARD.
01  INPUT-RECORD.
    05   INPUT-EMPLOYEE-NUMBER    PICTURE   9(6).
    05   INPUT-EMPLOYEE-NAME      PICTURE   X(28).
    05   INPUT-EMPLOYEE-CODE      PICTURE   9.
    05   INPUT-EMPLOYEE-SALARY    PICTURE   9(6)V99.
FD  I-O-FILE  LABEL RECORDS STANDARD.          MASTER
01  I-O-RECORD.
    05   I-O-EMPLOYEE-NUMBER      PICTURE   9(6).
    05   I-O-EMPLOYEE-NAME        PICTURE   X(28).
    05   I-O-EMPLOYEE-CODE        PICTURE   9.
    05   I-O-EMPLOYEE-SALARY      PICTURE   9(6)V99.
WORKING-STORAGE SECTION.
01  DISP-RECORD.
    05   OP-NAME   PICTURE   X(5).
    05   FILLER    PICTURE   XX VALUE SPACE.
    05   SK        PICTURE   XX VALUE "ZZ".
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT INPUT-FILE I-O I-O-FILE.
         IF SK NOT = "00"
              MOVE "OPEN" TO OP-NAME
              PERFORM ERROR-OUT-1 THRU ERROR-OUT-2
              GO TO CLOSE-FILES-2.
READ-INPUT.
    READ INPUT-FILE
         AT END GO TO CLOSE-FILES-1.
SWITCH-PARA.
    GO TO READ-I-O.
READ-I-O.
    READ I-O-FILE
         AT END ALTER SWITCH-PARA TO PROCEED TO ADD-ON-FILE
              CLOSE I-O-FILE OPEN EXTEND I-O-FILE
              GO TO READ-INPUT.
    IF INPUT-EMPLOYEE-NUMBER GREATER THAN I-O-EMPLOYEE-NUMBER
         GO TO READ-I-O.
    IF INPUT-EMPLOYEE-NUMBER = I-O-EMPLOYEE-NUMBER
         PERFORM I-O-REWRITE-1 THRU I-O-REWRITE-2
         GO TO READ-INPUT.
    IF INPUT-EMPLOYEE-NUMBER LESS THAN I-O-EMPLOYEE-NUMBER
         MOVE "INPUT ERR" TO DISP-RECORD
         PERFORM ERROR-OUT-1 THRU ERROR-OUT-2
         DISPLAY INPUT-RECORD
              UPON TYPEWRITER
         MOVE SPACES TO DISP-RECORD.
    GO TO READ-INPUT.
```

```
I-O-REWRITE-1.
    REWRITE I-O-RECORD FROM INPUT-RECORD.
         IF SK NOT = "00"
              MOVE "RWRTE" TO OP-NAME
              PERFORM ERROR-OUT-1 THRU ERROR-OUT-2.
I-O-REWRITE-2.
    EXIT.
ADD-ON-FILE.
    WRITE I-O-RECORD FROM INPUT-RECORD.
         IF SK NOT = "00"
              MOVE "WRITE" TO OP-NAME
              PERFORM ERROR-OUT-1 THRU ERROR-OUT-2.
    GO TO READ-INPUT.
ERROR-OUT-1.
    DISPLAY DISP-RECORD UPON TYPEWRITER.
ERROR-OUT-2.
    EXIT.
CLOSE-FILES-1.
    CLOSE I-O-FILE.
CLOSE-FILES-2.
    CLOSE INPUT-FILE.
    STOP RUN.
```

INDEXED FILE CREATION

This program creates an indexed file of summary records for bank depositors. The key within each indexed file record is REC-ID (the depositor's account number); the input records are ordered in ascending sequence upon this key. Records are read from the input file and transferred to the indexed file record area. The indexed file record is then written.

Note: The program-id used by the system is CREATEOI.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   CREATE-INDEXED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-370.
OBJECT-COMPUTER.   IBM-370.
SPECIAL-NAMES.     CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INDEXED-FILE ASSIGN TO SYS010-INDEXED
         ACCESS SEQUENTIAL ORGANIZATION INDEXED
         RECORD KEY IS REC-ID FILE STATUS IS SK.
    SELECT IN-FILE ASSIGN TO SYS011-DISKETTE.

DATA DIVISION.
FILE SECTION.
FD   INDEXED-FILE LABEL RECORDS STANDARD.
01   DISK-RECORD.
     05   REC-ID          PICTURE   X(10).
     05   DISK-FLD1       PICTURE   X(10).
     05   DISK-NAME       PICTURE   X(20).
     05   DISK-BAL        PICTURE   S99999V99.
FD   IN-FILE LABEL RECORDS STANDARD.
01   IN-RECORD.
     05   IN-KEY          PICTURE   X(10).
     05   IN-NAME         PICTURE   X(10).
     05   IN-BAL          PICTURE   S99999V99.
WORKING-STORAGE SECTION.
77  SK PIC XX VALUE "ZZ".
PROCEDURE DIVISION.
BEGIN-PROCESSING.
    OPEN INPUT IN-FILE OUTPUT INDEXED-FILE.
    IF SK NOT = "00"
         DISPLAY "OPEN FAILED " SK UPON TYPEWRITER
         STOP RUN.
PROCESS-PROCEDURE.
    READ IN-FILE AT END GO TO END-JOB.
    MOVE IN-KEY TO REC-ID.
    MOVE IN-NAME TO DISK-NAME
    MOVE IN-BAL TO DISK-BAL
    WRITE DISK-RECORD.
    IF SK NOT = "00"
         DISPLAY "WRITE FAILED " SK DISK-RECORD UPON TYPEWRITER.
    GO TO PROCESS-PROCEDURE.
END-JOB.
    CLOSE IN-FILE INDEXED-FILE.
    IF SK NOT = "00"
         DISPLAY "CLOSE FAILED " SK UPON TYPEWRITER.
    STOP RUN.
```

INDEXED FILE UPDATING

This program, using dynamic access, updates the indexed file created in the CREATE-INDEXED program.

The input records contain the key for the record (both generic and detailed), the depositor name, and the amount of the transaction.

When the input record is read, the program tests whether this is a transaction record (in which case all fields of the record are filled), or a record requesting sequential retrieval of a specific generic class (in which case only the IN-GEN-FLD of the input record contains data).

Random access is used for the updating and printing of the transaction records. Sequential access is used for the retrieval and printing of all records within one generic class.

Note: The program-id used by the system is UPDATEOI.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   UPDATE-INDEXED.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-370.
OBJECT-COMPUTER.   IBM-370.
SPECIAL-NAMES.     CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INDEXED-FILE ASSIGN TO SYS010-INDEXD
         ACCESS DYNAMIC ORGANIZATION INDEXED
         RECORD KEY REC-ID FILE STATUS SK.
    SELECT IN-FILE ASSIGN TO SYS011-DISKETTE.
    SELECT PRINT-FILE ASSIGN TO SYS012-PRINTER.
DATA DIVISION.
FILE SECTION.
FD  INDEXED-FILE LABEL RECORDS STANDARD.
01  DISK-RECORD.
    05   REC-ID.
    10   REC-GEN-FLD    PICTURE X(5).
    10   REC-DET-FLD    PICTURE X(5).
    05   DISK-FLD1      PICTURE X(10).
    05   DISK-NAME      PICTURE X(20).
    05   DISK-BAL       PICTURE S9(5)V99.
FD  IN-FILE   LABEL RECORD STANDARD.
01  IN-REC.
    05   IN-ID.
    10   IN-GEN-FLD     PICTURE X(5).
    10   IN-DET-FLD     PICTURE X(5).
    05   IN-NAME        PICTURE X(20).
    05   IN-AMT         PICTURE S9(5)V99.
FD  PRINT-FILE     LABEL RECORDS OMITTED
    LINAGE 62 FOOTING 59.
01  PRINT-RECORD-1.
    05   PRINT-ID       PICTURE X(10).
    05   FILLER         PICTURE X(5).
    05   PRINT-NAME     PICTURE X(20).
    05   FILLER         PICTURE X(5).
    05   PRINT-BAL      PICTURE $,.99-.
    05   FILLER         PICTURE X(5).
    05   PRINT-AMT      PICTURE $,.99-.
    05   FILLER         PICTURE X(5).
    05   PRINT-NEW-BAL  PICTURE $,.99-.
01  PRINT-RECORD-2      PICTURE X(89).
WORKING-STORAGE SECTION.
77  GO-TO-SWITCH        PICTURE 9 VALUE 1.
01  PAGE-HEAD.
    05   FILLER         PICTURE X(38)  VALUE SPACES.
    05   FILLER         PICTURE X(13)  VALUE "UPDATE REPORT".
    05   FILLER         PICTURE X(38)  VALUE SPACES.
```

```
01  PAGE-FOOT.
    05   FILLER         PICTURE X(81)  VALUE SPACES.
    05   FILLER         PICTURE A(6)   VALUE "PAGE".
    05   PG-NUMBER      PICTURE 99     VALUE 00.
01  ERROR-MESSAGE.
    05   OP-NAME        PICTURE X(7).
    05   FILLER         PICTURE XX     VALUE SPACES.
    05   SK             PICTURE XX     VALUE "ZZ".
PROCEDURE DIVISION.
BEGIN-PROCESSING.
    OPEN INPUT IN-FILE
         I-O INDEXED-FILE
         OUTPUT PRINT-FILE.
    IF SK NOT = "00" MOVE "OPEN" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO END-JOB-2.
    PERFORM PAGE-START.
READ-INPUT.
    MOVE SPACES TO IN-REC.
    READ IN-FILE AT END GO TO END-JOB-1.
    IF IN-DET-FLD = SPACES
         GO TO SEQ-PROCESS-2.
DYNAM-PROCESS-1.
    MOVE IN-ID TO REC-ID.
    READ INDEXED-FILE.
    IF SK NOT = "00" MOVE "READ-D" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO READ-INPUT.
    MOVE DISK-NAME TO PRINT-NAME.
    MOVE DISK-BAL TO PRINT-BAL.
    ADD IN-AMT TO DISK-BAL.
    MOVE DISK-BAL TO PRINT-NEW-BAL.
    PERFORM WRITE-PARA-1 THRU WRITE-PARA-2.
DYNAM-PROCESS-2.
    GO TO READ-INPUT.
SEQ-PROCESS-1.
    MOVE IN-GEN-FLD TO REC-GEN-FLD.
    START INDEXED-FILE
         KEY = REC-GEN-FLD.
    IF SK NOT = "00" MOVE "START_" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO READ-INPUT.
SEQ-PROCESS-2.
    READ INDEXED-FILE NEXT.
    IF SK NOT = "00" MOVE "READ-S" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO READ-INPUT.
    IF REC-GEN-FLD GREATER THAN IN-GEN-FLD
         GO TO READ-INPUT.
    MOVE DISK-NAME TO PRINT-NAME.
    MOVE DISK-BAL TO PRINT-BAL PRINT-NEW-BAL.
    PERFORM WRITE-PARA-1 THRU WRITE-PARA-2.
SEQ-PROCESS-3.
    GO TO SEQ-PROCESS-1.
WRITE-PARA-1.
    WRITE PRINT-RECORD-1
         AT END-OF-PAGE
         PERFORM PAGE-END THROUGH PAGE-START.
    REWRITE DISK-RECORD.
    IF SK NOT = "00" MOVE "REWRITE" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
WRITE-PARA-2.
    EXIT.
PAGE-END.
    ADD 1 TO PG-NUMBER.
    WRITE PRINT-RECORD-2 FROM PAGE-FOOT
         AFTER ADVANCING 3.
PAGE-START.
    WRITE PRINT-RECORD-2 FROM PAGE-HEAD
         AFTER ADVANCING PAGE.
```

```
ERROR-ROUTINE-1.
    DISPLAY ERROR-MESSAGE UPON TYPEWRITER.
ERROR-ROUTINE-2.
    EXIT.
END-JOB-1.
    CLOSE INDEXED-FILE.
    IF SK NOT = "00" MOVE "CLOSE" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
END-JOB-2.
    CLOSE IN-FILE PRINT-FILE.
    STOP RUN.
```

RELATIVE FILE CREATION

This program creates a relative file of summary sales records using sequential access. Each record contains a 5-year summary of unit and dollar sales for one week of the year; there are 52 records within the file, each representing 1 week.

Each input record represents the summary sales for one week of one year. The records for the first week of the last five years (in ascending order) are the first five input records. The records for the second week of the last five years are the next five input records, and so forth. Thus, five input records fill one output record.

Because it is not needed for sequential access unless the START statement is used, the RELATIVE KEY for the RELATIVE-FILE is not specified. (For updating, however, the key would be INPUT-WEEK.)

Note: The program-id used by the system is CREATEOR.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CREATE-RELATIVE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-370.
OBJECT-COMPUTER.   IBM-370.
SPECIAL-NAMES.     CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RELATIVE-FILE ASSIGN TO SYS010-RELATIVE
         ACCESS SEQUENTIAL ORGANIZATION RELATIVE
         FILE STATUS SK.
    SELECT INPUT-FILE ASSIGN TO SYS011-AS-SEQUENTIAL
         FILE STATUS SK1.
DATA DIVISION.
FILE SECTION.
FD  RELATIVE-FILE LABEL RECORDS ARE STANDARD.
01  RELATIVE-RECORD-01.
    05   RELATIVE-RECORD OCCURS 5.
         10   RELATIVE-YEAR            PICTURE 99.
         10   RELATIVE-WEEK            PICTURE 99.
         10   RELATIVE-UNIT-SALES      PICTURE S9(6).
         10   RELATIVE-DOLLAR-SALES    PICTURE S9(9)V99.
FD  INPUT-FILE LABEL RECORDS STANDARD.
01  INPUT-RECORD.
    05   INPUT-YEAR          PICTURE 99.
    05   INPUT-WEEK          PICTURE 99.
    05   INPUT-UNIT-SALES    PICTURE S9(6).
    05   INPUT-DOLLAR-SALES  PICTURE S9(9)V99.
WORKING-STORAGE SECTION.
77  OCCURS-CONTROL           PICTURE 9 VALUE 0.
01  WORK-RECORD.
    05   WORK-YEAR           PICTURE 99 VALUE 00.
    05   WORK-WEEK           PICTURE 99.
    05   WORK-UNIT-SALES     PICTURE S9(6).
    05   WORK-DOLLAR-SALES   PICTURE S9(9)V99.
01  ERROR-MESSAGE.
    05   OP-NAME             PICTURE X(5).
    05   FILLER              PICTURE XX VALUE SPACE.
    05   SK                  PICTURE XX VALUE "ZZ".
    05   SK1                 PICTURE XX VALUE "ZZ".
PROCEDURE DIVISION.
BEGIN-PROCESSING.
    OPEN INPUT INPUT-FILE    OUTPUT RELATIVE-FILE.
    IF SK NOT = "00" OR SK1 NOT = "00"
         MOVE "OPEN" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THROUGH ERROR-ROUTINE-2
         STOP RUN.
```

```
READ-FILE.
    READ INPUT-FILE AT END GO TO END-ROUTINE.
    IF SK1 NOT = "00"
         MOVE "READ " TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO READ-FILE.
PROCESS-RECORD.
    IF INPUT-YEAR GREATER THAN WORK-YEAR
         ADD 1 TO OCCURS-CONTROL.
    IF INPUT-YEAR LESS THAN WORK-YEAR
         PERFORM WRITE-1 THRU WRITE-2
         SUBTRACT 4 FROM OCCURS-CONTROL.
    MOVE INPUT-RECORD TO WORK-RECORD
    MOVE WORK-RECORD TO RELATIVE-RECORD (OCCURS-CONTROL).
    GO TO READ-FILE.
WRITE-1.
    WRITE RELATIVE-RECORD-01.
    IF SK NOT = "00" MOVE "WRITE" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
WRITE-2.
    EXIT.
ERROR-ROUTINE-1.
    DISPLAY ERROR-MESSAGE UPON TYPEWRITER.
ERROR-ROUTINE-2.
    EXIT.
END-ROUTINE.
    CLOSE RELATIVE-FILE INPUT-FILE.
    IF SK NOT = "00" OR SK1 NOT = "00"
         MOVE "CLOSE" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
    STOP RUN.
```

RELATIVE FILE UPDATING

This program updates the file of summary sales records created in the CREATE-RELATIVE program, using sequential access. The updating program adds a record for the new year, and deletes the oldest year's records from the RELATIVE-FILE.

