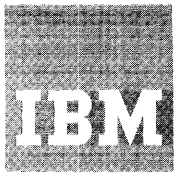
File No. S360-29
Form C28-6594-4

OS



Systems Reference Library

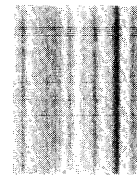
IBM System/360 Operating System

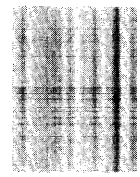
PL/I (F)

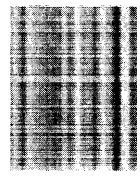
Programmer's Guide

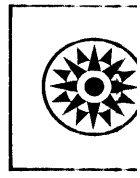
Program Number 360S-NL-511

The PL/I (F) Compiler translates PL/I source programs into object programs in System/360 machine language. This publication describes the facilities provided by the compiler, and the conventions and restrictions which the user must observe. It explains how to compile, link-edit, and execute PL/I source programs in the environment of System/360 Operating System. Features of PL/I which are not supported by the (F) Compiler are detailed in Appendix H of this publication.









PREFACE

This publication describes the facilities provided by the PL/I (F) Compiler, which functions under IBM System/360 Operating System. It covers those language restrictions and operating considerations that the user must take into account in using the (F) Compiler.

PREREQUISITE PUBLICATION

The reader is assumed to have a working knowledge of PL/I; he should therefore be familiar with the material contained in the following publication:

IBM System/360 Operating System: PL/I Reference Manual, Form C28-8201

RECOMMENDED PUBLICATIONS

The following publications are referred to in the text for information beyond the scope of this publication:

IBM System/360 Operating System:

Linkage Editor, Form C28-6538

Job Control Language, Form C28-6539

Job Control Language Charts, Form C28-6632

Operator's Guide, Form C28-6540

Supervisor and Data Management Services, Form C28-6646

System Programmer's Guide, Form C28-6550

System Generation, Form C28-6554

Utilities, Form C28-6586

Messages, Completion Codes and Storage Dumps, Form C28-6631

In addition to those mentioned above, the following publications contain information which may be helpful to the user.

IBM System/360 Operating System:

Concepts and Facilities, Form C28-6535

Principles of Operations, Form A22-6821

Storage Estimates, Form C28-6551

PL/I Subroutine Library, Computational Subroutines, Form C28-6590

Fifth Edition (November, 1968)

This is a major revision of, and obsoletes, C28-6594-3 and Technical Newsletters N33-6006 and N33-6007. Changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the • symbol to the left of the caption.

This edition applies to Release 17, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 Bibliography SRL Newsletter, Form N20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM United Kingdom Laboratories Ltd., Programming Publications, Hursley Park, Winchester, Hampshire, England.

INTRODUCTION 11

PL/I (F) Compiler. 11

Different Versions of the Compiler . . . 11
Changes at Second Version. 11
Changes at Third Version 12
Changes at Fourth Version. 12
Changes at Fourth Version, Release 17. 13
Programming Techniques and Optimization of Source Code . 14
Compatibility between Different Versions of the PL/I Library and Compiled Code 14

Language Level 16
Language Support 16
Language Features Not Supported in the Fourth Version 16
Compiler Options 16

Processing and the Operating System. . . 16
Job. 17
Job Step 17
Data Set 17

PL/I Processing. 18
Data Set Considerations. 19
Cataloged Procedures 19

JOB PROCESSING 20

Compiler Processing. 20
Compiler ddnames. 20
Compiler Device Classes 21
Blocking of Compiler Input and Output 23
Blocking of Compiler Input 23
Blocking of Compiler Output. . . . 23
Calculations for Storage Requirements. 23
SYSPUNCH/SYSLIN in Batched Compilations. 24
Compiler Options 24
Completion Codes for the Compiler . . 30
Job Control Procedure for Compilation. 30
Batched Compilation. 30
Compiler Output 31
Printed Listings. 31
Compiler Options 32
Compile-Time Processor Input Listing 32
Source Program Listing 32
Storage Requirements 33
Attribute and Cross Reference Table 33
Aggregate Length Table 35
External Symbol Dictionary Listing 36
Description of Contents of ESD Listing 36
STATIC INTERNAL Storage Map. . . . 37
Object Program Listing 38
Diagnostic Messages. 39
Time of Compilation. 40
Object Module Output. 40
Serialization of Object Decks . 40
Abnormal End of Compilation 40

Linkage Editor Processing. 40
Linkage Editor Input and Output. . 40
Unit Names and ddnames 41
Specifying Additional Libraries. . 41
Link-Editing a PL/I Subroutine into a Private Library. 41
Linkage Editor Priority. 41
Overlaying of PL/I Programs. . . . 42
Other Linkage Editor Facilities. . 44
Options for Linkage Editor Processing 44
Job Control Procedure for Linkage Editor 45
Linkage Editor Output 45
Size of Load Module. 45
Module Map 45
Cross Reference Table. 45
Disposition Data 45
Diagnostic Messages 46

Load Module Execution. 46
Execution Device Specification . . 46
DCB Parameter. 46
Job Control Procedure for Compiling, Processing by Linkage Editor, and Executing. 46
Object Program Output 47
Stream-Oriented Output 47
Record-Oriented Output 47
Object Time Diagnostic Messages. . 48
Operator Messages. 48
System Output 48

Use of Cataloged Procedures. 48
Compilation with Deck Output . . . 49
Compilation with Object Module Output. 49
Compilation and Link-Editing . . . 49
Compilation, Link-Editing and Execution 49
Link-Editing and Execution 50
Link-Editing and Execution of Several Separate Compilations . . 50

MANAGING DATA. 51

Files and Data Sets. 51
CONSECUTIVE Organization. 51
INDEXED Organization. 51
REGIONAL Organization 55
Source Keys and Recorded Keys. . . 55
REGIONAL(1) Data Sets. 56

REGIONAL(2) Data Sets. 56
REGIONAL(3) Data Sets. 57
Dummy Records within REGIONAL Data Sets 58

Data Set Definition. 58

Specifying Data Sets 58
Naming of the Data Set (DSNAME). . 58
Extent Allocation (UNIT, VOLUME, SPACE, LABEL) 58
Disposition (DISP,SYSOUT) 59
Data Set Attributes (DCB). 59
File Attributes and the DD Statement 59
Record Format Information 61
Record Types 61
Block Size 61
Logical Record Length. 62
Record Format. 62
Use of the Various Record Formats in STREAM I/O 63
Use of the Various Record Formats in RECORD I/O 64
Use of Spanned Records 64
Spanned Records and LOCATE I/O . . 65
Number of Buffers. 65
Number of Channel Programs 65
Chained Scheduling 65
Source Code (Paper tape) 65
Density (Magnetic Tape). 65
Conversion (Magnetic tape - 7-track) 66
Mode (Card Reader, Punch). 66
Stacking (Card reader, punch). . . 66
Print Spacing (Printer). 66
UCS Printer - Suppress TRANSMIT. . 66
Validity Check 66
Deleted Records. 66
Keys 66
Extended Search Limit. 67
Relative Key Position. 67
Master Indexes 67
Independent Overflow Area. 67
Cylinder Overflow Area 67
Data Set Organization. 67
PRINT Files. 68
The Tab Control Table. 69
Changing the Tab Settings. 69

Creating and Accessing Data Sets 70

Stream Data Sets 70
Creation of STREAM Data Sets . . . 70
Accessing of STREAM Data Sets. . . 70

Record Data Sets 70
Use of the EVENT Option. 70
CONSECUTIVE 70
Creation of CONSECUTIVE Data Sets. 70
Accessing of CONSECUTIVE Data Sets. 70
INDEXED 71
Creation of INDEXED Data Sets. . . 71
Accessing of INDEXED Data Sets . . 72
SEQUENTIAL Access. 72
DIRECT Access. 73
REGIONAL. 73
Creation of REGIONAL Data Sets . . 73
Accessing of REGIONAL Data Sets. . 74
SEQUENTIAL Access. 74
DIRECT Access. 75

Processing Modes in Record I/O 76
Move Mode 76
Characteristics. 76
Language Forms 76
File Attributes. 76
Locate Mode 76
Characteristics. 76
Language Forms 77
File Attributes. 77

SYSTEM REQUIREMENTS. 78

Minimum System/360 Requirements for the (F) Compiler. 78
Instruction Sets 78
Timing Clock 78
Printer Character Sets 78
Operator's Console Character Sets. 78
System/360 Operating System Requirements 78
Primary Control Program of the Operating System. 78
Multiprogramming with a Fixed Number of Tasks (MFT) 79
Multiprogramming with a Variable Number of Tasks (MVT) 79

Compiler Support 79

Object Program Support 79
Control Program Options. 79
Usage of Data Management Access Methods for STREAM I/O. 79
Usage of Data Management Access Methods for RECORD I/O. 79

MANAGING PROGRAMS. 81

Program Segmentation 81
The MAIN Option. 81
Communication Between Separate Compilations. 81
Estimation of Program Size 81

Non-PL/I Modules in PL/I Programs. . . . 82
Combination of PL/I With Other Languages 82
Variable-Length Argument List. . . 82
PL/I Library Subroutines 83

Conditional Execution of Job Steps . . . 83
Setting Conditions 83
Return Code Setting by PL/I Object Program. 83

Checkpoint/Restart 83
Compatibility with Release 11 Checkpoint/ Restart 83
Introduction 83
Step Restarts. 84
Checkpoint Restarts. 84

Single Checkpoint. 84
Multiple Checkpoints 84
Specifying Checkpoint Restart. . . 85
Restarts 86
Data Sets for Programmer-Deferred Checkpoint Restarts. 86
Card Input when Using PCP. 86
SYSIN. 86
SYSOUT - PCP 86
SYSOUT - MVT 87
Preservation of Data Sets 87
Temporary Data Sets. 87
Updated Data Sets. 87

Multitasking 87
Multitasking Requirements 87
System/360 Requirements. 87
Operating System Requirements. . . 87
Programming Requirements 88
Multitasking Management 88
Programming Considerations 88
Use of Priorities in PL/I. 90
I/O Handling 91
Strings. 92
Task Termination 93
Multiprocessing 94
Synchronization. 94
Object Program Management 95
Pseudo-Register Vector (PRV) . . . 95

PL/I Sort. 95
PL/I Sort Environment. 95
User Control of SORT ddnames . . . 96
Sorting Records from One Data Set to Another. 97
Sorting Records from a PL/I Program or Procedure onto a Data Set. 100
Sorting Records from a Data Set into a PL/I Program or Procedure 101
Supplying, Sorting, and Passing Back Records to a PL/I Procedure 102
Use of PL/I SORT in a Multitasking Environment. 103

Data Interchange 103

PROGRAMMING TECHNIQUES 104

1. Common Errors and Pitfalls 104
a. Operating System and Job Control 104
b. Source Program and General Syntax. 104
c. Program Control. 104
d. Declarations and Attributes. .105
e. Assignments and Initialization. 107
f. Arithmetic and Logical Operations. 108
g. DO groups 110
h. Data Aggregates 111
i. Strings. 111
j. Functions and Pseudo-Variables. 111
k. ON-Conditions and ON-Units . .111
l. Input/Output 112

2. Programming for Increased Efficiency. 115
a. Improving Speed of Compilation 115
b. Improving Speed of Execution .116
c. Decreasing Size of Dictionary. 120
d. Use of Storage 122
e. Use of Compile-Time Facilities. 123
f. Use of Input/Output Facilities. 124
g. Additional Hints 126

TESTING PROGRAMS 128

Debugging Facilities 128
Control of Interruption and Error Handling. 128
ON-Codes 128
Trace of Active Procedures 131
Symbolic Output. 132
Communication with the Program . .132
User Requested Dump. 132
User Completion Codes for Abnormal Termination. 133
Return Codes 134

APPENDIX A: DATA FORMATS 135

Compiler Input 135

Representation of Data 135
Coded Arithmetic Data. 135
String Data. 136
Pictured Data. 137
Data Element Descriptor (DED). . .137
Pointer Data 138
Offset Data. 138
Label Data 138
Task Data. 138
Event Data 139
Area Data. 139
Arrays 140
Structures 140
The Creation of Dope Vectors . . .140

APPENDIX B: IMPLEMENTATION CONVENTIONS AND RESTRICTIONS. 141

Input/Output Conventions and Restrictions. 141
PL/I and Data Sets 141
DISPLAY. 141
PAGESIZE 141
LINESIZE 141
LINESIZE, SKIP and COLUMN in Non-PRINT Files 142
Block Size and Record Size 142
Data-Directed Input/Output 142
Edit-Directed Input/Output 142
Character Code 142
48-Character Set 142
The ENVIRONMENT Attribute. 142
GENKEY Option. 145
EVENT Option 146

WAIT Statement 146
TITLE Option 146
BASED Variables. 146
Initializing LABEL Variables in Structures with the LIKE Attribute 146

Compile-Time Processing Conventions and Restrictions. 147
The MACRO Option 147
Precision. 147
INCLUDE Conventions. 147
Compile-Time Procedures. 147
Compile-Time DECLARE 147
Combined Level of Nesting and Depth of Replacement. 147
Limitations on Size of Compile-Time Processor Input. . .147
Limitations on Number of Compile-Time Variables. 148
Output Line Numbering. 148

Other Compiler Conventions and Restrictions. 148
OPTIONS Attribute. 148
Parameter to the MAIN Procedure. .148
Number of Variables. 148
Number of Executable Statements. .148
Size of an Individual Statement. .148
Factoring of Attributes. 149
Limitations on Nesting 149
The GENERIC Attribute. 149
Number of Blocks in a Compilation 149
Level Numbers. 149
Number of Parameters 149
Number of Dimensions 149
Array Bounds 149
Data-Directed List 150
Structure and Array Expressions. .150
Constants. 150
Sterling Constants 150
String Constants 150
Floating-Point Constants and E Format Items. 150
Constants Returned by Procedures .150
Compiler-Generated Names 150
Temporary Results in Expression Evaluation. 151
Multiple Assignments and Pseudo-Variables. 151
Function Values. 151
Qualified Names. 151
String Lengths 151
String Lengths in Intermediate Result Fields 151
AREA Sizes 151
LABEL Attribute. 152
POSITION 152
PICTURE. 152
SETS List. 152
Scale Factor 152
Precision. 152
Floating-Point Magnitude 152
Built-In Functions 152
MAX, MIN, MOD Built-In Functions .152
MOD Built-In Function. 153
COMPLETION Built-In Function and Pseudo-variable 153
STRING Built-In Function 153
Length of Identifiers. 153
Subscripted Identifiers. 153
CHECK Lists. 153