The input record represents the summary sales record for one week of the last preceding year. The RELATIVE KEY for the RELATIVE-FILE is present in the input record as INPUT-WEEK. The RELATIVE KEY is used to check that the record was correctly written.

Note: The program-id used by the system is UPDATE0R.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   UPDATE-RELATIVE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-370.
OBJECT-COMPUTER.   IBM-370.
SPECIAL-NAMES.     CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RELATIVE-FILE ASSIGN TO SYS010-RELATIVE
    ACCESS SEQUENTIAL ORGANIZATION RELATIVE
    RELATIVE KEY INPUT-WEEK FILE STATUS SK.
    SELECT INPUT-FILE ASSIGN TO SYS011-AS-SEQUENTIAL
    FILE STATUS SK1.
DATA DIVISION.
FILE SECTION.
FD  RELATIVE-FILE  LABEL RECORDS STANDARD.
01  RELATIVE-RECORD             PICTURE X(105).
FD  INPUT-FILE LABEL RECORDS STANDARD.
01  INPUT-RECORD.
    05   INPUT-YEAR             PICTURE 99.
    05   INPUT-WEEK             PICTURE 99.
    05   INPUT-UNIT-SALES       PICTURE S9(6).
    05   INPUT-DOLLAR-SALES     PICTURE S9(9)V99.
WORKING-STORAGE SECTION.
01  WORK-RECORD.
    05   FILLER                 PICTURE X(21).
    05   CURRENT-WORK-YEARS     PICTURE X(84).
    05   NEW-WORK-YEAR.
    10   WORK-YEAR              PICTURE 99.
    10   WORK-WEEK              PICTURE 99.
    10   WORK-UNIT-SALES        PICTURE S9(6).
    10   WORK-DOLLAR-SALES      PICTURE S9(9)V99.
66  WORK-OUT-RECORD RENAMES
    CURRENT-WORK-YEARS THRU NEW-WORK-YEAR.
01  ERROR-MESSAGE.
    05   OP-NAME                PICTURE X(7).
    05   FILLER                 PICTURE XX VALUE SPACES.
    05   SK                     PICTURE XX VALUE "ZZ".
    05   SK1                    PICTURE XX VALUE "ZZ".
PROCEDURE DIVISION.
BEGIN-PROCESSING.
    OPEN INPUT INPUT-FILE       I-O RELATIVE-FILE.
    IF SK NOT = "00" OR SK1 NOT = "00"
         MOVE "OPEN" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         STOP RUN.
READ-FILES.
    READ INPUT-FILE INTO NEW-WORK-YEAR
       AT END GO TO END-PROCESSING.
    READ RELATIVE-FILE INTO WORK-RECORD
       AT END GO TO END-PROCESSING.
```

```
*RELATIVE KEY INPUT-WEEK WAS SPECIFIED.  THEREFORE
*    INPUT-WEEK NOW CONTAINS THE RELATIVE RECORD
*    NUMBER OF THE RECORD JUST RETRIEVED FROM
*    RELATIVE-FILE.
     IF SK NOT = "00" MOVE "READ-R" TO OP-NAME
          PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
          GO TO READ-FILES.
     IF SK1 NOT = "00" MOVE "READ-I" TO OP-NAME
          PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
          GO TO READ-FILES.
     IF INPUT-WEEK NOT = WORK-WEEK
          DISPLAY "INVALID RECORD MATCH"
               INPUT-WEEK WORK-WEEK
              UPON TYPEWRITER
              GO TO READ-FILES.
WRITE-RELATIVE.
     REWRITE RELATIVE-RECORD FROM WORK-OUT-RECORD.
     IF SK NOT = "00" MOVE "REWRITE" TO OP-NAME
          PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2.
     GO TO READ-FILES.
ERROR-ROUTINE-1.
     DISPLAY ERROR-MESSAGE UPON TYPEWRITER.
ERROR-ROUTINE-2.
     EXIT.
END-PROCESSING.
     CLOSE RELATIVE-FILE INPUT-FILE
     STOP RUN.
```

RELATIVE FILE RETRIEVAL

This program, using dynamic access, retrieves the summary file created by the CREATE-RELATIVE program.

The records of the INPUT-FILE contain one required field (INPUT-WEEK), which is the RELATIVE KEY for RELATIVE-FILE, and one optional field (END-WEEK).

An input record containing data in INPUT-WEEK and spaces in END-WEEK requests a printout for that one specific RELATIVE-RECORD; the record is retrieved through random access.

An input record containing data in both INPUT-WEEK and END-WEEK requests a printout of all the RELATIVE-FILE records within the RELATIVE KEY range of INPUT-WEEK through END-WEEK, inclusive; these records are retrieved through sequential access.

Note: The program-id used by the system is RETRVE0R.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   RETRVE-RELATIVE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-370.
OBJECT-COMPUTER.   IBM-370.
SPECIAL-NAMES.     CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RELATIVE-FILE ASSIGN TO SYS010-RELATIVE
         ACCESS DYNAMIC ORGANIZATION RELATIVE
         RELATIVE KEY INPUT-WEEK
         FILE STATUS SK.
    SELECT INPUT-FILE ASSIGN TO SYS011-CARDS.
    SELECT PRINT-FILE ASSIGN TO SYS012-PRINTER.
DATA DIVISION.
FILE SECTION.
FD  RELATIVE-FILE LABEL RECORDS ARE STANDARD.
01  RELATIVE-RECORD-01.
    05   RELATIVE-RECORD OCCURS 5.
         10   RELATIVE-YEAR                 PICTURE 99.
         10   RELATIVE-WEEK                 PICTURE 99.
         10   RELATIVE-UNIT-SALES           PICTURE S9(6).
         10   RELATIVE-DOLLAR-SALES         PICTURE S9(9)V99.
FD  INPUT-FILE LABEL RECORDS OMITTED.
01  INPUT-RECORD.
    05   INPUT-WEEK                    PICTURE 99.
    05   END-WEEK                    PICTURE 99 BLANK WHEN ZERO.
FD  PRINT-FILE LABEL RECORDS OMITTED.
01  PRINT-RECORD.
    05   PRINT-WEEK                    PICTURE 99.
    05   FILLER                        PICTURE X(5).
    05   PRINT-YEAR                    PICTURE 99.
    05   FILLER                        PICTURE X(5).
    05   PRINT-UNIT-SALES              PICTURE ZZZ,ZZ9.
    05   FILLER                        PICTURE X(5).
    05   PRINT-DOLLAR-SALES            PICTURE $,,.99.
WORKING-STORAGE SECTION.
77  OCCURS-CONTROL                     PICTURE 9 VALUE 0.
01  ERROR-MESSAGE.
    05   OP-NAME                       PICTURE X(5).
    05   FILLER                        PICTURE XX VALUE SPACES.
    05   SK                            PICTURE XX VALUE "ZZ".
PROCEDURE DIVISION.
BEGIN-PROCESSING.
    OPEN INPUT INPUT-FILE RELATIVE-FILE
         OUTPUT PRINT-FILE.
    IF SK NOT = "00" MOVE "OPEN" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO END-ROUTINE-2.
    MOVE SPACES TO PRINT-RECORD.
```

```
READ-INPUT-FILE.
    READ INPUT-FILE AT END GO TO END-ROUTINE-1.
    IF END-WEEK NOT = SPACES
         GO TO SEQUENTIAL-PROCESS.
RANDOM-PROCESS.
    READ RELATIVE-FILE
    PERFORM WRITE-PRINT-1 THRU WRITE-PRINT-3.
    GO TO READ-INPUT-FILE.
SEQUENTIAL-PROCESS.
    START RELATIVE-FILE.
    IF SK NOT = "00" MOVE "START" TO OP-NAME
         PERFORM ERROR-ROUTINE-1 THRU ERROR-ROUTINE-2
         GO TO READ-INPUT-FILE.
    PERFORM READ-REL-SEQ-1 THRU READ-REL-SEQ-3
         UNTIL PRINT-WEEK = END-WEEK.
    GO TO READ-INPUT-FILE.
READ-REL-SEQ-1.
    READ RELATIVE-FILE NEXT AT END GO TO END-ROUTINE-1.
READ-REL-SEQ-2.
    PERFORM WRITE-PRINT-1 THRU WRITE-PRINT-3.
READ-REL-SEQ-3.
    EXIT.
WRITE-PRINT-1.
    ADD 1 TO OCCURS-CONTROL.
    WRITE PRINT-RECORD FROM RELATIVE-RECORD (OCCURS-CONTROL)
         AFTER ADVANCING 2.
    IF OCCURS-CONTROL = 5 MOVE 0 TO OCCURS-CONTROL
         GO TO WRITE-PRINT-3.
WRITE-PRINT-2.
    GO TO WRITE-PRINT-1.
WRITE-PRINT-3.
    EXIT.
ERROR-ROUTINE-1.
    DISPLAY ERROR-MESSAGE UPON TYPEWRITER.
ERROR-ROUTINE-2.
    EXIT.
END-ROUTINE-1.
    CLOSE RELATIVE-FILE.
END-ROUTINE-2.
    CLOSE INPUT-FILE PRINT-FILE.
    STOP RUN.
```

APPENDIX F. OS/VS COBOL RESERVED WORD LIST

The following list identifies all reserved words in:

- American National Standard, 3.23-1974
- CODASYL COBOL (from CODASYL COBOL Journal of Development, dated March 1980) but not in OS/VS COBOL.
- OS/VS COBOL

		CODASYL	
	ANS	COBOL	OS/VS
Reserved Word	COBOL	only	COBOL
ACCEPT	X	-	X
ACCESS	X	-	X
ACTUAL	-	-	X
ADD	X	-	X
ADVANCING	X	-	X
AFTER	X	-	X
ALL	X	-	X
ALPHABET	-	X	-
ALPHABETIC	X	-	X
ALPHANUMERIC	-	X	-
ALPHANUMERIC-EDITED	-	X	-
ALSO	X	-	X
ALTER	X	-	X
ALTERNATE	X	-	X
AND	X	-	X
ANY	-	X	-
APPLY	-	-	X
ARE	X	-	X
AREA	X	-	X
AREAS	X	-	X
ASCENDING	X	-	X
ASSIGN	X	-	X
AT	X	-	X
AUTHOR	X	-	X
BASIS	-	-	X
BEFORE	X	-	X
BEGINNING	-	-	X
BINARY	-	X	-
BIT	-	X	-
BITS	-	X	-
BLANK	X	-	X
BLOCK	X	-	X
BOOLEAN	-	X	-
BOTTOM	X	-	X
BY	X	-	X
CALL	X	-	X
CANCEL	X	-	X
CBL	-	-	X
CD	X	-	X
CF	X	-	X

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
CH	X	-	X
CHANGED	-	-	X
CHARACTER	X	-	X
CHARACTERS	X	-	X
CLOCK-UNITS	X	-	-
CLOSE	X	-	X
COBOL	X	-	-
CODE	X	-	X
CODE-SET	X	-	X
COLLATING	X	-	X
COLUMN	X	-	X
COMMA	X	-	X
COMMIT	-	X	-
COMMON	-	X	-
COMMUNICATION	X	-	X
COMP	X	-	X
COMP-1	-	-	X
COMP-2	-	-	X
COMP-3	-	-	X
COMP-4	-	-	X
COMPUTATIONAL	X	-	X
COMPUTATIONAL-1	-	-	X
COMPUTATIONAL-2	-	-	X
COMPUTATIONAL-3	-	-	X
COMPUTATIONAL-4	-	-	X
COMPUTE	X	-	X
CONFIGURATION	X	-	X
CONNECT	-	X	-
CONSOLE	-	-	X
CONTAINS	X	-	X
CONTENT	-	X	-
CONTINUE	-	X	-
CONTROL	X	-	X
CONTROLS	X	-	X
CONVERTING	-	X	-
COPY	X	-	X
CORE-INDEX	-	-	X
CORR	X	-	X
CORRESPONDING	X	-	X
COUNT	X	-	X
CSP	-	-	X
CURRENCY	X	-	X
CURRENT	-	X	-
CURRENT-DATE	-	-	X
C01	-	-	X
C02	-	-	X
C03	-	-	X
C04	-	-	X
C05	-	-	X
C06	-	-	X
C07	-	-	X
C08	-	-	X
C09	-	-	X
C10	-	-	X
C11	-	-	X