Object-Time Conventions and Restrictions. 153
Data-Directed Input. 153
Edit-Directed Output 154
CHECK Condition. 154
CONVERSION ON-Condition. 154
ON-Units and Entry Parameter Procedures. 154
Exponentiation 154
Collating Sequence 154
ENTRY Names as Arguments and ON Statements in Recursive Contexts. 154
Concatenated Data Sets 155
Locate Mode 155
ON Conditions. 155
Record Alignment 155

APPENDIX C: OBJECT PROGRAM ORGANIZATION AND CONVENTIONS. 158
Introduction. 158
Pseudo-Register Vector (PRV). 158
Run-Time Stack. 158
Dynamic Storage Area (DSA). 159
Variable Data Area (VDA). 159
Prologues and Epilogues 159
Interrupt Activity and Control. . . .160
Initial Entry to Procedures with the MAIN Option. 160
Combination of PL/I with Other Languages. 160
Calling Sequences and Register Usage. 160
Linkage Conventions for Library Modules. 161
Presentation of Arguments 161
String Dope Vector (SDV). 161
AREA Dope Vector. 162
Array Dope Vector (ADV) 162
Structure Dope Vector 163
String Array Dope Vector (SADV) . . .163
Structure Mapping 164
Rules for Order of Pairing 165
Rules for Mapping One Pair 165
Effect of UNALIGNED Attribute. . .165
Example of Structure Mapping . . .167
Allocation and Release of Storage in an Area 174

APPENDIX D: PROGRAMMING EXAMPLE. 176

APPENDIX E: CATALOGED PROCEDURES 182
Installation Modifications 182

Compilation with Deck Output 182

Compilation with Object Module Output. .183

Compilation and Link-Editing 183

Compilation, Link-Editing, and Execution 184

Link-Editing and Execution 184

Overriding Cataloged Procedures. 184
Overriding Parameters in the EXEC Statement. 184
Overriding and Adding DD Statements. 185

APPENDIX F: DYNAMIC INVOCATION OF THE COMPILER. 186

APPENDIX G : DIAGNOSTIC MESSAGES 188

Source Program Diagnostic Messages . . .188

Compile-Time Processing Diagnostic Messages. 250

Object-Time Diagnostic Messages. 260

APPENDIX H: LANGUAGE FEATURES RESTRICTED OR NOT SUPPORTED IN THE FOURTH VERSION. 272
IDENT Option 272
EVENT Option in the DISPLAY Statement 272
EVENT Option in the WRITE Statement 272
INITIAL Attribute. 272
DEFINED Attribute. 272
Structures and Arrays of Structures in Certain Special Contexts. 272
VARYING Strings. 273
Interleaved Arrays of Varying Strings Passed as Arguments . . .273
LABEL Arrays 273
The FLOAT Attribute. 273
List Processing, Table Handling, and Locate-Mode Input/Output Facilities. 273
The SECONDARY Attribute. 273
The NORMAL and ABNORMAL Attributes. 274
BASED Variables. 274
OFFSET and POINTER Built-In Functions 274
Based Variable Declaration 275
ONCOUNT Built-In Function. 275

APPENDIX I: MODEL 91 276

INDEX. 279

FIGURES

Figure 1. Job Control Language Statements. 20
Figure 2. Input/Output Data Sets 21
Figure 3. Device Class Names 21
Figure 4. BCD and EBCDIC Punched Card Codes and Graphics for PL/I 60-character Set. 28
Figure 5. Specimen ESD Listing 36
Figure 6. Linkage Editor Input/Output Flow. 41
Figure 7. Linkage Editor ddnames 42
Figure 8. Example of PL/I Procedure Using STREAM I/O. 47
Figure 9. Example of LIST/DATA/EDIT Output to PRINT Files using the PUT Statement 48
Figure 10. Invoking the Cataloged Procedure PL1DFC. 49
Figure 11. Invoking the Cataloged Procedure PL1LFC. 49
Figure 12. Invoking the Cataloged Procedure PL1LFCL 49
Figure 13. Invoking the Cataloged Procedure PL1LFCLG. 50
Figure 14. Invoking the Cataloged Procedure PL1LFLG 50
Figure 15. CONSECUTIVE Data Set Organization and Applicable Language Features. 52
Figure 16. INDEXED Data Set Organization and Applicable Language Features. 53
Figure 17. REGIONAL Data Set Organization and Applicable Language Features. 54
Figure 18. DCB Subparameters for STREAM I/O. 60
Figure 19. Additional DCB Subparameters for RECORD I/O. 60
Figure 20. Format of the Tab Control Table 69
Figure 21. Control Program Options . . . 79
Figure 22. Usage of Data Management Access Methods for RECORD-Oriented I/O 80
Figure 23. Storage Required by PL/I Library Modules when Opening and Closing Files 82
Figure 24. Basic Parameters for the SYSCHK DD Statements. 85
Figure 24.1. Figure-Length 160-byte Records, showing Fields on which Sort is to be Made 97
Figure 25. Implicit Data Conversions Performed In-Line 119
Figure 25. Implicit Data Conversions Performed In-Line (continued) 120
Figure 26. Conditions under which the String Operations are Handled In-Line .121
Figure 27. Conditions under which the String Functions are Handled In-Line. .122
Figure 28. Situations under which RECORD Condition is raised in RECORD-Oriented I/O 126
Figure 29. Main ON-Code Groupings . . .131
Figure 30. Detailed ON-Code Groupings .131
Figure 31. Abbreviations for ON-Conditions 132
Figure 32. Attributes and Precisions for Coded Arithmetic Data 136
Figure 33. Data Element Descriptor (DED) 138
Figure 34. Eight-Bit Encoded Form of Declared Information in Flags 138
Figure 35. Format of the Task Variable .139
Figure 36. Event Variable Used with I/O 139
Figure 36.1. Event Variable Used with a Task. 139
Figure 37. Format of the Area Variable. 140
Figure 37.1. Effect of LEAVE and REWIND Options on Repositioning of Magnetic-Tape Volumes 144
Figure 38. Equivalence of COBOL and PL/I Data 145
Figure 39. Format of Structure S. . . .156
Figure 40. Block Created from Structure S 156
Figure 41. Block Created by Structure S with Correct Alignment. 156
Figure 42. Alignment of Data in a Buffer in Locate Mode I/O, for Different Formats and File Organizations 157
Figure 43. Functional Content of a Dynamic Storage Area. 159
Figure 44. Initial Entry to Procedures with the MAIN Option. 160
Figure 45. Format of the String Dope Vector (SDV). 161
Figure 46. Format of the Array Dope Vector (ADV). 162
Figure 47. Format of the Structure Dope Vector (SDV) 163
Figure 48. Format of the Primary String Array Dope Vector (SADV) 164
Figure 49. Summary of Alignment Requirements for ALIGNED Data 166
Figure 49.1. Summary of Alignment Requirements for UNALIGNED Data 167
Figure 50. Mapping of Minor Structure G 168
Figure 51. Mapping of Minor Structure E 169
Figure 52. Mapping of Minor Structure N 170
Figure 53. Mapping of Minor Structure S 170
Figure 54. Mapping of Minor Structure C 171
Figure 55. Mapping of Minor Structure M 172

Figure 56. Mapping of Major Structure A 173
Figure 56.1. Offsets in Final Mapping of Structure A. 174
Figure 57. AREA Format, Showing Example of Allocated Storage and Free Elements. 175
Figure 58. Cataloged Procedure (PL1DFC) for Compilation with Deck Output. 183
Figure 59. Cataloged Procedure (PL1LFC) for Compilation with Object Module Output 183
Figure 60. Compilation and Link-Editing Cataloged Procedure (PL1LFCL) 183
Figure 61. Compilation, Link-Editing, and Execution Cataloged Procedure (PL1LFCLG). 184
Figure 62. Link-Editing and Execution Cataloged Procedure (PL1LFLG) 184

PL/I (F) COMPILER

The PL/I (F) Compiler translates PL/I source programs into object programs in System/360 machine language. It is designed to provide fast compilation and comprehensive diagnostic facilities.

The source program is maintained in storage throughout the compilation process, as far as possible, and successive phases of the compiler are passed against it. This means that the use of input/output data sets is kept to a minimum, with a consequent improvement in performance.

The compiler is of modular construction. For the compilation of a given source program, it uses only those modules that are actually required, and these are selected automatically.

A comprehensive set of compiler options is available to the user. The default values for these options are set at system generation time, and the options required for a particular compilation are selected at compilation time.

Wide use is made of modular library routines, using selective loading techniques to minimize the storage space required by object programs.

DIFFERENT VERSIONS OF THE COMPILER

This edition Form C28-6594-4 of the PL/I (F) Programmer's Guide documents the fourth version of the compiler with the improvements incorporated for Release 17 of the operating system.

Earlier versions are:

1st version Form: C28-6594-0
2nd version Form: C28-6594-1
3rd version Form: C28-6594-2
4th version Form: C28-6594-3

The more important differences between these versions of the compiler are listed below. There then follows a statement concerning the compatibility between compiled code and Library modules of various versions.

Changes at Second Version

The most significant changes for the second version of the compiler were:

RECORD I/O: The statements: READ, WRITE, REWRITE, and DELETE

The attributes: RECORD, UPDATE, SEQUENTIAL, DIRECT, BACKWARDS, BUFFERED, UNBUFFERED, and KEYED

The ON-conditions: RECORD and KEY

The built-in functions: ONFILE and ONKEY

Note: The usage UPDATE SEQUENTIAL was not supported except for INDEXED data set organization.

COMPILE-TIME PROCESSING: The compile-time processing feature of PL/I

COMPILER OPTIONS: Abbreviated names as alternatives to the full names for compiler options

ARRAY INITIALIZATION: Initialization of arrays of STATIC variables by means of the INITIAL attribute

STREAM I/O: The options PAGESIZE and LINESIZE

The ON-condition NAME

LIST/DATA-DIRECTED OUTPUT: Alignment of data on preset tab positions

RECORD FORMAT: The use of undefined-format source records

PAPER TAPE: Paper tape as input to the compiler and object program

OPERATORS: The operators ¬> and ¬<, and their 48-character set equivalents NG and NL

QUALIFIED NAMES: The resolution of apparently ambiguous name qualification

OBJECT PROGRAM LISTING: Double-column format for the object program listing

OBJECT-TIME ERROR HANDLING: Optional inclusion of the statement number in object-time diagnostic messages

Combination of SNAP output with SYSTEM action for ON statements

RECURSION ENVIRONMENTS: A change in the interpretation of ENTRY parameters and ON units in recursive contexts

CATALOGED PROCEDURES: A new cataloged procedure (PL1LFLG) for linkage editing and execution

Changes at Third Version

The most significant changes for the third version of the compiler are:

OBJECT PERFORMANCE: Changes in the object code generated by the compiler will result in considerable improvements in the object-time performance. The most significant improvements are in the following areas: data conversions, the SUBSTR function and pseudo-variable, the INDEX function, the UNSPEC function, object-time error handling and procedural housekeeping, and the usage GO TO label-variable.

ARRAY INITIALIZATION: Initialization of arrays of AUTOMATIC or CONTROLLED variables by means of the INITIAL attribute.

UPDATE SEQUENTIAL: The usage UPDATE SEQUENTIAL for CONSECUTIVE and REGIONAL data set organizations.

ASYNCHRONOUS OPERATION: The EVENT option on I/O statements, the COMPLETION built-in function and pseudo-variable, and the WAIT statement.

BATCHED COMPILATION: The facility for batched compilation of programs and a new compiler option, OBJNM.

LINKAGE EDITING: A changed method of link-editing library routines into an object program, facilitating both the link-editing of PL/I object modules from a library and the use of overlay technique with PL/I object modules.

MIXED DEFINING: The severity of diagnostic messages for defined data of type different from the type of the base is reduced from terminal to error, permitting the compilation of programs using mixed defining.

Changes at Fourth Version

The most significent changes for the fourth version of the compiler are:

LOCATE I/O AND LIST PROCESSING: The following language is now supported:

Statements and options:

```
READ FILE(filename) SET(pointer
  variable) [KEY(expression)|
  KEYTO(character-string
  variable)];
LOCATE based variable
  FILE(filename) [SET(pointer
  variable)] [KEYFROM(expres-
  sion)];
REWRITE FILE(filename);
ALLOCATE based variable
  [IN(area variable)]
  [SET(pointer variable)];
FREE based variable [IN(area
  variable)];
```

Assignment:

```
AREA to AREA
POINTER/OFFSET to
  POINTER/OFFSET
```

Attributes:

```
AREA[(expression)]
BASED (pointer)
OFFSET (based variable)
POINTER
REFER(identifier)
```

Built-in functions:

```
ADDR
EMPTY
NULL
NULLO
```

Condition:

AREA

Operation:

-> in 60-character set
PT in 48-character set

ASYNCHRONOUS OPERATIONS AND MULTITASKING : The following language is now supported:

Statements and options:

CALL statement with TASK, EVENT and PRIORITY options in any combination.
WAIT statement extended to allow array names in the event list
DISPLAY statement with REPLY and EVENT options
UNLOCK statement
NOLOCK option in READ statement

Assignment:

EVENT to EVENT

Attributes:

EVENT
EXCLUSIVE
TASK

Built-in functions/pseudo-variables:

COMPLETION
PRIORITY
STATUS

Multitasking is supported by the MVT system

Multiprocessing

DATA INTERCHANGE: The COBOL option in the ENVIRONMENT attribute; the ALIGNED/UNALIGNED attributes (for FORTRAN data interchange)

ASSEMBLER SUBROUTINES: A variable-length argument list can be passed to assembler subroutines invoked by a PL/I program

STRING HANDLING: The STRINGRANGE condition for use with SUBSTR; the STRING function

STREAM I/O: LINESIZE, SKIP and COLUMN in non-PRINT files; PUT DATA with no data list

RECORD I/O: Some types of VARYING string may be used with the INTO or FROM options; the KEY option in the DELETE statement is now optional. The DELETE statement is now supported for INDEXED data sets using SEQUENTIAL access. Four new ENVIRONMENT options (INDEXAREA, NOWRITE, REWIND, and GENKEY) provide improved performance

COMPILER OPTIONS: SIZE and SORMGIN have been changed, and four new options (OPLIST, EXTDIC, MACDCK and NEST) have been added

COMPILE-TIME OPTIMIZATION: Macro-processor concatenations are improved

OBJECT PROGRAM OPTIMIZATION: Constant subscript and constant expression evaluation; some instances of VARYING strings in assignment; in-line code for some VARYING string operations; prologue optimization; in-line handling of certain data conversions and some bit-string assignments; rounding-off (instead of truncation) for E- and F-format output; dope vector initialization improved; optimization of some IF statements.