		CODASYL	
	ANS	COBOL	OS/VS
Reserved Word	COBOL	only	COBOL
C12	-	-	X
DATA	X	-	X
DATE	X	-	X
DATE-COMPILED	X	-	X
DATE-WRITTEN	X	-	X
DAY	X	-	X
DAY-OF-WEEK	-	X	-
DB	-	X	-
DB-ACCESS-CONTROL-KEY	-	X	-
DB-DATA-NAME	-	X	-
DB-EXCEPTION	-	X	-
DB-RECORD-NAME	-	X	-
DB-SET-NAME	-	X	-
DB-STATUS	-	X	-
DE	X	-	X
DEBUG	-	-	X
DEBUG-CONTENTS	X	-	X
DEBUG-ITEM	X	-	X
DEBUG-LINE	X	-	X
DEBUG-NAME	X	-	X
DEBUG-SUB-1	X	-	X
DEBUG-SUB-2	X	-	X
DEBUG-SUB-3	X	-	X
DEBUGGING	X	-	X
DECIMAL-POINT	X	-	X
DECLARATIVES	X	-	X
DELETE	X	-	X
DELIMITED	X	-	X
DELIMITER	X	-	X
DEPENDING	X	-	X
DESCENDING	X	-	X
DESTINATION	X	-	X
DETAIL	X	-	X
DISABLE	X	-	X
DISCONNECT	-	X	-
DIS	-	-	X
DISPLAY	X	-	X
DISPLAY-n	-	X	-
DISPLAY-ST	-	-	X
DIVIDE	X	-	X
DIVISION	X	-	X
DOWN	X	-	X
DUPLICATE	-	X	-
DUPLICATES	X	-	X
DYNAMIC	X	-	X
EGI	X	-	X
EJECT	-	-	X
ELSE	X	-	X
EMI	X	-	X
EMPTY	-	X	-
ENABLE	X	-	X
END	X	-	X
END-ADD	-	X	-
END-CALL	-	X	-
END-COMPUTE	-	X	-

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
END-DELETE	-	X	-
END-DIVIDE	-	X	-
END-EVALUATE	-	X	-
END-IF	-	X	-
END-MULTIPLY	-	X	-
END-OF-PAGE	X	-	X
END-PERFORM	-	X	-
END-READ	-	X	-
END-RECEIVE	-	X	-
END-RETURN	-	X	-
END-REWRITE	-	X	-
END-SEARCH	-	X	-
END-START	-	X	-
END-STRING	-	X	-
END-SUBTRACT	-	X	-
END-UNSTRING	-	X	-
END-WRITE	-	X	-
ENDING	-	-	X
ENTER	X	-	X
ENTRY	-	-	X
ENVIRONMENT	X	-	X
EOP	X	-	X
EQUAL	X	-	X
EQUALS	-	X	-
ERASE	-	X	-
ERROR	X	-	X
ESI	X	-	X
EVALUATE	-	X	-
EVERY	X	-	X
EXAMINE	-	-	X
EXCEEDS	-	X	-
EXCEPTION	X	-	X
EXCLUSIVE	-	X	-
EXHIBIT	-	-	X
EXIT	X	-	X
EXOR	-	X	-
EXTEND	X	-	X
EXTERNAL	-	X	-
FALSE	-	X	-
FD	X	-	X
FILE	X	-	X
FILE-CONTROL	X	-	X
FILE-LIMIT	-	-	X
FILE-LIMITS	-	-	X
FILLER	X	-	X
FINAL	X	-	X
FIND	-	X	-
FINISH	-	X	-
FIRST	X	-	X
FOOTING	X	-	X
FOR	X	-	X
FREE	-	X	-
FROM	X	-	X
FUNCTION	-	X	-
GENERATE	X	-	X

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
GET	-	X	-
GIVING	X	-	X
GLOBAL	-	X	-
GO	X	-	X
GREATER	X	-	X
GROUP	X	-	X
HEADING	X	-	X
HIGH-VALUE	X	-	X
HIGH-VALUES	X	-	X
I-O	X	-	X
I-O-CONTROL	X	-	X
ID	-	-	X
IDENTIFICATION	X	-	X
IF	X	-	X
IN	X	-	X
INDEX	X	-	X
INDEX-n	-	X	-
INDEXED	X	-	X
INDICATE	X	-	X
INITIAL	X	-	X
INITIALIZE	-	X	X
INITIATE	X	-	X
INPUT	X	-	X
INPUT-OUTPUT	X	-	X
INSERT	-	-	X
INSPECT	X	-	X
INSTALLATION	X	-	X
INTO	X	-	X
INVALID	X	-	X
IS	X	-	X
JUST	X	-	X
JUSTIFIED	X	-	X
KEEP	-	X	-
KEY	X	-	X
LABEL	X	-	X
LAST	X	-	X
LD	-	X	-
LEADING	X	-	X
LEAVE	-	-	X
LEFT	X	-	X
LENGTH	X	-	X
LESS	X	-	X
LIMIT	X	-	X
LIMITS	X	-	X
LINAGE	X	-	X
LINAGE-COUNTER	X	-	X
LINE	X	-	X
LINE-COUNTER	X	-	X
LINES	X	-	X
LINKAGE	X	-	X
LOCALLY	-	X	-
LOCK	X	-	X
LOW-VALUE	X	-	X
LOW-VALUES	X	-	X
MEMBER	-	X	-

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
MEMORY	X	-	X
MERGE	X	-	X
MESSAGE	X	-	X
MODE	X	-	X
MODIFY	-	X	-
MODULES	X	-	X
MORE-LABELS	-	-	X
MOVE	X	-	X
MULTIPLE	X	-	X
MULTIPLY	X	-	X
NAMED	-	-	X
NATIVE	X	-	X
NEGATIVE	X	-	X
NEXT	X	-	X
NO	X	-	X
NOMINAL	-	X	X
NOT	X	-	X
NOTE	-	-	X
NULL	-	X	-
NUMBER	X	-	X
NUMERIC	X	-	X
NUMERIC-EDITED	-	X	-
OBJECT-COMPUTER	X	-	X
OCCURS	X	-	X
OF	X	-	X
OFF	X	-	X
OMITTED	X	-	X
ON	X	-	X
OPEN	X	-	X
OPTIONAL	X	-	X
OR	X	-	X
ORDER	-	X	-
ORGANIZATION	X	-	X
OTHER	-	X	-
OTHERWISE	-	-	X
OUTPUT	X	-	X
OVERFLOW	X	-	X
OWNER	-	X	-
PACKED-DECIMAL	-	X	-
PADDING	-	X	-
PAGE	X	-	X
PAGE-COUNTER	X	-	X
PASSWORD	-	-	X
PERFORM	X	-	X
PF	X	-	X
PH	X	-	X
PIC	X	-	X
PICTURE	X	-	X
PLUS	X	-	X
POINTER	X	-	X
POSITION	X	-	X
POSITIONING	-	-	X
POSITIVE	X	-	X
PRINTING	X	-	-
PRIOR	-	X	-

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
PROCEDURE	X	-	X
PROCEDURES	X	-	X
PROCEED	X	-	X
PROCESSING	-	-	X
PROGRAM	X	-	X
PROGRAM-ID	X	-	X
PROTECTED	-	X	-
PURGE	-	X	-
QUEUE	X	-	X
QUOTE	X	-	X
QUOTES	X	-	X
RANDOM	X	-	X
RD	X	-	X
READ	X	-	X
READY	-	-	X
REALM	-	X	-
REALMS	-	X	-
RECEIVE	X	-	X
RECONNECT	-	X	-
RECORD	X	-	X
RECORD-NAME	-	X	-
RECORD-OVERFLOW	-	-	X
RECORDS	X	-	X
REDEFINES	X	-	X
REEL	X	-	X
REFERENCE	-	X	-
REFERENCES	X	-	X
RELATIVE	X	-	X
RELEASE	X	-	X
RELOAD	-	-	X
REMAINDER	X	-	X
REMARKS	-	-	X
REMOVAL	X	-	X
RENAMES	X	-	X
REORG-CRITERIA	-	-	X
REPEATED	-	X	-
REPLACE	-	X	-
REPLACING	X	-	X
REPORT	X	-	X
REPORTING	X	-	X
REPORTS	X	-	X
REREAD	-	-	X
RERUN	X	-	X
RESERVE	X	-	X
RESET	X	-	X
RETAINING	-	X	-
RETRIEVAL	-	X	-
RETURN	X	-	X
RETURN-CODE	-	-	X
REVERSED	X	-	X
REWIND	X	-	X
REWRITE	X	-	X
RF	X	-	X
RH	X	-	X
RIGHT	X	-	X

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
ROLLBACK	-	X	-
ROUNDED	-	-	X
RUN	X	-	X
SAME	X	-	X
SD	X	-	X
SEARCH	X	-	X
SECTION	X	-	X
SECURITY	X	-	X
SEEK	-	-	X
SEGMENT	X	-	X
SEGMENT-LIMIT	X	-	X
SELECT	X	-	X
SELECTIVE	-	-	X
SEND	X	-	X
SENTENCE	X	-	X
SEPARATE	X	-	X
SEQUENCE	X	-	X
SEQUENTIAL	X	-	X
SET	X	-	X
SETS	-	X	-
SIGN	X	-	X
SIZE	X	-	X
SKIP-1	-	-	X
SKIP-2	-	-	X
SKIP-3	-	-	X
SORT	X	-	X
SORT-CORE-SIZE	-	-	X
SORT-FILE-SIZE	-	-	X
SORT-MERGE	X	-	X
SORT-MESSAGE	-	-	X
SORT-MODE-SIZE	-	-	X
SORT-RETURN	-	-	X
SOURCE	X	-	X
SOURCE-COMPUTER	X	-	X
SPACE	X	-	X
SPACES	X	-	X
SPECIAL-NAMES	X	-	X
STANDARD	X	-	X
STANDARD-1	X	-	X
STANDARD-2	-	X	-
START	X	-	X
STATUS	X	-	X
STOP	X	-	X
STORE	-	-	X
STRING	X	-	X
SUB-QUEUE-1	X	-	X
SUB-QUEUE-2	X	-	X
SUB-QUEUE-3	X	-	X
SUB-SCHEMA	-	X	-
SUBTRACT	X	-	X
SUM	X	-	X
SUPPRESS	X	-	X
SYMBOLIC	X	-	X
SYNC	X	-	X
SYNCHRONIZED	X	-	X

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
SYSIN	-	-	X
SYSOUT	-	-	X
SYSPUNCH	-	-	X
S01	-	-	X
S02	-	-	X
TABLE	X	-	X
TALLY	-	-	X
TALLYING	X	-	X
TAPE	X	-	X
TENANT	-	X	-
TERMINAL	X	-	X
TERMINATE	X	-	X
TEST	-	X	-
TEXT	X	-	X
THAN	X	-	X
THEN	-	-	X
THROUGH	X	-	X
THRU	X	-	X
TIME	X	-	X
TIME-OF-DAY	-	-	X
TIMES	X	-	X
TO	X	-	X
TOP	X	-	X
TOTALED	-	-	X
TOTALING	-	-	X
TRACE	-	-	X
TRACK-AREA	-	-	X
TRACK-LIMIT	-	-	X
TRACKS	-	-	X
TRAILING	X	-	X
TRANSFORM	-	-	X
TRUE	-	X	-
TYPE	X	-	X
UNEQUAL	-	X	-
UNIT	X	-	X
UNSTRING	X	-	X
UNTIL	X	-	X
UP	X	-	X
UPDATE	-	X	-
UPON	X	-	X
UPSI-0	-	-	X
UPSI-1	-	-	X
UPSI-2	-	-	X
UPSI-3	-	-	X
UPSI-4	-	-	X
UPSI-5	-	-	X
UPSI-6	-	-	X
UPSI-7	-	-	X
USAGE	X	-	X
USAGE-MODE	-	X	-
USE	X	-	X
USING	X	-	X
VALUE	X	-	X
VALUES	X	-	X
VARYING	X	-	X

Reserved Word	ANS COBOL	CODASYL COBOL only	OS/VS COBOL
WHEN	X	-	X
WHEN-COMPILED	-	-	X
WITH	X	-	X
WITHIN	-	X	-
WORDS	X	-	X
WORKING-STORAGE	X	-	X
WRITE	X	-	X
WRITE-ONLY	-	-	X
ZERO	X	-	X
ZEROES	X	-	X
ZEROS	X	-	X
+	X	-	X
-	X	-	X
*	X	-	X
/	X	-	X
**	X	-	X
<	X	-	X
>	X	-	X
=	X	-	X

APPENDIX G. EBCDIC AND ASCII COLLATING SEQUENCES

The ascending collating sequences for both the EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American National Standard Code for Information Interchange) character sets are given in this appendix. Decimal positions within the sequence are given, as well as the binary representation, symbol, and meaning for each character.

EBCDIC COLLATING SEQUENCE

Collating Sequence	Bit Configuration	Symbol	Meaning
0	00000000		space
.			
.			
.			
74	01001010	¢	Cent sign
75	01001011	.	Period, decimal point
76	01001100	<	Less than sign
77	01001101	(	Left parenthesis
78	01001110	+	Plus sign
79	01001111	|	Vertical bar, Logical OR
80	01010000	&	Ampersand
.			
.			
.			
90	01011010	!	Exclamation point
91	01011011	$	Dollar sign
92	01011100	*	Asterisk
93	01011101	)	Right parenthesis
94	01011110	;	Semicolon
95	01011111	¬	Logical not
96	01100000	-	Minus, hyphen
97	01100001	/	Slash
.			
.			
.			