LISTING IMPROVEMENTS: More details in attribute listings; aggregate listing is in alphabetical order; sizes of the STATIC and program control sections are given; the size of each DSA is given; statement number provided in diagnostic message for invalid pictures; improvements of aggregate length table for BASED items.

PROGRAM RESTART: The operating system checkpoint/restart facility is available under PCP

EVALUATION OF EXPRESSIONS: The order of priority is changed; concatenation now comes before the comparison and logical operators in the sequence of priority

PL/I SORT: The operating system sort program is available for use with PL/I programs

Changes at Fourth Version, Release 17

The most significant changes for the fourth version at Release 17 are:

RECORD I/O: Spanned records (VS- or VBS-format) can be specified to span blocks

Generic keys (GENKEY option) can be specified to access groups of records on an INDEXED data set

PL/I SORT: User control of SORT ddnames for multiple use of PL/I SORT within a single job step is provided

MULTIPROCESSING: More than one PL/I task may be executed simultaneously by a multiprocessing system

PROGRAM RESTART: Improved checkpoint/restart facilities are supported by PCP and MVT systems

CATALOGED PROCEDURES: Changes to some condition codes and to the dsnames for temporary data sets have been incorporated into the PL/I cataloged procedures

Programming Techniques and Optimization of Source Code

Details on avoiding common pitfalls and coding for improved efficiency are given in the section entitled Programming Techniques.

Compatibility between Different Versions of the PL/I Library and Compiled Code

Certain changes and improvements have been made to PL/I, the compiler, and the library between the four versions of the compiler. As a result, certain incompatibilities have unavoidably arisen between library modules of the different versions. The purpose of this compatibility statement is to make clear to the user what incompatibilities exist, and how the problems raised by them can be overcome.

Several changes in the fourth version prevent this version from being completely compatible with earlier versions:

1. In the evaluation of expressions, the priority of concatenation has been increased so that it now takes precedence over the comparison and logical operations.

2. In string to arithmetic conversion, the precision obtained is now the maximum precision, not the default precision.

3. E- and F-format items are now rounded on output, not truncated.

4. Multitasking is supported in this version of the compiler. Therefore a fourth-version program with the TASK option will only execute successfully with programs based on earlier versions if these programs have been recompiled with the TASK option. Even then, subprograms of these earlier programs may need rearranging to execute succesfully.

   A program containing a CALL statement with the EVENT option should be compiled with the TASK option in the main procedure. If the TASK option is not specified, it is assumed by default.

   Note: There is an incompatibility between multitasking in Release 15 and multitasking in Release 16, caused by the removal of multitasking code from some PL/I library modules. Multitasking programs coded using a Release 15 version of the compiler must be recompiled and re-link edited if they are to be executed with a Release 16 version.

5. If a procedure is to be used recursively or in a reentrant manner, the attributes RECURSIVE (for any procedure) or REENTRANT (for the main procedure) must be specified in the PROCEDURE statement. In earlier versions if these attributes were omitted, a procedure would function correctly if used recursively or in a reentrant manner.

6. The removal of PACKED from the language, and the introduction of UNALIGNED as the complementary attribute to ALIGNED, has brought differences between the current and earlier versions of the compiler in the mapping of some aggregates and in the application of default attributes.

   When a PL/I program containing a statement with the PACKED attribute is compiled by a PL/I (F) compiler that supports ALIGNED/UNALIGNED, then PACKED is recognised and ignored, and the current defaults for the data type are applied. The mapping is the same as that formerly obtained with PACKED.

   For example:

```
DCL A(5) BIT(7) PACKED;
```

The default for string elements in aggregates is UNALIGNED, which provides the same mapping as for PACKED.

```
DCL 1 STR PACKED,
        2 A FLOAT(5),
        2 B BIT(3);
```

The data-type defaults are applied; A is ALIGNED and B is UNALIGNED. The mapping is the same as that for PACKED.

It must be remembered that PACKED and UNALIGNED are not similar in meaning. If UNALIGNED is substituted for PACKED in the explicit declaration of a major structure, then there may be significant differences in the two mappings. For example:

```
DCL 1 STR PACKED,
        2 A CHAR(3),
        2 B FLOAT DECIMAL(15),
        2 C BIT(2);
```

Here A would be aligned on a byte boundary, B on a doubleword, and C on a bit boundary. If UNALIGNED were substituted for PACKED, A and B would be byte-aligned, and C bit-aligned.

The change in the default attributes has brought an incompatibility for bit string arrays. In the statement:

```
DCL A(10) FLOAT(5),
    B(5) BIT(3);
```

the former defaults for A and B would be ALIGNED; A would be an array of word-aligned arithmetic data, and B would be an array of byte-aligned bit strings. But the default for arrays is now according to data type; A is ALIGNED and hence still word-aligned and B is UNALIGNED and becomes bit-aligned. To make string arrays in programs with PACKED as default compatible (without recompiling) with those in programs with the data-type default, the latter must be explicitly declared ALIGNED.

On the other hand, the default mapping for structures is the same as before. A structure declared as:

```
DCL 1 STR,
    2 A FLOAT(5),
    2 B CHAR(4),
    2 C(10) BIT(7);
```

formerly had the default attribute PACKED; it now has default attributes that depend on the data type. The mapping is the same in both cases: A is word-aligned, B is byte-aligned, and C is bit-aligned.

7. Exponentation by integers is now changed. Previously, both operands were converted to floating-point, and the result precision was the greater of the operand precisions. Now, if the second operand is a fixed-point variable with precision (p,0), the first operand is converted to floating-point, and the precision of the result is the precision of the first operand.

Two definite compatibility statements can be made about the compilers in general:

1. Compiled code from any version of the compiler must always be executed using a library of the same version or a later version.

2. Library modules of different versions can be mixed only in the following circumstances:

   a. All link-edited modules must be of the same version as each other, and

   b. All dynamically linked or loaded modules must be of the same version as each other and must be of at least as late a version as the link-edited modules.

Unless a user has link-edited PL/I external procedures with modules from a PL/I library and placed them in a private library for future use with main programs compiled by a later version of the compiler, these incompatibilities should cause no problems. Provided the user has installed the latest compiler and library components, all future link-editing operations will result in the incorporation of the correct library modules. If, however, he has link-edited some of his external procedures, then, if he intends to use them in conjunction with a main program containing later-version library modules, he must remove the earlier-version library modules from them.

Two methods may be employed to carry this out, one temporary and the other permanent:

1. The linkage editor map for the external procedure is examined to see whether any library modules have been incorporated in the load module; these can be identified by the initial letters IHE. If there are no library modules present in the load module, no further action is required. If library modules are present, then

every time a main program needs to use the external procedure, it should be link-edited with it, using INCLUDE cards naming the library modules which are to be replaced (i.e., all of them) and an INCLUDE card naming the external procedure itself. (The latter must be a separate card and it must follow the INCLUDE card for the library modules.) This will result in the incorporation of the correct later-version library modules and then the external procedure itself. This method is temporary.