Collating Sequence	Bit Configuration	Symbol	Meaning
107	01101011	,	Comma
108	01101100	%	Percent sign
109	01101101	_	Underscore
110	01101110	>	Greater than sign
111	01101111	?	Question mark
.			
.			
.			
122	01111010	:	Colon
123	01111011	#	Number sign
124	01111100	@	At sign
125	01111101	'	Apostrophe, prime
126	01111110	=	Equals sign
127	01111111	"	Quotation marks
.			
129	10000001	a	
130	10000010	b	
131	10000011	c	
132	10000100	d	
133	10000101	e	
134	10000110	f	
135	10000111	g	
136	10001000	h	
137	10001001	i	
.			
.			
.			
145	10010001	j	
146	10010010	k	
147	10010011	l	
148	10010100	m	
149	10010101	n	
150	10010110	o	
151	10010111	p	

Collating Sequence	Bit Configuration	Symbol	Meaning
152	10011000	q	
153	10011001	r	
.			
.			
.			
162	10100010	s	
163	10100011	t	
164	10100100	u	
165	10100101	v	
166	10100110	w	
167	10100111	x	
168	10101000	y	
169	10101001	z	
.			
.			
.			
193	11000001	A	
194	11000010	B	
195	11000011	C	
196	11000100	D	
197	11000101	E	
198	11000110	F	
199	11000111	G	
200	11001000	H	
201	11001001	I	
.			
.			
.			
209	11010001	J	
210	11010010	K	
211	11010011	L	
212	11010100	M	
213	11010101	N	
214	11010110	O	

Collating Sequence	Bit Configuration	Symbol	Meaning
215	11010111	P	
216	11011000	Q	
217	11011001	R	
.			
.			
.			
226	11100010	S	
227	11100011	T	
228	11100100	U	
229	11100101	V	
230	11100110	W	
231	11100111	X	
232	11101000	Y	
233	11101001	Z	
.			
.			
.			
240	11110000	0	
241	11110001	1	
242	11110010	2	
243	11110011	3	
244	11110100	4	
245	11110101	5	
246	11110110	6	
247	11110111	7	
248	11111000	8	
249	11111001	9	

ASCII COLLATING SEQUENCE

Collating Sequence	Bit Configuration	Symbol	Meaning
0	00000000		Null
32	00100000	SP	Space
33	00100001	|	Logical OR
34	00100010	"	Quotation mark
35	00100011	#	Number sign
36	00100100	$	Dollar sign
37	00100101	%	Percent
38	00100110	&	Ampersand
39	00100111	'	Apostrophe, prime
40	00101000	(	Opening parenthesis
41	00101001	)	Closing parenthesis
42	00101010	*	Asterisk
43	00101011	+	Plus
44	00101100	,	Comma
45	00101101	-	Hyphen, minus
46	00101110	.	Period, decimal point
47	00101111	/	Slash
48	00110000	0	
49	00110001	1	
50	00110010	2	
51	00110011	3	
52	00110100	4	
53	00110101	5	
54	00110110	6	
55	00110111	7	
56	00111000	8	
57	00111001	9	
58	00111010	:	Colon
59	00111011	;	Semicolon
60	00111100	<	Less than
61	00111101	=	Equals
62	00111110	>	Greater than

Collating Sequence	Bit Configuration	Symbol	Meaning
63	00111111	?	Question mark
64	01000000	@	Commercial At
65	01000001	A	
66	01000010	B	
67	01000011	C	
68	01000100	D	
69	01000101	E	
70	01000110	F	
71	01000111	G	
72	01001000	H	
73	01001001	I	
74	01001010	J	
75	01001011	K	
76	01001100	L	
77	01001101	M	
78	01001110	N	
79	01001111	O	
80	01010000	P	
81	01010001	Q	
82	01010010	R	
83	01010011	S	
84	01010100	T	
85	01010101	U	
86	01010110	V	
87	01010111	W	
88	01011000	X	
89	01011001	Y	
90	01011010	Z	
91	01011011	[	Opening bracket
92	01011100	\	Reverse slant
93	01011101	]	Closing bracket
94	01011110	^ ¬	Circumflex, Logical NOT
95	01011111	_	Underscore
96	01100000	`	Grave Accent

Collating Sequence	Bit Configuration	Symbol	Meaning
97	01100001	a	
98	01100010	b	
99	01100011	c	
100	01100100	d	
101	01100101	e	
102	01100110	f	
103	01100111	g	
104	01101000	h	
105	01101001	i	
106	01101010	j	
107	01101011	k	
108	01101100	l	
109	01101101	m	
110	01101110	n	
111	01101111	o	
112	01110000	p	
113	01110001	q	
114	01110010	r	
115	01110011	s	
116	01110100	t	
117	01110101	u	
118	01110110	v	
119	01110111	w	
120	01111000	x	
121	01111001	y	
122	01111010	z	
123	01111011	{	Opening Brace
124	01111100	—	Vertical Line
125	01111101	}	Closing Brace
126	01111110	~	Tilde

APPENDIX H. COBOL STATEMENTS FLAGGED BY SPECIFYING MIGR

The COBOL statements that are flagged when you specify the MIGR compiler option, grouped by subject, are:

LANGLVL(1)

- COPY language — 1968
- JUSTIFIED|JUST clause with VALUE
- MOVE ALL literal to numeric item (not flagged with SYNTAX compile option)
- MOVE/COMPARE of scaled integer to AN/ANE (not flagged with SYNTAX compile option)
- NOT in a Combined Abbreviated Relation Condition
- PERFORM in independent segment referencing a segment of different priority. (This is only significant if an ALTER is in effect in the PERFORMed segment.)
- PERFORM of independent segment
- RESERVE integer AREAS
- SELECT OPTIONAL clause — 1968 Standard interpretation
- SPECIAL-NAMES paragraph: use of L, /, and =
- UNSTRING with DELIMITED BY ALL

COMMUNICATIONS

- COMMUNICATION SECTION
- ACCEPT MESSAGE
- SEND, RECEIVE, ENABLE, and DISABLE verbs. (Note that RECEIVE ...MESSAGE is LANGLVL sensitive, but is only flagged under Communications.)

REPORT WRITER

- INITIATE, GENERATE, and TERMINATE verbs
- LINE-COUNTER, PAGE-COUNTER, and PRINT-SWITCH special registers
- Nonnumeric literal IS mnemonic-name
- REPORT clause of FD
- REPORT SECTION
- USE BEFORE REPORTING declarative

ISAM

- APPLY REORG-CRITERIA (ISAM)
- APPLY CORE-INDEX (ISAM)
- I/O verbs — all that reference ISAM files

- ISAM file declarations
- NONIMAL KEY clause
- Organization parameter "I"
- TRACK-AREA clause
- USING KEY clause on START statement

USE FOR DEBUGGING

- USE FOR DEBUGGING ON [ALL REFERENCES OF] identifiers, file-names, cd-names

BDAM

- ACTUAL KEY clause
- APPLY RECORD-OVERFLOW (BDAM)
- BDAM file declarations
- I/O verbs — all that reference BDAM files
- Organization parameters "D", "R", and "W"
- SEEK statement
- TRACK-LIMIT clause

OTHER STATEMENTS

- Assignment-name organization parameter "C" indicating ASCII
- APPLY WRITE-ONLY
- APPLY RECORD-OVERFLOW
- ASSIGN integer system-name
- ASSIGN ... OR
- CLOSE ... WITH POSITIONING/DISP
- CURRENT-DATE and TIME-OF-DAY special registers
- Debug packets
- EJECT
- EXAMINE
- EXHIBIT, READY TRACE, and RESET TRACE
- FILE-LIMITS
- FOR MULTIPLE REEL/UNIT phrase on ASSIGN clause
- LABEL RECORD ... TOTALING/TOTALED AREA
- LABEL RECORD clause on SD
- Nested OCCURS DEPENDING ON clauses (ODOs)
- Nonsubordinate item following ODO
- NOTE statement
- OCCURS ... DEPENDING ON subordinate to any OCCURS clause

- ON statement
- OPEN ... LEAVE/REREAD/DISP
- OTHERWISE synonym for ELSE on IF statement
- PROCESSING MODE
- REMARKS paragraph
- RESERVE NO/ALTERNATE AREAS
- SEARCH ... WHEN condition using KEY items as object, not subject
- SERVICE RELOAD statement
- SORT-MODE-SIZE, SORT-CORE-SIZE, SORT-MESSAGE, and SORT-FILE-SIZE special registers
- TALLY special register
- THEN as a statement connector
- TRANSFORM statement
- USE BEFORE STANDARD LABEL
- USE AFTER STANDARD ERROR ... GIVING
- USING procedure-name or file-name on CALL statement
- WRITE AFTER POSITIONING
- References to UPSI and UPSI mnemonic name
- References to unsupported compilation options on a CBL card
- WHEN-COMPILED special register

ARITHMETIC ITEMS

- Definitions of floating point data items
- Usage of floating point literals
- Usage of exponentiation

GLOSSARY

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms may or may not have the same meaning in other languages.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the following publications:

- American National Standard Programming Language COBOL, ANSI X3.23-1974 (Copyright 1974 by American National Standards Institute, Inc.), which was prepared by Technical Committee X3J4, which had the task of revising American National Standard COBOL, X3.23-1968.
- American National Standard Vocabulary for Information Processing, ANSI X3.12-1970 (Copyright 1970 by American National Standards Institute, Inc.), which was prepared by the subcommittee on Terminology and Glossary, X3.5.

American National Standard definitions are preceded by an asterisk (*).

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

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

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

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

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

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

Alphanumeric Edited Character. A character within an alphanumeric character string which contains at least one B or 0.

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

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

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

Character	**Meaning**
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

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

ASCII. American National Standard Code for Information Interchange, which allows tape file processing in accordance with the following standards:

- American National Standard Code for Information Interchange, X3.4-1968
- American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1969
- American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI), X3.22-1967

Assignment-name. A name that identifies the organization of a COBOL file and the name by which it is known to the system.

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

* **AT END Condition.** A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.
2. During the execution of a RETURN statement, when no next logical

record exists for the associated sort or merge file.

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

Binary Item. A numeric data item represented internally in binary notation (on the base 2). Binary items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign. The leftmost bit of the item is the operational sign.

* **Binary Search.** A dichotomizing search in which the number of items of the set is divided into two equal parts at each step of the process. Appropriate adjustments are usually made for dividing an odd number of items.

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

Body Group. The name for report group description entries of TYPE DETAIL, CONTROL HEADING, or CONTROL FOOTING.

Buffer. A portion of virtual storage into which data is read or from which it is written.

Byte. A sequence of eight adjacent binary bits. When properly aligned, two bytes form a halfword, four bytes a fullword, and eight bytes a doubleword.

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

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

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

Channel. The means by which vertical positioning of a printer can be indicated by means of a punched tape.

* **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 IBM.

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

Checkpoint. A reference point in a program at which information about the contents of main storage can be recorded so that, if necessary, the program can be restarted at an intermediate point.

* **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.

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

Character	**Meaning**
0,1...,9	Digit
A,B,...,Z	Letter
	Space (blank)
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol

* **COBOL Library Management Facility.** A COBOL feature that allows installations running with multiple COBOL regions/partitions to save considerable main storage by sharing some or all of the COBOL library subroutine modules.

* **COBOL Word.** (See Word.)

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

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

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

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

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

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

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

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

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

Compiler. A program which translates a program written in a higher level language into a machine language object program.

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

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

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

* **Condition.** A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, and so forth) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, and so forth) 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 implementer-defined switch or device.

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

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

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

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

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

* **Connective.** A reserved word that is used to:

1. Associate a data-name, paragraph-name, condition-name, or text-name with its qualifier.

2. Link two or more operands written in a series.

3. Form conditions (logical connectives). (See Logical Operator.)

CONSOLE. A COBOL mnemonic-name associated with the console typewriter.

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

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

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

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

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

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

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

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

Control Hierarchy. A designated sequence of report subdivisions defined by the positional order of operands within a CONTROL clause.

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

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

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

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

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

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

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

DATA DIVISION. One of the four main component parts of a COBOL program. The Data Division describes the data to be processed by the object program: files to be used and the records contained within them; internal Working-Storage records that will be needed; data to be made available in more than one program in the run unit; and/or the physical and logical characteristics of any reports to be generated.

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

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

Debugging Line. Any line with 'D' in the indicator area of the line.

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

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

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

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

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

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

Destination Queue. In communication, an MCS storage queue for one or more messages from one or more remote stations or to one or more stations. Destination queues serve as buffers between a COBOL communication program and the remote stations.

Digit. Any of the numerals from 0 through 9. In COBOL, the term is not used in reference to any other symbol.

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 IBM.

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

Direct Access Storage File. A collection of records that is assigned to a direct access storage medium.

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

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

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

* **Dynamic Access.** An access mode in which specific logical records can be obtained from or placed into a mass storage file in a 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.

EBCDIC Character. Any one of the symbols included in the eight-bit Extended Binary-Coded-Decimal Interchange Code (EBCDIC) set. All 51 COBOL characters are included.

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

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

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

EMI (End of Message Indicator). See "End of Message Indicator."

End of Group Indicator (EGI). In COBOL Communication programs, an indicator added at the end of a message group to show that the group of messages is complete.

End of Message Indicator (EMI). In COBOL Communication programs, an indicator added at the end of a message to show that the message is complete.

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

End of Segment Indicator (ESI). In COBOL Communication programs, an indicator added at the end of a message segment to show that the message segment is complete.

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

Entry-name. A programmer-specified name that establishes an entry point in a COBOL subprogram.

Entry Sequenced Data Set (ESDS). A VSAM data set in which the logical sequence of the records is determined by the order in which they are presented for creation. In COBOL, an ESDS uses sequential organization.

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

Environment Division. One of the four main component parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

ESDS (Entry Sequenced Data Set). See "Entry Sequenced Data Set."

ESI (End of Segment Indicator). See "End of Segment Indicator."

Execution Time. See "Object Time."

Exponent. A number, indicating how many times another number (the base) is to be repeated as a factor. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, and exponentiation is indicated with the symbol ** followed by the exponent.

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

External Decimal Item. A numeric data item that is represented internally in decimal notation. Such an item can contain any of the decimal digits from 0 through 9, plus a sign. Internally, the item is stored as one digit per byte; in each byte the four leftmost bits are the zone field (hexadecimal 'F'), and the four rightmost bits represent the value. If the item is not separately signed, the sign is carried in the zone field of the rightmost byte.

* **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 the following Data Division entries:

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

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

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

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

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

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

* **Format.** A specific arrangement of a set of data.

Function-name. A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, and/or program switches. When a function-name is associated with a mnemonic-name in the Environment Division, the mnemonic-name may then be substituted in any format in which such substitution is valid.

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

Header Label. A record that identifies the beginning of a physical file or a volume.

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

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

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

IDENTIFICATION DIVISION. One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program and, in addition, may include such documentation as the author's name, the installation where written, date written, etc.

* **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.

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

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

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

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

* **Indexed File.** A file with indexed organization.

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

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

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

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

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

* **INPUT PROCEDURE.** A set of statements that is executed each time a record is released to the sort file.

Input Queue. In communication, an MCS destination queue from which the COBOL communication program accepts messages from the remote stations.

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

Internal Decimal Item. A numeric data item represented internally in decimal notation. Such an item may contain any of the decimal digits from 0 through 9, plus a sign. The item is stored as two digits per byte; the sign is carried in the four rightmost bits of the rightmost byte. Also known as packed decimal.

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

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

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

* **Key Sequenced Data Set (KSDS).** A VSAM data set in which the logical sequence of the records is determined by the ascending order of an embedded prime record key. Alternate record keys may also be specified. Access to the data set is through the indexes of prime record keys and/or alternate record keys. In COBOL, a KSDS uses indexed organization.

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

KSDS (Key Sequenced Data Set). See "Key Sequenced Data Set."

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

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

* **Level-Number.** A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one or two digit number. Level-numbers in the range from 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range from 1 through 9 may be written either as a single digit or as a 0 followed by a significant digit. Level numbers 66, 77, and 88 identify special properties of a data description entry.

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

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

* **Line.** (See Report Line.)

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

* **LINKAGE SECTION.** The section in the Data Division of the called program that describes data items available from the

calling program. These data items may be referred to by both the calling and called program.

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

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

Logical Page. A vertical division of output data representing a logical separation of such data. A logical page may or may not coincide with a physical page. (See Page.)

* **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.

Main Program. The highest level COBOL program involved in a step. (Programs written in other languages that follow COBOL linkage conventions are considered COBOL programs in this sense.) Separation is based on internal logical requirements and/or external characteristics of the output medium.

Mass Storage. (See Direct Access Storage.)

Mass Storage File. (See Direct Access Storage files.)

MCP. (See Message Control Program.)

* **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 Program (MCP). A user-written communication control program that supports the processing of messages.

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

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

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

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

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

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

Name. A word composed of not more than 30 characters, which defines a COBOL operand.

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

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

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

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

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

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

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

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

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

* **Nonnumeric Literal.** A character-string bounded by quotation

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

Nonswitched Line. In communication, a line that is a continuous link between a remote station and the computer. It may connect the central computer with either a single station or more than one station.

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

Numeric Edited Item. A numeric item which is in such a form that it may be used in printed output. It may consist of external decimal digits 0 through 9, the decimal point, commas, the dollar sign, editing sign control symbols, plus other editing symbols.

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

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

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

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

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

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

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

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

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

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

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

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

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

Output Queue. In communication, an MCP destination queue into which a COBOL communication program places messages for one or more remote stations.

Overflow Condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

Overlay. The technique of repeatedly using the same areas of internal storage during different stages in processing a problem.

Packed Decimal Item. (See Internal Decimal Item.)

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

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

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

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

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

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

In the Identification Division:

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

In the Environment Division:

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

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

Parameter. A variable that is given a specific value for a specific purpose or process. In COBOL, parameters are most often used to pass data values between calling and called programs.

Password. A unique string of characters that a program, computer operator, or user must supply to meet security requirements before gaining access to data.

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

* **Physical Record.** (See Block.)

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

Printable Group. A report group that contains one or more print lines.

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

Priority-number. A number, ranging in value from 0 to 99, which classifies source program sections in the Procedure Division.

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

PROCEDURE DIVISION. One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative-statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections.

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

Process. Any operation or combination of operations on data.

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

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

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

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

Character	**Meaning**
,	Comma
;	Semicolon
.	Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
	Space
=	Equal sign

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

* **Qualifier.**

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

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

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

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

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

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

* **Record.** (See Logical Record.)

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

* **Record Description.** (See Record Description Entry).

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

* **Record Key.** A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file. within the Report Section of the Data Division.

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

Recording Mode. The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), or S (spanned).

Reel. A unit of external storage associated with a tape device.

* **Relation.** (See Relational Operator.)

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

Character	**Meaning**
>	Greater than
<	Less than
=	Equal to

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

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

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

* **Relative File.** A file with a 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 0, which specifies the record's logical ordinal position in the file.

Relative Record Data Set (RRDS). A VSAM data set in which the logical sequence of the records is determined by the ascending order of their relative record numbers from the beginning of the data set. Access to the data set can be sequential, or random through a RELATIVE KEY containing the relative record number. In COBOL, an RRDS uses relative organization.

Remote Station. In communication, a control unit and one or more input/output devices connected to the central computer through common carrier facilities. A remote station may be a terminal device, or it may be another computer.

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

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

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

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

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

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

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

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

Report Section. A Data Division section that contains one or more report description (RD) entries and their associated report group description entries.

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

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

Routine. A set of statements in a program that causes the computer to perform an operation or series of related operations.

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

RRDS (Relative Record Data Set). (See Relative Record Data Set.)

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

S-Mode Records. Records which span physical blocks. Records may be fixed or variable in length; blocks may contain one or more segments. Each segment contains a segment-descriptor field and a control field indicating whether it is the first and/or last or an intermediate segment of the record. Each block contains a block-descriptor field.

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

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

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

In the Environment Division:

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

In the Data Division:

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

REPORT SECTION.
```

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

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

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

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

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

* **Sequential File.** A file with sequential organization.

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

Sequential Processing. The processing of logical records in the order in which records are accessed.

Serial Search. A search in which each member of the set is consecutively examined, beginning with the first and ending with the last.

*** 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 0.

*** Simple Condition.** Any single condition chosen from the set:

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

Slack Bytes. Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

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

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

Sort-Work-File. A collection of records involved in the sorting operation as this collection exists on intermediate device(s).

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

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

Source Item. An identifier named in a SOURCE clause that provides the value of a printable item.

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

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

Character	**Meaning**
+	Plus sign
-	Minus sign
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than

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

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

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

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

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

*** Sub-Queue.** A logical hierarchic division of a queue.

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

*** Subprogram.** (See Called Program.)

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

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

Sum Counter. A signed numeric data item defined through a SUM clause in the Report Section of the Data Division. The Report Writer uses the sum counter to contain the result of designated summing operations that take place during production of a report.

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

Switched Line. A communication line for which no single continuous path between the central computer and the remote station exists. Several alternative paths are available for transmission; the common carrier switching equipment selects the path. The remote station is continuously connected to the switching center by an access line associated with a specific telephone number.

SYSIN. The system logical input device.

SYSOUT. The system logical output device.

SYSPUNCH. The system logical punch device.

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

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

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

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

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

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

Trailer-Label. A record that identifies the ending of a physical file or of a volume.

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

true

false

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

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

UPSI Switch. A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

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

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

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

Volume. A module of external storage. For tape devices it is a reel; for mass storage devices it is a unit.

Volume Switch Procedures. Standard procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

VSAM (Virtual Storage Access Method). A high-performance mass storage access method. Three types of data organization are available: entry sequenced data sets (ESDS), key sequenced data sets (KSDS), and relative record data sets (RRDS). Their COBOL equivalents are, respectively: sequential, indexed, and relative organizations.

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

* **WORKING-STORAGE SECTION.** The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

Zoned Decimal Item. (See External Decimal Item.)

INDEX

Special Characters

. (see decimal point and period) 79, 80
+ (see plus sign) 79, 83, 84, 86
$ (see currency sign) 79, 83, 84, 86
* (see slash and stroke) 79, 86
- (see hyphen and minus) 79, 83, 84
/ (see slash and stroke) 82
, (see comma) 79

A

A, PICTURE clause symbol
   character-string representation 79
   function of 75
abbreviated combined relation condition
   definition 456
   description and format 120
About This Book iii
ACCEPT MESSAGE COUNT statement 315, 328
ACCEPT statement
   data transfer 158
   formats 158
   mnemonic name in 158, 159
   programming notes 160
   system information transfer 159
ACCESS IS DYNAMIC
   See dynamic access mode
ACCESS IS RANDOM
   File-control clause 365
ACCESS IS SEQUENTIAL
   File-control clause 365
   WRITE statement 145
access mode
   definition 456
   description 40
ACCESS MODE clause
   description 44
   formats 41
acknowledgment
   CODASYL v
actual decimal point
   definition 456
ACTUAL KEY clause, compatibility extension 366
ADD statement
   formats 166
   common options 163
   CORRESPONDING option 163
ADVANCING option 333
   omission of 142
   WRITE statement 357
AFTER option of INSPECT 179
AFTER POSITIONING option, compatibility item 387
alignment rules
   decimal point in arithmetic statements 164
   standard 66
ALL literal figurative constant
   description 17
ALL option of INSPECT statement 178
alphabet-name
   definition 456
alphabet-name clause
   CODE-SET clause and 35
   description 35
   examples 36
   options 35
alphabetic character
   ACCEPT statement 158
   definition 456
   list of 12
ALPHABETIC class test 112
alphabetic item
   alignment rules 66
   INSPECT statement and 176
   PICTURE clause and 80
alphanumeric character, definition 456
alphanumeric class and category
   alignment rules 66
   description 66
   list of characters 12
alphanumeric edited character, definition 456
alphanumeric edited item
   alignment rules 66
   elementary move rules 173
   INSPECT statement and 176
   PICTURE clause and 81
alphanumeric item
   alignment rules 66
   INSPECT statement and 176
   PICTURE clause and 81
ALTER statement
   description and format 196
   GO TO statement and 196
   segmentation considerations 291
alternate record key
   definition 456
ALTERNATE RECORD KEY clause
   description 46
   formats 41
   WRITE statement and 144
AND logical connective
   combined conditions 116
Area A (cols. 8-11) 22
Area B (cols. 12-72) 22
arithmetic expression
   definition 456
   description 109
arithmetic operator
   definition 456
   description 109
   valid permissible symbol pairs 110
arithmetic statements
   ADD statement formats 166
   common options 163
   COMPUTE statement 171
   CORRESPONDING option 163
   DIVIDE statement format 169
   GIVING option 164
   multiple results 166
   MULTIPLY statement formats 169
   operands 165
   programming notes 166, 171
   ROUNDED option 164
   SUBTRACT statement format 167
ASCENDING/DESCENDING KEY option
   definition 456
   description 241

  - OCCURS clause 221
  - SEARCH ALL statement results 230
- ASCII, definition 456
- ASCII, processing considerations 410-412
- ASSIGN clause
  - compatibility extension 365
  - description and format 43
  - system dependencies 356
- assigning index values 224
- assignment-name
  - ASSIGN clause and 43
  - definition 456
  - RERUN clause 48
  - sequential access mode 154
- assumed decimal point
  - definition 456
- asterisk (*) 76, 79, 86
  - check protect symbol 76
  - invalid in CURRENCY SIGN clause 37
  - precedes comment line 24
- AT END condition
  - definition 456
  - READ statement, sequential access 150
  - SEARCH ALL statement 230
  - SEARCH statement 227

B

- B, PICTURE clause symbol
  - character-string representation 79
  - function of 75
  - simple insertion editing 82
- BASIS statement 297
- BEFORE/AFTER options, INSPECT 179
- binary arithmetic operators 109
- binary item
  - definition 457
- binary search of a table
  - description 229
  - format 227
- blank line
  - description 24
- BLANK WHEN ZERO clause 74
- BLOCK CONTAINS clause
  - description 58
  - formats 56
- block, definition 457
- body group
  - definition 457
- boundary violation, WRITE statement 144
- brackets ([]) 6
- buffer
  - definition 457
- byte, definition 457

C

- C
  - embedded operational sign 111
- CALL statement
  - formats 306
  - segmentation considerations 292, 307
  - subprogram linkage 301, 306
  - USING option 305, 361
- called program, definition 457
- called subprogram
  - Linkage Section 306
- calling program
  - definition 457
  - segmented 292
- card devices, multifunction 3
- CD (Communication Description) entry
  - description 315, 317, 318
  - format 319-326
  - restrictions 319
- channel, definition 457
- Character Code, CODE-SET clause 62
- character position, definition 457
- character set, COBOL definition 13
- character-string
  - and item size 67
  - definition 13, 457
  - literal 16
  - representation 79
- character, definition 457
- characters allowed
  - COBOL program 12
  - user-defined word 14
- CHARACTERS option
  - BLOCK CONTAINS clause 56
  - INSPECT statement 178
- check protect symbol 76
- checkpoint
  - definition 457
- class condition
  - definition 457
  - description and format 111
  - Numeric class test 111
- classes of data, concepts 66
- clause sequence, required order 11
- clause, definition 11, 457
- CLOSE statement
  - combined function card processing 417
  - compatibility extension 390
  - formats 155
  - system dependencies 357
- COBOL (COmmon Business Oriented Language)
  - character set 13
  - clauses and statements 11
  - divisions 10
  - general description 1
  - industry standards iii
  - language, structure of 12
  - program structure 10-11
  - words, detailed description 13
- COBOL character set
  - definition 457
- CODASYL acknowledgment v
- CODE clause
  - description 253
  - report writer 259
- CODE-SET clause
  - description 62
- collating sequence
  - and OBJECT-COMPUTER paragraph 34
  - ASCII 450
  - definition 457
- COLLATING SEQUENCE option
  - description 242
- COLUMN clause
  - format 272
  - function illustrated 255
  - invalid with null group 265
  - with printable group 265
- column 7
  - asterisk (*) specifies comment 24
  - continuation area 23
  - slash (/) specifies comment 24

- column, definition 457
- combined condition
  - definition 457
  - description and format 117
  - evaluation rules 119
  - order of evaluation 119
- comma (,) 76, 79
  - character, definition 12
  - Configuration Section 32
  - overall punctuation rules 19
  - PICTURE symbol 76
  - separator, rules for using 19
- comment
  - characters valid in 18
  - description 18
- comment line
  - definition 18, 458
  - description 24
  - in library text 294
- comment-entry
  - definition 18, 457
  - Identification Division 19, 30
- common data, subprogram linkage 300
- common processing facilities
  - description 134
- Communication Description entry
  - See CD (Communication Description) entry
- Communication Feature 313, 334
  - concepts 313
  - Data Division 317
  - Procedure Division 326
- Communication Section 317
- COMP-1 option 397
- COMP-2 option 397
- COMP-3 option 95
- COMP-4 option 95
- comparison rules
  - index names 223
  - INSPECT statement 176
  - OPEN statement execution 139
- compile time, definition 458
- COMPILE-TIME DEBUGGING PACKET 405
- compile-time switch 335
- compiler calculation of intermediate results 419
- compiler directing statement
  - debugging declarative and 209
  - definition 458
  - description 108
  - ENTER statment 209
  - list of 108
  - sentence category 108
- compiler features, OS/VS COBOL 3-4
- compiler, definition 458
- complex conditions
  - abbreviated combined relation 120
  - combined condition 117
  - definition 458
  - description and format 116
  - IF statement condition 121
  - negated simple 117
- compound condition
  - See combined condition
- COMPUTATIONAL item 95
  - valid for table KEY 221
- COMPUTATIONAL-1 option 397
- COMPUTATIONAL-2 option 397
  - compatibility extension 398
- COMPUTATIONAL-3 option 95
- COMPUTATIONAL-4 option 95
- COMPUTE statement
  - common options 164
  - format 171
  - programming notes 171
- computer-name
  - definition 458
  - SOURCE-COMPUTER, OBJECT-COMPUTER paragraph use 33
  - system-name 33
- condition
  - abbreviated combined relation 120
  - complex 116
  - condition-name 112
  - definition 458
  - negated simple 117
  - relation 113
  - sign 115
  - switch-status 116
  - to combine, listing 117
- condition-name
  - and conditional variable 69
  - definition 458
- condition-name-condition
  - definition 458
  - description and format 112
- conditional expression
  - definition 458
  - description 111
- conditional statement
  - description 105
  - GO TO statement 195
  - in IF statement 121
  - list of 106
  - PERFORM statement 200
  - specification of 121
- conditional variable
  - and condition-name entries 69
  - definition 458
  - description 69
- Configuration Section
  - definition 458
- connective words, description 15
- connective, definition 458
- console, definition 458
- contiguous items, definition 458
- continuation
  - of lines 23
- continuation area
  - column 7 21
  - description 21
  - use of asterisk (*) or slash (/) 24
- continuation line
  - description 23
- control break
  - definition 458
- control break level, definition 459
- CONTROL clause
  - description 252
  - report writer 259
- control data item, definition 459
- control data-name, definition 459
- CONTROL FOOTING report groups
  - definition 459
- control group, definition 459
- CONTROL HEADING report groups
  - definition 459
- control hierarchy, definition 459
- control transfer rules
  - explicit and implicit 27
  - segmentation considerations 289
  - subprogram linkage concepts 300
- conversion of data
  - DISPLAY statement 160
- COPY statement
  - example 296
  - format 293
  - old language extension 408

REPLACING option 294
SUPPRESS option 294
CORRESPONDING option
abbreviation 163
description 163
MOVE statment 172
SIZE ERROR option and 165
count field in INSPECT statement 178
COUNT IN option, UNSTRING 186
counter, definition 459
CR (credit) PICTURE symbol 76, 80, 83
currency sign ($) 77, 79
definition 459
CURRENCY SIGN clause
format 33
CURRENCY SIGN symbol
definition 459
currency symbol ($) 83, 84, 86
current record pointer
common processing facility 137
definition 459
READ statement 150
REWRITE statement 152
current record, definition 459
CURRENT-DATE special register 395

D

D
embedded operational sign 111
data alignment 66
data categories
description 66
PICTURE clause and 80
data clause, definition 459
data conversion
in Communication Feature 317
data description
basic concepts 64
data description entry
compatibility extension 396
data-name/FILLER clause 71
definition 459
formats 68
general description 68
indentation and 66
level-number 70
Data Division
ASCII considerations 411, 412
combined function card processing 415
data relationships 53
definition 459
external data 52
FILE-SECTION 54
format 53
general description and formats 52
in Communication Feature 315
internal data 52
LINKAGE-SECTION 55
organization 53
overview 10
punctuation in 20
report writer 257
REPORT-SECTION 55
sort/merge 238
specification of entries 23
subprogram linkage 303
table handling considerations 219
data hierarchies
used in qualification 25
data item
data description entry 68
definition 459
data item description entry (level-77 entry)
general description 54
data manipulation statements
INSPECT statement 175
MOVE statement 172
overlapping operands 172
programming notes 180, 185, 192, 193
STRING statement 181
TRANSFORM statement 192
UNSTRING statement 185
data organization
description 39
data receiving fields, UNSTRING statement 186
DATA RECORDS clause
description 60
formats 56
data reference, Procedure Division 27
data references, methods of 25-28
data transfer
See transfer of data
data truncation
See truncation of data
data-name
definition 459
duplication restriction 26
qualification format 25
subscript 215
data-name/FILLER Clause, description 71
date of compile in source listing 31
DATE special register 159
DATE-COMPILED paragraph
description and format 30
DAY special register 159
DB (debit) PICTURE symbol 76, 80, 83
DEBUG CARD, compatibility extension 405
DEBUG-ITEM special register 339
debugging features 335, 341
Debugging Language, compatibility extension 402
debugging line and section
COPY statement 294
definition 459
debugging section, definition 459
decimal point (.)
arithmetic statements 164
PICTURE symbol 76
special insertion editing 83
DECIMAL POINT IS COMMA clause
format 33
declarative procedures
description and format 126
overall description 104
declarative-sentence, definition 459
DECLARATIVES key word
begin in Area A 23
definition 459
precedes Declarative procedures 126
definition 461
DELETE statement (input/output)
dynamic access 154
format 154
programming notes 154
random access 154
DELETE statement (source program library)
format 298
DELIMITED BY ALL option, UNSTRING statement 186
DELIMITED BY option

STRING statement execution 182
delimiter
definition 459
INSPECT statement 179
STRING statement 181
UNSTRING statement 186
descending key
definition 459
DESCENDING/ASCENDING Key option
See ASCENDING/DESCENDING KEY option
DESTINATION COUNT clause 324
DESTINATION TABLE clause 325
DETAIL report group
GENERATE statement 277
GROUP indicate clause valid 273
last printable line 262
TYPE DETAIL and body group 261
detail reporting
description 256
digit position, definition 460
digit, definition 460
direct access storage
definition 460
direct indexing 216
DISABLE statement 315, 327
disk facilities 3
DISPLAY option 94
DISPLAY option of USAGE clause
DISPLAY statement
format 160
programming notes 162
DIVIDE statement
common options 164
formats 169
division header
definition 460
format, Environment Division 32
format, Identification Division 30
specification of 22
division, COBOL
definition 460
DUPLICATES option of ALTERNATE RECORD KEY clause
WRITE statement and 144
dynamic access mode
DELETE statement 154
description 40
READ statement 152
VSAM indexed files 41
dynamic access, definition 460
dynamic values in a table 218

E

E
embedded operational sign 111
EBCDIC
CODE-SET clause and 62
EBCDIC Character, definition 460
editing
signs used in 68
editing character, definition 460
editing sign
description 68
EJECT Statement 406
elementary item
alignment rules 66
basic subdivisions of a record 64
classes and categories 66
definition 460
MOVE statement 173
nonnumeric operand comparison 115