2. The permanent method is to link-edit the external procedure using the replace facility on the NAME card. In other words, the linkage editor is executed using the INCLUDE cards naming the library modules which are to be replaced, followed by an INCLUDE card naming the external procedure, followed by a NAME card naming the external procedure with the replace option.

LANGUAGE LEVEL

Language Support

The language features supported by the (F) Compiler are as defined in the publication IBM System/360 Operating System, PL/I Reference Manual. Certain minor restrictions are necessary for the efficient operation of the compiler. For full details of these restrictions refer to Appendix B of this publication.

Language Features Not Supported in the Fourth Version

In addition to the restrictions specified in Appendix B, certain areas of the language are not implemented in the fourth version of the compiler. For fuller details of features not supported by this version, refer to Appendix H of this publication.

Compiler Options

A number of compiler options are available to the user. These can be specified at compilation time as parameters on the execute statement (EXEC) card. They include the following:

Storage size

Line count

Source program listing

Object program listing

Compiler options listing

External reference listing

Attribute listing

Cross reference listing

Object program deck

Object program load file

Production of a NAME card for linkage editor processing

Optimization

Use of either BCD or EBCDIC

Specification of source margins and of carriage control character for source program listing

Use of either 48- or 60-character set

Inclusion of source program statement numbers in diagnostic messages

Level at which diagnostic messages are printed

Compile-time processing

Compile-time processor input listing

Compilation to follow compile-time processing

Extended dictionary

Nesting count for source program listing

Macro-processor card deck

Model 91 option

For details of these options, see the section called "Job Processing."

PROCESSING AND THE OPERATING SYSTEM

The basic units of processing in the operating system are the job and job step. The programmer communicates with the PL/I (F) compiler via the PL/I language; he communicates with the operating system via the job control language.

Job

On receiving a problem, a programmer analyzes that problem and constructs a precise procedure to solve it; in other words, he writes a program to solve the problem. The computer then carries out the work specified in the programmer's procedure. The amount of work specified by the programmer for the computer is called a job, and is defined by using the JOB control statement in the job control language. For example, executing a single program to solve an equation is a job to the computing system.

Job Step

If the problem is complex, the programmer may break it down into a series of steps, each step corresponding to a program. For example, the programmer may receive a tape containing raw data from an aircraft test flight. His objective may be to transform the raw data into a series of charts and reports. He may define the three following steps:

1. Refining Raw Data: Because of intermittent errors in meters and data transmission facilities, errors may occur in the raw data. The first step is to compare the raw data to projected data and to eliminate errors.

2. Developing Values: To use the refined data and a set of parameters as input to a set of equations to develop the values for the creation of charts and reports.

3. Generating Reports and Graphs: To use the values to develop points for the charts, and to print the charts and reports.

The important aspect of a job is that it is defined by the programmer. In this example the job can cover all three steps. In operating system terminology, a stage of processing executed to perform part of a job is called a job step. The programmer defines the job step to the operating system by using an EXEC control statement of the job control language.

By designating several related steps as one job, with each step designated as a job step, efficient use is made of the operating system. In the aircraft test flight example, each step may be defined as a job step in a job encompassing all the required processing, as follows:

```
JOB: Aircraft test flight

     JOB STEP 1: Refine raw data

     JOB STEP 2: Develop values

     JOB STEP 3: Generate reports and
                 graphs
```

Data Set

In the example, job step 1 used three collections of data as input and output -- raw data, projected data, and refined data. Any job step can use collections of data. In the System/360 operating system, a collection of data is called a data set. A data set is defined to the operating system by a DD (Data Definition) statement of the job control language. The word "file ," as defined for PL/I, may for the most part be equated with the term "data set," as defined for the operating system.

A data set resides on one or more volumes. A volume is a standard unit of external storage that can be written on or read by an input/output device. (For example, a volume may be a reel of tape, a disk pack, or a card deck.) The operating system provides the feature of device independence; that is, when writing his source code, the user need not concern himself with the physical device from or onto which he may be reading or writing. The same data set may at different times reside on different volumes (and hence different types of device).

The names of data sets and any information identifying the volumes on which they reside may be placed in a catalog to help the operating system find the data set. This catalog resides on the direct-access volume that contains the operating system. Any data set whose name and volume identification are placed in the catalog is called a cataloged data set. Other information concerning the data set, such as device specification, the position of the data set in the volume or the format of records in the data set, can also be accessed by the operating system. If the data set is cataloged when it is first created, the only information needed to retrieve the data set is its name.

Note: Data set names that begin with the letters SYS and have a P as the nineteenth character of the name should not be used. Data sets with such names are created for temporary data sets on PCP systems, but not for temporary data sets on MVT or MFT systems, and are deleted when the IEHPROGM

utility is used and the SCRATCH utility control statement is specified with the VTOC, PURGE, and SYS keywords.

Furthermore, a hierarchy of indexes may be devised to enable the operating system to find data sets faster. For example, an installation may divide its cataloged data sets into four groups: SCIENCE, ENGRNG, ACCNTS, and INVNTRY. In turn, each of these groups may be subdivided into groups. For example, the SCIENCE group may be divided into groups called MATH, PHYSICS, CHEM, and BIOLOGY; MATH may be further divided into ALGEBRA, CALCULUS, and BOOL. To find the data set BOOL, it is necessary to specify the names of all indexes of which it is a part, beginning with the largest group (SCIENCE), then the next largest group (MATH), and finally, the data set BOOL. The complete identification needed to find the data set BOOL is SCIENCE.MATH.BOOL.

Data set names may be either unqualified or qualified. An unqualified name is a data set name that is not preceded by an index name; for example, in the preceding text, if data sets were not indexed, BOOL would be an unqualified name. A qualified name is a data set name preceded by index names representing index levels; for example, in the preceding text, the qualified name of the data set BOOL is SCIENCE.MATH.BOOL.

Data set identification may also be based upon the time of generation. In the System/360 operating system, a collection of successive historically related data sets is called a generation data group.

Some data sets are updated periodically, or are logically part of a group of data sets, each of which is created at a different time: for example, a series of data sets used for weather reporting and forecasting. The data set name for these data sets might be WEATHER. A generation number is attached to the data set name to refer to a particular generation. The most recently cataloged data set is always 0; the generation before 0 is -1; the generation before -1 is -2; and so on. The generations for the generation data group WEATHER are:

WEATHER (0)

WEATHER (-1)

WEATHER (-2)

When a new generation data group is created, it may be called generation +1. After a job has created WEATHER (+1), the operating system changes its name to WEATHER (0), if cataloged automatically. The data set that was WEATHER (0) at the beginning of the job becomes WEATHER (-1), and so on, and the oldest one is usually deleted automatically.

PL/I PROCESSING

In the System/360 operating system, the output of the (F) compiler is called an object module (object program). The object module cannot be executed until the required references to functions and subroutines are resolved (that is, identified for the use of the operating system), and the object module is put into a format suitable for loading. Its external references are resolved by a program supplied by IBM called the linkage editor.