size determination 67
elementary move rules
description 173
summary 174
ENABLE statement 315, 326
End Indicator Codes 333
END KEY clause 321
end of execution 361
end of file processing 155
End of Group Indicator (EGI) 333
End of Group Indicators (EGI) 317
End of Message Indicator (EMI) 333
End of Message Indicators (EMI) 317
End of Procedure Division, definition 460
End of Segment Indicator (ESI) 333
End of Segment Indicators (ESI) 317
End of Transmission (EOT) line control character 322
END-OF-PAGE option
compatibility extension 387
description 143
WRITE statement 141
ENTER statement
description and format 209
Entry Sequenced Data Set (ESDS), definition 460
ENTRY statement
format 308
subprogram linkage 308
entry-name, definition 460
entry, definition 460
Environment Clause, definition 460
Environment Division
ASCII considerations 410, 411
Configuration Section 32
definition 460
general description 10
Input-Output Section 39
old language extension 407
punctuation in 20
report writer 256
segmentation 290
sort/merge 236
EOP option, compatibility extension 387
equal sign (=)
relation condition 12, 113
separator 18
EQUAL TO relational operator 113
error conditions
See EXCEPTION/ERROR Declarative
ERROR KEY clause 325
evaluation rules
combined conditions 119
EBCDIC 446
nested IF statement 122
EXAMINE statement, compatibility extension 399
examples
abbreviated combined relation condition 120
alphabet-name clause 36
condition-name condition 112
COPY statement 296
data transformation 193
Nested IF statement 122
STRING statement 183
subprogram linkage 310
table building and searching 231
UNSTRING statement 189, 190
EXCEPTION/ERROR Declarative
CLOSE statement and 155

compatibility extension 375
DELETE statement 154
description and format 127
EXTEND option 128
file-name option 127
general considerations 128
INPUT option 127
OUTPUT option 128
execution flow
ALTER statement changes 196
GO TO statement changes 195
PERFORM statement changes 197
STOP statement halts 207
execution-time statistics 3
EXHIBIT Statement 402
EXIT PROGRAM statement
system dependencies 361
EXIT statement 207
explicit attribute, description 27
explicit references 27-28
exponent, definition 461
exponent, fractional 171
extend mode, definition 461
EXTEND option of OPEN statement
description and format 138
VSAM indexed files 140
extended source program library
BASIS statement 297
DELETE statement 298
INSERT statement 298
external data concepts 52
external-decimal item
definition 461
DISPLAY statement 160
INSPECT statement and 176

F

F
embedded operational sign 111
FD (file description) entry
compatibility extension 370
definition 461
description and formats 56
report writer format 257
system dependencies 354
figurative constant
definition 16, 461
description 17-18
file
file clause, definition 461
file labels 59
file organization, definition 461
file processing summary 39
file processing,sample programs 422
File Section
definition 461
description and formats 56
report writer 257
FILE STATUS clause
DELETE statement and 154
description 45
formats 41
INVALID KEY option and 136
status key 134
FILE-CONTROL entry
compatibility extension 364
description and formats 41
report writing, use in 257
sort/merge 237
system dependencies 356
VSAM indexed file 42
VSAM relative file 42
FILE-CONTROL, definition 461
file-name
definition 461
description 58
formats 56
FILLER key word
description and format 71
for CD entry 319
final arithmetic results
See result field
FIPS Flagger 341, 352
UNSTRING statement 185
FIRST option of INSPECT REPLACING 179
fixed insertion editing 83
fixed length item
maximum length 68
fixed length record
size specification 58, 258
fixed length table
OCCURS clause format 219
fixed program segments 288
floating insertion editing 84
floating point, as extension 395
FOOTING option
LINAGE clause 60
FOR REMOVAL option, CLOSE statement 156
format notation, description 5-7
format, definition 461
fractional exponent 171
FROM option
ACCEPT statement 158
TRANSFORM statement 192
function-name
definition 461
SPECIAL-NAMES paragraph 34

G

general punctuation rules 19
GENERATE statement
report writing, use in 256
generic key, START statement uses 383
GIVING option, arithmetic
GIVING option 164
GIVING option, sort/merge
description 243
GO TO statement
formats 195
GOBACK statement
format 309
segmentation concepts 289
system dependencies 361
GREATER THAN symbol (>)
relation character 12
relation condition 113
group entry
GROUP INDICATE clause
group item
class and categories 66
definition 461
description 64
level-number concepts 64
MOVE statement 175
nonnumeric operand comparison 115

H

header label, definition 461
HEADING report group
high order end, definition 461
HIGH-VALUE/HIGH-VALUES figurative constant
   description 17
hyphen (-)
   in continuation area 23
   meaning as COBOL character 13

I

I-O option, OPEN statement 139
I-O-CONTROL paragraph
   compatibility extension 368
   definition 461
   description and formats 47
   sort/merge format 237
I-O-Mode, definition 461
IBM extensions
   description iii, 7
   Full ANS COBOL compatibility items 364
   how printed in this manual 7
   report writer 251
Identification Division
   compatibility extension 395
   definition 461
   description and format 30
   general description 10
identifier
   and arithmetic expressions 109
   definition 461
   general description 104
IF statement
   compatibility extension 399
   description and format 121
   nested IF 122
imperative statement
   definition 461
   general description 106
   list of 106
   sentence category 108
implicit attribute, description 27
implicit redefinition
   multiple record description 57
implicit references 27-28
indentation to clarify logic 66
indentation, COBOL format 23
independent program segments 288
index
   assigning values 224
   definition 216, 461
index data item
   comparison rules 223
   definition 222, 462
   MOVE statement rules 172
   SET TO statement, receiving field 225
INDEX usage
   format 222
index-name
   assigning values 224
   comparison rules 223
   definition 462
   description 216
   initialization 217
   OCCURS clause, INDEX BY option 222
   SET statement 224
INDEXED BY option
   OCCURS clause 222
indexed file
   definition 462
   DELETE statement 154
   organization 40
   sample programs 426
indexed organization, definition 462
indexing
   definition and format 216
   MOVE statement evaluation 172
   OCCURS clause specification 219
   restrictions for qualifiers 27
   UNSTRING statement evaluation 188
industry standards, COBOL iii
initialization by programmer
   indexes 217
   table values 218
INITIATE statement
   report writing, use in 256
input file
   definition 462
input mode, definition 462
input procedure for sort
   description 244
input procedure, definition 462
input queue 313
Input-Output Section
   definition 462
input/output device
   See assignment-name
input/output file
   definition 462
input/output statements
   ACCEPT statement 158
   CLOSE statement 155
   DELETE statement 154
   DISPLAY statement 160
   EXCEPTION/ERROR procedures, activate 128
   general description 130
   OPEN statement 138
   START statement 146
   system dependencies 357
   WRITE statement 141, 152
INSERT statement
   format 298
insertion editing 82
INSPECT statement
   comparison rules 176
   examples 179
   formats 175
   REPLACING option 178
integer
   definition 462
intermediate results
   descriptions 419
internal data concepts 52
internal decimal item
   definition 462
   DISPLAY statement 161
interrecord slack bytes 89, 92
INTO option of RETURN statement 246
intra-record slack bytes
intrarecord slack bytes 89
invalid key condition
   common processing facility 136
   definition 462
INVALID KEY option
   compatibility extension, REWRITE statement 389

  - compatibility extension, WRITE statement 388
  - DELETE statement 154
  - REWRITE statement 153
  - sequential organization 153
  - START statement execution 147
  - WRITE statement restriction 144

J

- JUSTIFIED clause 74

K

- key of reference
  - definition 462
  - indexed files and 40
- KEY option
  - OCCURS clause 221
  - READ statement 151
- Key Sequenced Data Set (KSDS) 462
- key word
  - definition 462
  - detailed description 15
  - format notation description 6
- key, definition 462

L

- LABEL RECORDS clause
  - compatibility extension 370
  - description 59
  - formats 56
  - VSAM considerations 60
- language base, OS/VS COBOL 1-2
- language considerations
  - compatibility extension 395
  - description 9-28
  - multiple languages in source program 209
  - subprogram linkage 301
- language-name
  - as system-name 14
  - definition 462
- LEADING option of INSPECT 178
- LEADING option of SIGN clause
- left truncation
  - See truncation of data
- LESS THAN symbol(<)
  - COBOL character 12
  - relation condition 113
- level indicator
  - begins in Area A 23
  - definition 462
- level number
  - begins in Area A or Area B 23
  - compatibility extension 396
  - concepts 64
  - definition 462
  - illustration 65
  - record description 64
  - rules 70
- level 02-49 entry
  - description 64
- level-01 entry
  - concepts 64
  - description 64
- level-66 entry
  - description 66
- level-77 entry
  - description 66
- level-88 entry
  - description 66
- library text, definition 462
- library-name
  - See also parentheses
  - definition 462
- LINAGE clause
  - description 60
  - formats 56
  - general considerations 61
- LINAGE integer-5/data-name-5, description 60
- LINAGE-COUNTER special register
  - END-OF-PAGE condition 144
- line advancing
  - See ADVANCING option
- LINE clause
  - LINAGE-COUNTER rules 143
  - LINEAGE-COUNTER and 62
  - report writing, use in 254
- line number, definition 462
- LINE-COUNTER special register
  - description 62
  - LINAGE clause and 62
- line, definition 462
- LINES AT TOP option of LINAGE Clause 61
- Linkage Section
  - called subprogram 306
  - definition 462
  - format 53
  - general description 55
  - subprogram linkage 300
- lister feature
  - description 3
  - 2 column format used 21
- literal
  - and arithmetic expressions 109
  - character string 16
  - definition 463
  - detailed description 16
  - nonnumeric and numeric 16
  - nonnumeric operand comparison 115
  - subscript 215
- logical connective
  - complex condition 116
  - description 15
  - list of 116
- logical operator
  - definition 463
  - in evaluation of combined conditions 118
  - list of 116
- logical page, definition 463
- logical record
  - definition 463
- low order end, definition 463
- LOW-VALUE/LOW-VALUES figurative constant
  - description 17
- lowercase, user-defined words printed in 6

M

main program
   definition 463
manual organization, description iv
maximum number/size
MEMORY SIZE clause as documentation 34
merge file
   concepts 236
   definition 463
MERGE statement
   format 240
   segmentation considerations 291
message 317
Message Control System (MCS)
   COBOL word 13
   Communication Section entry 68
   computational item 95
   data description entry 68
   description 313
   external decimal item 94
   File Section entry 68
   index value 217
   lines on printed page 60
   Linkage Section entry 68
   nonumeric literal 16
   numeric edited item character positions 81
   numeric literals 16
   PICTURE character-string 75
   Report Section entry 68
   scheduling 320
   sort records 248
   sort/merge keys 242
   subscript value 215
   table element 219
   table keys 221
   table length 219
   VALUE clause, bytes initialized in one 99
   Working-storage Section entry 68
MESSAGE COUNT clause 323
MESSAGE DATE clause 321
message segments 317
MESSAGE TIME clause 321
MIGR option, statements flagged 453
minus sign(-)
   COBOL character 12
   editing sign control symbol 76
   fixed insertion editing 83
   floating insertion editing 86
   SIGN clause, in 87
   use in PICTURE character-string 79
mnemonic-name
   ACCEPT statement operand 158
   definition 463
   DISPLAY operand 161
   SPECIAL-NAMES paragraph 34
MOVE statement
   compatibility extension 401
   CORRESPONDING option 163, 172
   elementary moves 173
   formats 172
   group moves 175
   scaling, old language extension 408
MULTIPLE FILE TAPE clause
   description 50
   format 47
multiple results, arithmetic statements
   execution rules 166
MULTIPLY statement
   common options 164
   format 168
multivolume files
   READ statement 150
   WRITE statement 144

N

name, definition 463
native character set, definition 463
native collating sequence, definition 463
negated combined condition, definition 463
negated simple condition
   definition 463
   description and format 117
negative sign
   See minus sign(-)
nested IF statement
   description 122
   flowchart 125
   true/false combinations 124
next executable sentence, definition 463
next executable statement
   definition 463
NEXT GROUP clause
   concepts 254
   format 271
   print suppression 282
next record, definition 463
NEXT SENTENCE phrase 121
noncontiguous items, definition 463
nonnumeric item
   definition 463
nonnumeric literal
   characters valid in 16
   definition 463
   description 16
   figurative constant as 17
NOT connective, old language extension 408
NOT logical connective
   See logical connective
NOTE statement 400
   TABLE HANDLING FEATURE, compatibility extension 401
numerals
   numeric literal 16
numeric characters
   definition 464
   user-defined words allowed 14
NUMERIC class test 111
numeric edited category
   alignment rules 67
numeric edited item
   alignment rules 67
   definition 464
   elementary move rules 174
   INSPECT statement and 176
   PICTURE clause and 81
numeric item
   alignment rules 67
   definition 464
   PICTURE clause and 80
numeric literal
   definition 464
   description 16
   floating point, compatibility extension 395

O

object of entry, definition 464
object program
communication scheduling 317
definition 464
object time
definition 464
PASSWORD clause control 45
OBJECT-COMPUTER paragraph
and PROGRAM COLLATING SEQUENCE clause 34
definition 464
description 34
SEGMENT-LIMIT clause 290
object-time switch 335
OCCURS clause
ASCENDING/DESCENDING KEY Option 221
compatibility extension 401
formats 219
INDEXED BY option 222
restrictions 219
table definition and 219
OCCURS DEPENDING ON clause
format 219
object 220
old language extensions 407
OMITTED option, FD entry 59
ON OVERFLOW option
STRING statement 182
UNSTRING statement 188
ON Statement 404
open mode, definition 464
OPEN statement
compatibility extension 383
description and format 138
LINAGE-COUNTER and 62
OPEN statement positions 137
options 138
system dependencies 357
operands
comparison of nonnumeric 114
comparison of numeric 114
definition 464
overlapping 165, 172, 192
operational sign
algebraic, description of 68
definition 464
valid in EBCDIC collating sequence 111
optional entries
description 43
optional word
definition 464
detailed description 15
format notation description 6
order of entries
FILE-CONTROL entry 43
I-O-CONTROL paragraph 47
Identification Division 30
order of execution
See execution flow
ORGANIZATION clause
description 44
output file
definition 464
output mode, definition 464
output procedure sort/merge
definition 464
description 244
output queue 313, 315
overflow condition
definition 464
overlapping operands invalid in
arithmetic statements 165
data manipulation statements 172
overlay, definition 464

P

P, PICTURE clause symbol
character-string representation 79
function of 75
page body, definition 464
PAGE clause
description 251
PAGE FOOTING group
definition 464
PAGE HEADING group
definition 465
PAGE-COUNTER special register
description 282
report-name qualifier 259
page, definition 464
paragraph
definition 465
general description 104
paragraph header
definition 465
specification of 22
paragraph-name
definition 465
general description 104
specification of 22
parameter, definition 465
parentheses
combined conditions, use 118
in arithmetic expressions 109
separators, rules for using 19
PARM, from JCL 361
PASSWORD clause
description 45
formats 41
option of OPEN statement 140
system dependencies 356
password, definition 465
PERFORM statement
conditional PERFORM 200
description and formats 197
execution sequences 199
EXIT statement 207
initializes index 217
segmentation considerations 291, 408
STOP statement 207
TIMES option 200
VARYING option 200
period (.) 76, 80, 83
special insertion editing 83
phrase, definition 11, 465
physical sequential file
ADVANCING option of WRITE statement 142
CLOSE statement 156
file description entry 56
I-O-CONTROL paragraph, format 47
multivolume files 150
OPEN statment options 140
RERUN clause 48
SAME clause restriction 49
status key values 136
WRITE statement format 141
PICTURE clause
and class condition 111

character string 18
compatibility extension 396
computational items and 95
CURRENCY SIGN clause 37
data categories and 80
description 74
editing 82
symbols for 75
PICTURE clause symbols
list of and descriptions 75
numeric items and 80
plus sign (+) 79, 86
editing sign control symbol 76
fixed insertion editing 83
floating insertion editing 86
floating insertion symbol 84
numeric edited items 81
use in PICTURE character-string 79
POINTER option
STRING statement execution 182
UNSTRING statement 187
POSITIONING option 386
Preface iii
prime record key
definition 465
print files
combined function card processing 417
print suppression
report writer 282
zero suppression 86
PRINT-SWITCH
report writer 282
printable item, definition 465
printable report group
definition 465
priority number
definition 465
Environment Division 290
in EXCEPTION/ERROR declarative 127
Procedure Division 290
procedure
definition 465
general description 104
procedure branching statements 195, 208, 209
executed sequentially 195
GO TO statement 195
Procedure Division 208, 209
compatibility extension 372
data reference in 27
Declarative procedures 126
definition 465
description 103
ENTER statement 209
EXIT statement 207
for Communication Feature 326
formats 105
general description 10
in Communication Feature 315
old language extension 408
organization of 104
priority-numbers 290
punctuation in 20
report writer 255
segmentation 290
sort/merge 239
subprogram linkage 304
procedure-name
definition 465
process, definition 465
processing considerations, ASCII 410
PROGRAM COLLATING SEQUENCE clause
alphabet-name clause and 35
description 34
format 33
programming notes 37
SPECIAL-NAMES paragraph and 34
program segments 288
program structure
general description 10
PROGRAM-ID paragraph
description and format 30
program-name
definition 465
PROGRAM-ID paragraph, description 31
programming notes
ACCEPT statement 160
altered GO TO statement 197
arithmetic statements 166, 171
COMPUTE statement 171
data manipulation statements 180, 185, 192, 193
DELETE statement 154
DISPLAY statement 162
EXCEPTION/ERROR procedures 129
EXIT statement 207
fractional exponents 171
INSPECT statement 180
PERFORM statement 205
RECORDS CONTAINS clause 59
SAME Clause 50
sort/merge 248
STOP statement 208
STRING statement 185
table handling 230
TRANSFORM statement 193
UNSTRING statement 192
psuedo-text