The output of the linkage editor is called a load module. However, the input to the linkage editor may be either object modules or load modules. Linkage editor execution can be expanded further: several object modules and load modules may be combined to form one load module. The linkage editor automatically picks out the library functions and the requested subroutines and inserts them into the load module. The name of the library is specified to the linkage editor by a SYSLIB DD card. For example, if the compiled object module TEST calls subroutines ALPHA and BETA (which in this example are assumed to be object modules) and the library function SIN (a load module), the linkage editor combines the object modules ALPHA, BETA, and TEST and the load module SIN to form a single load module.

A program written in PL/I may call subprograms written in the assembler language. An example is given in Appendix D.

After an object module is processed by the linkage editor, the resulting load module may be executed. Therefore, to execute a PL/I program, a minimum of three steps is necessary:

1. Compile the PL/I source program
2. Process the resulting object module and any load modules (which may include any PL/I subprograms) to form a load module
3. Execute the load module

Compiling, link-editing, and execution are a series of jobs and/or job steps; to the operating system, each job and job step is defined by the programmer.

Each compilation, the linkage editor execution, and the load module execution may be defined as separate jobs. (If this is done, there is only one job step in each job.) However, several single job steps may be combined into one job. For example, all the compilations may be combined. In this case, each compilation is a single job step. Furthermore, all of the steps, compilations, link-editing, and execution may be combined into one job. Then each compilation, the linkage editor processing, and the execution of the load module are separate steps. An exception is that a batched compilation facility (described later) permits more than one compilation within a single job step.

To further clarify these distinctions, assume that a source program MAIN is to be compiled and executed. MAIN requires the services of two subprograms, SUB1 and SUB2, and neither subprogram is compiled. In this example, five steps are used to perform the job:

JOB: Multiple compilations, link-editing, and execution

JOB STEP 1: Compile MAIN

JOB STEP 2: Compile SUB1

JOB STEP 3: Compile SUB2

JOB STEP 4: Process by linkage editor

JOB STEP 5: Execute the load module called MAIN

Data Set Considerations

A data set has been defined as a collection of data. The (F) compiler and linkage editor are concerned with two types of data sets, sequential data sets and partitioned data sets.

A sequential data set is a data set in which the records are arranged to be read in the sequence in which they are physically stored. A sequential data set may reside on any type of volume.

A partitioned data set (PDS) is a group of members, each of which has many of the properties of a sequential data set, and which can be used individually as a sequential data set. A partitioned data set must reside on a direct access volume, and it includes a directory which is used to locate a particular member. One of the uses of partitioned data sets is the storage of load modules. In fact, a load module can be executed only if it is a member of a partitioned data set.

Cataloged data sets and generation data groups do not have to reside on direct-access devices. A catalog is maintained by the operating system for cataloged data sets and generation data groups. However, a PDS must reside on a direct-access device, and its members must be sequential data sets. The PDS maintains its own directory of its members.

Cataloged Procedures

An installation may have certain procedures to follow in its daily processing. For example, weather reporting is processed daily. To reduce the possibility of error in the daily reproduction of these job control statements, a cataloged procedure may be written. A cataloged procedure is a set of EXEC and DD control statements that are placed in a PDS accessed by the operating system. The JOB statement cannot be cataloged. To describe a job step an EXEC statement may invoke a cataloged procedure. Because EXEC statements may be cataloged, the cataloged procedure may consist of a series of steps. The equivalent of a job step in a job is called a procedure step in a cataloged procedure. Cataloged procedures cannot be nested; that is, one cataloged procedure cannot invoke another cataloged procedure.

For a job step, a number of data sets may be defined by the DD statement. DD statements may be written in cataloged procedures. To simplify the steps involved in compiling and link-editing, five cataloged procedures have been supplied by IBM for the PL/I (F) compiler. These five cataloged procedures and their uses are:

PL1DFC	compilation with deck output
PL1LFC	compilation with object module output for linkage editor input
PL1LFCL	compilation and link-editing
PL1LFCLG	compilation, link-editing, and execution
PL1LFLG	link-editing and execution

JOB PROCESSING

To execute a PL/I program three steps are required: compilation, processing of object output by the linkage editor, and execution of the load module. These steps all require the use of job control language statements. A summary of these statements is given in Figure 1.

Statement	Function
JOB	Indicates the beginning of a new job
EXEC	First statement for each job step. It indicates the cataloged procedure or program to be executed
DD	Describes data sets and controls device and volume assignment
delimiter (/*)	Used, when data is included in the input stream, to separate data from subsequent control statements
null (//)	Marks the end of the last job in an input stream

Figure 1. Job Control Language Statements

If data is included in the input stream, the data cards must normally be preceded by a DD * statement (e.g., SYSIN DD *) and followed by a delimiter statement, although these are not always necessary when operating with a priority scheduler. For full details, refer to the publication IBM System/360 Operating System, Job Control Language.

Job processing can often be simplified by using the cataloged procedures described in this chapter and in Appendix E of this publication.

COMPILER PROCESSING

The names for DD statements (ddnames) connect I/O statements in the compiler with data sets used by the compiler. Names for I/O device classes have also been established and must be used by the programmer. The program name for the compiler is IEMAA, so that if the compiler is to be executed in a job step, the parameter PGM=IEMAA must be used in the EXEC statement. Normally, however, the parameter would appear on an EXEC card within a cataloged procedure.

If the compiler is to be dynamically invoked by the CALL, LINK, XCTL, or ATTACH macro instructions, details of the method to be used will be found in Appendix F.

COMPILER DDNAMES

The (F) compiler uses a maximum of seven standard data sets. Each data set has been assigned a specific ddname in order to establish communication between the compiler and the programmer. Each data set is given a specific function which must meet device requirements for the PL/I (F) compiler. The ddnames, functions, and device classes for the data sets are given in Figure 2. These ddnames must be specified as ddnames for compiler DD statements, and must correspond to the function listed in Figure 2. The requirements for each data set designate which type of input/output device must be used for the data set. In addition to the seven standard data sets, each data set explicitly referenced in a compile-time INCLUDE statement must also have a corresponding DD statement. The device requirements for these data sets are the same as for the SYSLIB data set.

To compile a PL/I program on the (F) compiler, two of these data sets are necessary: SYSIN and SYSPRINT, along with the direct-access volume that contains the operating system. With these two data sets, only the listing is generated by the compiler, If an object deck or a MACDCK deck is to be provided, a SYSPUNCH DD statement must be specified. If the object module is to be written, a SYSLIN DD statement must be supplied. SYSUT1 is always required if the SIZE option (explicitly or by default) specifies a value less than 53,248. With higher values of SIZE, the SYSUT1 data set is used only when the source program and internal tables cannot be wholly contained in main storage. The table of space requirements for SYSUT1 (see "Compiler Device Classes") is a guide to when the data set will be used. For practical purposes, however, it is advisable to include the DD statement for SYSUT1 as a matter of course. SYSUT3 is used for compile-time processing, and for programs using the 48-character set. SYSLIB is used for compile-time processing of INCLUDE statements in which a ddname is not explicitly specified.

ddname	Function	Possible Device Classes
SYSIN	Source Input	SYSSQ, the input stream device (specified by DD *), or paper tape reader
SYSPRINT	Listing Output	SYSSQ, or SYSOUT device
SYSPUNCH	Deck and MACDCK Output, or SYSOUT (punch) device	SYSCP, SYSSQ
SYSLIN	Object Module Output	SYSSQ
SYSUT1	Auxiliary Storage for Text and Dictionary	SYSDA
SYSUT3	Auxiliary Storage for Compile-Time Processing and 48-Character Syntax	SYSSQ
SYSLIB	Default input for compile-time INCLUDE statements	SYSDA

Figure 2. Input/Output Data Sets

Class Name	Class Function	Device Type
SYSSQ	Writing, reading	Magnetic tape, DASD
SYSDA	Writing, reading, updating records in place	DASD
SYSCP	Intermediate or ultimate device for punched cards	Card punch, magnetic tape DASD

Figure 3. Device Class Names

For the DD statements SYSIN, SYSPUNCH, or SYSPRINT, an intermediate storage device may be specified instead of the card reader, card punch, or printer, respectively.