COPY statement operand 294
definition 465
psuedo-text delimiter(= =)
definition 465
separator, rules for using 19
punch files
combined function card processing 414
punctuation character
defined as separator 18
definition 465
list of 18
rules for use 19
within numeric literals 16

Q

qualification
detailed description 25
rules 26
qualified data name, definition 465
qualifier
definition 465
detailed description 25
queue structures 313
quotation mark (") character
as a separator 19
nonnumeric literal and 24
Quotation Mark, compatibility extension 406
QUOTE/QUOTES figurative constant
description 17

R

random access
   definition 466
   FILE CONTROL paragraph 365
   random or dynamic access mode 154
   READ statement and 151
RD entry
   CODE Clause 253
   CONTROL Clause 252
   description 251
   PAGE Clause 251
   report writer format 259
READ statement
   AT END condition 150
   AT END option and 385
   combined function card processing 416
   compatibility extension 385
   description and format 148
   INTO identifier option 149
   INTO/FROM identifier option 136
   KEY option 151
   random access files 151
   READ statement considerations 137
   READ statement execution 136
   sequential access 150
   VSAM files 150
READY/RESET TRACE statement 402
RECEIVE statement 315, 320, 329
receiving field
   MOVE statement 172
   multiple results rules 166
   SET statement 225
   STRING statement 181
receiving item
   arithmetic statement option rules 163
   COMPUTE statement and 171
record area
   definition 466
   description 59
RECORD CONTAINS clause
   formats 56
   report writer 258
   system dependencies 357
record description entry
   description 54
   report writer 258
record description entry, definition 466
RECORD KEY clause
   compatibility extension 367
   description 46
   formats 41
record key in indexed file
   definition 466
   DELETE statement uses 154
record-name
   definition 466
recording mode
   compatibility extension 371
   definition 466
   system dependencies 355
RECORDS option of RERUN Clause 368
RECORDS option, FD entry 59
REDEFINES clause
   compatibility extension 396
   description and format 71
   general considerations 72
   invalid in redefined item 71
   OCCURS clause restriction 72
redefinition
   implicit 57
REEL/UNIT option
   CLOSE statement 156
reel, definition 466
relation character
   COPY statement 295
   definition 466
   INSPECT statement 175, 178
   symbols 12
relation condition
   abbreviated combined 120
   definition 466
   description and format 113
   table-handling rules 223
relational operator
   definition 466
   in abbreviated combined relation condition 120
   relation condition use 113
relative file
   definition 466
   DELETE statement 154
   organization 40
   required and optional entries 133
   sample programs 430
relative indexing 216
relative key
   definition 466
RELATIVE KEY clause
   description 47
   formats 41
relative organization, definition 466
relative record data set, definition 466
RELEASE statement
   format (Sort Statement Only) 245
REMAINDER option of DIVIDE statement 170
RENAMES clause 100
REPLACING option
   COPY statement 294
   INSPECT statement 176
REPORT clause
   report writer 257
REPORT clause, description 62
report description (RD) entry, definition 466
report file
   definition 466
report footing
   definition 466
report group
   definition 466
report group description entry
   contents 254
   definition 467
   description 253
   horizontal spacing 254
   LINE Clause 254
   NEXT GROUP Clause 254
   procedure division concepts 255
   SOURCE Clause 254
   SUM Clause 254
   VALUE Clause 254
   vertical spacing 253
report heading
   definition 467
report line
   definition 467
Report Section
   definition 467
   general description 55
   report writer 258

- Report Writer
  - CODE Clause 259
  - concepts 251
  - CONTROL Clause 259
  - data division 257
  - environment division 256
  - FD entry format 257
  - FILE-CONTROL Entry 257
  - RECORD CONTAINS Clause 258
  - REPORT Clause 257
  - report section 258
  - report-name 259
  - SPECIAL-NAMES Paragraph 256
- report writer logical record, definition 467
- Report Writer special registers
  - LINE-COUNTER 283
  - PAGE-COUNTER 282
  - report writer 275
- Report Writer subroutines 255
  - report writing, use in 280
- report-name
  - definition 467
  - description 259
  - qualifier 26
- RERUN clause
  - compatibility extension 368
  - description 48
  - format 48
  - sort/merge 238
- RESERVE clause
  - format 407
  - old language extension 407
  - system dependencies 356
- reserved word
  - definition 467
  - description 15-16
- reserved word list 436
- RESET option
- result field
  - GIVING option 164
  - ROUNDED option rules 164
  - SIZE ERROR option rules 164
- RETURN statement for sort/merge
  - format 246
- RETURN-CODE special register
  - description 358
- REWRITE statement
  - compatibility extension 389
  - description and format 152
  - INTO/FROM identifier option 136
- right parenthesis())
  - See parentheses
- ROUNDED option
  - ADD statement 167
  - description 164
  - DIVIDE statement 170
  - floating point considerations 164
  - fractional exponents 171
  - size error checking and 164
  - SUBTRACT statement 168
- routine-name
  - definition 467
- routine, definition 467
- run unit
  - definition 300, 467

S

- S MODE record
  - definition 467
- S, PICTURE clause symbol
  - character-string representation 80
  - function of 75
- SAME clause
  - description 49
  - format 48
- SAME RECORD AREA clause
  - description 49
  - format 48
- SAME SORT/SORT-MERGE AREA clause
  - description 238
- sample programs
  - file processing 422
  - sort/merge 248
  - table handling 231
- SD entry
  - format 238
  - system dependencies 358
- SEARCH ALL statement
  - ASCENDING/DESCENDING KEY 221
  - compatibility extension 402
  - format 227
  - initializes index 217
- SEARCH statement
  - classification, format 290
  - binary search 229
  - description and formats 226
  - serial search 226, 227
- section
  - definition 467
  - general description 104
- section header
  - definition 467
  - general description 104
  - specification of 22
- section-name
  - as a qualifier, rules 26
  - definition 467
  - general description 104
  - in EXCEPTION/ERROR declarative 127
- SEEK statement
  - compatibility extension 385
- SEGMENT phrase 330
- SEGMENT-LIMIT clause
  - format 290
- segmentation feature 206
  - ALTER statement 291
  - CALL statement 307
  - calling and called programs 292
  - concepts 288
  - control 289
  - Environment Division 290
  - logic 289
  - PERFORM statement 205, 291
  - Procedure Division 290
  - SORT and MERGE statements 291
- SELECT clause
  - description 43
- SELECT OPTIONAL clause
  - CLOSE statement 157
  - specification 43
- semicolon (;)
  - Configuration Section 32
  - separator 18
  - series connectives 15
- SEND statement 315, 331
- sending field
  - MOVE statement 172

SET statement 225, 226
STRING statement 181
UNSTRING statement 186

sentence
categories of 108
COBOL, definition 11
definition 467
general description 104

SEPARATE option of SIGN clause 87

separate sign
class condition and 111

separator
definition 12, 467
detailed description 18

sequence number area (cols. 1-6) 21

sequential access
definition 467
DELETE statement 154
READ statement 148, 150
REWRITE statement 152, 153
WRITE statement 145

sequential file
definition 467
format 41
general description 40
organization 39
required and optional entries 131
REWRITE statement 152
sample programs 422

sequential organization
definition 467
description 39
READ statement 150

sequential processing, definition 467

serial search
definition 467
end of search 228
evaluation 227
format 226
with two WHEN options 228

series connective 15

SET statement
description and formats 224
index data item values assigned 222
initializes index 217
TO option 224
UP/DOWN BY option 226

SIGN clause 87

sign condition
definition 468
description and format 115

SIGN IS SEPARATE clause
CODE-SET clause and 62

signed data categories 68

signed numeric item
INSPECT statement and 176

simple condition
combined 117
definition 468
description and types 111
Negated 117

simple insertion editing 82

SIZE ERROR option 164
DIVIDE statement 170

SIZE-ERROR condition
definition 164

SKIP1/2/3 Statements 406

slack bytes 88, 93
definition 468
interrecord 92
intrarecord 89

slash (/) 79, 82
COBOL character 13
comment lines 24

compatibility extension 407
PICTURE character-string 81
PICTURE clause editing 82
PICTURE symbol 76

sort file
concepts 235
definition 468
SORT-CORE-SIZE 247
SORT-FILE-SIZE 247
SORT-MODE-SIZE 247

SORT statement
ASCII considerations 411
format 240

SORT-CORE-SIZE special register 359

SORT-MESSAGE special register 359

SORT-RETURN, special register 359

sort-work-file, definition 468

Sort/Merge Feature
ASCII considerations 411
concepts 235
differences and similarities 235
environment division 236
file-control entry 237
format 237
I-O-CONTROL paragraph format 237
INPUT/OUTPUT PROCEDURE Control 245
MERGE statement format 240
procedure division 239
programming notes 248
RELEASE Statement format (Sort Statement Only) 245
RERUN clause 238
RETURN Statement format 246
SAME SORT/SORT-MERGE AREA clause 238
sample program 248
SORT statement format 240
SORT-RETURN 247
special registers 247
statement options 241

Sort/Merge File Description (SD) entry
data division 238
definition 468
SD entry format 238

Sort/Merge File statement options
ASCENDING/DESCENDING KEY Option 241
COLLATING SEQUENCE Option 242
GIVING Option 243
INPUT PROCEDURE Option for sort 244
OUTPUT PROCEDURE Option 244
USING Option 243

sorted records
segmentation considerations 291

SOURCE clause
report writing, use in 254

source computer, definition 468

source item, definition 468

SOURCE LISTING FORMAT CONTROL, compatibility extension 406

source program
definition 468
standard COBOL format and 21

source program library
COPY statement 293
description 293
extended source program library 297
old language extension 408
programming notes 297
system dependencies 358

SOURCE-COMPUTER paragraph
description and format 33

space character
rules for using 18

SPACE/SPACES figurative constant
description 17

special character
   definition 468
special character word, definition 468
special features
   Report Writer 251
   segmentation feature 288
   sort/merge 235
   table handling 212
special insertion editing 83
special registers
   compatibility extension 395
   CURRENT-DATE 395
   definition 468
   description of use 15
   LINAGE-COUNTER 62
   RETURN-CODE 358
   SORT-CORE-SIZE 359
   SORT-MESSAGE 359
   SORT-RETURN 359
   sort/merge 247
   system information transfer 159
   system, DAY/DATE/TIME 159
special-character word, description 16
SPECIAL-NAMES paragraph
   alphabet-name clause and 35
   combined function card processing 415
   condition-name defined 116
   DECIMAL POINT IS COMMA Clause 37
   definition 468
   description 34
   format for report writer 257
   function-name-1 34
   function-name-2 35
   old language extension 407
   system dependencies 358
standard alignment rules
   See alignment rules
standard COBOL format
   description 21-24
standard data format
   definition 468
   description 67
standard label processing
   OPEN statement execution 140
   VSAM sequential files 140
STANDARD option, FD entry 59
STANDARD-1 option
   ASCII-encoded file specification 62
   CODE-SET clause 62
standards, COBOL iii
START statement
   description and format 146
   INVALID KEY option 147
   START statement execution 136
   START statement positions 137
   status key considerations 147
   VSAM indexed files 147
   VSAM relative files 148
statement
   ACCEPT MESSAGE COUNT 315, 328
   categories of 105
   definition 468
   DISABLE 315, 327
   ENABLE 315, 326
   general description 104
   RECEIVE 315, 320, 329
   SEND 315, 331
statements flagged by MIGR option 453
static values in a table 218
STATUS KEY clause 322, 325
status key values, VSAM files 135
status key, file processing
   common processing facility 134
   EXCEPTION/ERROR procedures check 128
   WRITE statement 142
STOP RUN statement
   description and format 207
   system dependencies 361
STOP statement
   description and format 207
storage layout of table, example 214
STRING statement
   example 183
   execution 181
   format 181
   programming notes 185
stroke 12, 76
structure of COBOL language 12
SUB-QUEUE clause
   description 320
subject of entry, definition 468
subprogram
   definition 468
subprogram linkage
   concepts 300
   Data Division 303
   examples 310
   Procedure Division 304
subscript
   definition 468
   description 215
subscripted data-name, definition 468
subscripting
   definition and format 215
   MOVE statement evaluation 172
   restrictions for qualifiers 27
   UNSTRING statement evaluation 188
substitution field of INSPECT REPLACING 178
SUBTRACT statement
   common options 163
   CORRESPONDING option 163
   formats 167
SUM clause
   report writing, use in 254
SUM counter
   definition 469
summary reporting
   description 256
SUPPRESS option, COPY 294
suppression and replacement editing 82
switch-status condition
   definition 469
   description and format 116
SYMBOLIC DESTINATION clause 326
SYMBOLIC QUEUE clause
   description 320
symbolic source 313, 321
SYMBOLIC SOURCE clause 321
SYNCHRONIZED clause 88
system considerations, subprogram linkage
   CALL statement 302
system dependencies
   description 354
   RETURN-CODE 358
system information transfer, ACCEPT statement 159
system input device, ACCEPT statement 159
system-name
   definition 469
   description 14

T

table element, definition 212, 469
table handling
  Data Division 219
  defining a table 212
  description 212
  initializing table values 218
  referring to a table element 214
  sample program 231
  table layout, example 214
table references
  indexing 216
  restrictions 217
  SEARCH ALL statement results 230
  subscripting 215
table, definition 212, 469
TALLY special register, compatibility extension 395
tallying field in INSPECT statement 178
TALLYING option
  INSPECT statement 178
  UNSTRING statement 187
TERMINATE statement
  report writing, use in 256
termination of execution
  system dependencies 361
TEXT LENGTH clause 321, 325
text-name
  COPY statement 293
  definition 469
text-word, definition 469
TIME special register 159
TIME-OF-DAY special register 395
TIMES option of PERFORM statement 200
TO option
  SET statement 224
  TRANSFORM statement 192
TRACK-AREA Clause, compatibility extension 367
trailer-label, definition 469
TRAILING option of SIGN clause
  TRANSFORM statement 192
transfer of control
  explicit and implicit 27
transfer of data
  MOVE statement 172
  STRING statement 181
TRANSFORM statement
  examples 193
  format 192
  FROM and TO options 194
  programming notes 193
truncation of data
  elementary moves 173
  ROUNDED option 164
truth value
  complex conditions 116
  condition statements specify 121
  definition 469
  IF statement 121
  sign condition 116
TSO, dependent features 4
TYPE clause
  format for use in report writer 267

U

unary arithmetic operators 109
unary operator
  definition 469
unconditional GO TO statement 195
UNIT option, CLOSE statement 156
unit record file
  definition 156
unit, definition 469
UNSTRING statement
  data receiving fields 186
  example 190
  execution 187
  format 185
  sending field 186
UPON option, DISPLAY 161
UPSI-0 through UPSI-7, program switches and switch-status condition 116
  definition 469
USAGE clause 94
  floating point items, compatibility extension 396
USAGE DISPLAY
  class condition identifier 111
  STRING statement and 181
USAGE IS INDEX
  compatibility extension 402
  description and format 222
USE AFTER EXCEPTION/ERROR sentence
  in Declarative section 126
USE BEFORE REPORTING sentence
  in Declarative section 126
  report writing, use in 256
USE FOR DEBUGGING Declarative
  for Communication Feature 336
user-defined word
  definition 469
  detailed description 14
  formation rules 14
user-defined word, list 14
user-defined words 6
USING option, SORT/MERGE statements
  description 243
USING option, subprogram linkage
  EXEC statement PARM field 361
  format 305
  linkage section 303
  subprogram linkage 305
  system dependencies 361

V

V, PICTURE clause symbol
  character-string representation 80
  funtion of 76
VALUE clause 97
  compatibility extension 398
VALUE OF clause, as documentation 60
variable length tables 219
variable, definition 469
VARYING option
  PERFORM statement 200
  SEARCH statement 228
verb
  definition 469
  profiles, compiler feature 3
Verb profiles 3
vertical spacing

report writer 253
Volume Switch Procedure
definition 469
volume, definition 469
VSAM indexed files
access modes allowed 41
CLOSE statement 157
description 40
file description entry 57
OPEN statement options 140
READ statement 150
READ statement, format 2 151
required and optional entries 132
REWRITE statement 153
START statement 147
VSAM relative files
access modes allowed 41
CLOSE statement 157
OPEN statement options 141
organization 40
PASSWORD clause in OPEN statement 141
required and optional entries 133
REWRITE statement 153
START statement 148
WRITE statement 145
VSAM sequential files
CLOSE statement 157
description 39
OPEN statement options 140
PASSWORD clause valid with 45
WRITE statement 144
VSAM, definition 469

W

WHEN option, SEARCH ALL statement 230
WHEN option, SEARCH statement 228
WHEN-COMPILED special register 359
WITH DEBUGGING MODE clause 335
WITH FOOTING option of LINAGE clause
description 60
WITH NO REWIND option, CLOSE statement 156
word
definition 469
detailed description 13
reserved word 15
Working-Storage Section
definition 469
format 53
general description 54
WRITE statement
ADVANCING option 142
AFTER POSITIONING option 386
ALTERNATE RECORD KEY 144
combined function card processing 416
compatibility extension 386
description and format 141
END-OF-PAGE option 143
INTO/FROM identifier option 136
INVALID KEY option 145
physical sequential file options 142
RELATIVE KEY 145
system dependencies 357
VSAM files 144

X

X, PICTURE clause symbol
character-string representation 79
function of 76

Z

Z, PICTURE clause symbol
character-string representation 79
function of 76
zero suppress and replacement editing 86
zero filling
elementary moves 173
zero suppression and replacement editing 86
ZERO,ZEROES,ZEROS figurative constant
description 17
permissible moves 174
reserved word 17
TRANSFORM statement, in 192

0

0, PICTURE symbol 79
0, simple insertion editing 82

9

9, PICTURE symbol 79

GC26-3857-3

IBM VS COBOL for OS/VS (File No. S370-24) Printed in U.S.A. GC26-3857-3

IBM VS COBOL for OS/VS
GC26-3857-3

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ______________________

Previous TNL ____________________

Previous TNL ____________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

GC26-3857-3

Reader's Comment Form

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape

Please do not staple

Fold and tape

IBM®

IBM VS COBOL for OS/VS (File No. S370-24) Printed in U.S.A. GC26-3857-3