If an intermediate storage device is specified for SYSIN, the compiler assumes that the source program deck was placed there by a previous job or job steps. If an intermediate storage device is specified for SYSPRINT, the listing and diagnostic messages are written on the device; a new job or job step can print the contents of the data set. Similarly, if an intermediate storage device is specified for SYSPUNCH, the card deck is written on that device, and another job or job step will be needed to punch the card deck. Under the MVT system, this can be done automatically by specifying a SYSOUT data set.

COMPILER DEVICE CLASSES

Names for input/output device classes used for compilation are specified to the operating system at system generation time. The usual class names, functions, and types of devices are shown in Figure 3.

The data sets used by the compiler must be assigned to devices eligible for the classes listed in Figure 2.

It should be noted that a direct-access storage device may be used for all compiler devices. The SPACE parameter in the DD statement must be used if there is a possibility that the data set will be written on a direct-access device.

For most practical situations, the values used in the SPACE parameters of the data sets defined in the cataloged procedures in Appendix E will be adequate. However, typical space requirements for various data sets can be calculated from the following information (All values given are in bytes):

IR = number of input records (i.e., number of records containing source text + number of records included through the compile-time INCLUDE statement).

SR = number of records containing source statements after compile-time time processing is completed.

SN = number of source statements after compile-time processing is completed.

VN = number of variables used in the program.

SYSPRINT Space Requirements

Space required =

```
120x( IR          if SOURCE2 option
                  is specified
    + SR          if SOURCE option
                  is specified
 + 10xSN          if LIST option is
                  specified
    + VN          if ATR option is
                  specified
    + VN          if XREF option is
                  specified
  + 30) bytes     if EXTREF option
                  is specified
```

SYSLIN and SYSPUNCH Space Requirements

Space required = 80 x SN bytes

SYSUT1 Space Requirements

Source programs which are large enough to cause the SYSUT1 data set to be used will compile more efficiently if this data set resides on a drum or in a contiguous area on a disk.

Since it is not advisable to use EXTDIC if SIZE=44K, the figures for this value must be taken as applying to a 48K capacity.

SN	SIZE	No. of tracks required		
		2311	2301	2321
150	44K	8	4	22 24*
	100K	0	0	0
	200K	0	0	0
500	44K	31	14	92 97*
	100K	18 22*	9 11*	27 33*
	200K	0	0	0
1000	44K	64	28	192 202*
	100K	68 74*	34 37*	102 111*
	200K	15 20*	9 12*	27 28*

* EXTDIC option specified

SYSUT3 Space Requirements

Space required for 48-character set processing = 2 x SR x (average record length) bytes

Space required for compile-time processing = size of program after compile-time processing

Space required for both 48-character set and compile-time processing = 2 x (size of program after compile-time processing

These formulas give only approximate answers, in that considerable variation may result from the nature of individual source programs. The figures given should provide adequate space in most cases. The secondary allocation facility should be used as a protection against exceptional requirements.

SYSUT3 Records

Record parameters on this data set depend on whether the MACRO or the CHAR48 option is used.

If the MACRO option is specified (with or without the CHAR48 option), SYSUT3 will be opened with the following parameters:

Record format: F(available size < 56K)
FB(available size ≥ 56K)

Record size: 80 bytes

Blocking factor: 1(available size < 56K)
2(available size ≥ 56K)

For available size, see 'SIZE Option'.

If the CHAR48 option (but not the MACRO option) is specified, SYSUT3 contains a CHAR48 and a CHAR60 copy of each source record. It is opened with the following parameters:

Record format: F(available size < 56K)
FB(available size ≥ 56K)
U(if SYSIN record format is U)

Record size: SYSIN record length

Blocking factor: 1(available size < 56K)
2(available size ≥ 56K)

For available size, see 'SIZE Option.'

Data Set Write Verify (DASD)

The Write Verify option for DASD is not fully supported for SYSUT1. Although the Write Verify may be requested by specifying OPTCD = W in the DCB for SYSUT1, not all the data blocks written on this data set will in fact be verified. Specifying the OPTCD parameter results in write verification only when a new dictionary or text block is written on the device; reuse of this block will not result in verification of the data written.

The Write Verify feature is always available for use with a data block on other data sets on direct-access devices, and is obtained by specifying the appropriate option on the data set DD card. (See 'Supervisor and Data Management Services')

BLOCKING OF COMPILER INPUT AND OUTPUT

Blocking of Compiler Input

The compiler allows the programmer to request blocked input on SYSIN (other than SYSIN DD DATA or *) by specifying a BLKSIZE parameter on the appropriate DD card or by obtaining the block size from the data set label (see Appendix A, "Compiler Input").

Of the 44K bytes minimum storage necessary for the F compiler, 1000 bytes are reserved for two SYSIN buffers, so that the SYSIN block size can always be up to 500 bytes (i.e., the size of one SYSIN buffer). A block size greater than 500 bytes is allowed if the extra storage necessary for the SYSIN buffers is available in the amount of storage requested in the SIZE option (i.e., SIZE must be at least 44K - 1K + 2 x block size). If the SIZE option does not allow for 45056 bytes more than the additional storage required by the SYSIN buffers, compilation will be terminated.

Blocking of Compiler Output

The compiler allows the programmer to request blocked output on SYSPRINT, SYSPUNCH and/or SYSLIN by specifying a BLKSIZE parameter on the appropriate DD card.

Of the 44K bytes minimum storage necessary for the F compiler, 258 bytes are reserved for two SYSPRINT buffers (129 bytes each), 400 bytes are reserved for the SYSPUNCH buffer, and 400 bytes are reserved for the SYSLIN buffer. Thus when SIZE=44K, SYSPRINT block size must be 129 bytes and SYSPUNCH and SYSLIN block sizes can be up to 400 bytes. Greater block sizes than these are allowed if extra storage for larger buffers is specified in the SIZE option. (See next section "Calculations for Storage Requirements.")

If the SYSPUNCH block size or the SYSLIN block size is greater than 400 bytes and the SIZE parameter is not great enough to accommodate it, then the corresponding compiler options will be deleted.

SYSPRINT block size must be of the form (4 + n x 125) bytes. If the SIZE option does not allow for 45056 bytes more than the additional storage required by the SYSPRINT buffers, compilation will be terminated.

The E-level Linkage Editor cannot accept blocked input, so the compiler output should be blocked only if the F-level Linkage Editor is in use.

Calculations for Storage Requirements

Storage requirements for blocking compiler input and output should be based on the following:

44K is needed for the compilation, and of this

1000 bytes are saved for two SYSIN buffers (500 bytes each)
258 bytes are saved for two SYSPRINT buffers (129 bytes each)
400 bytes are saved for the SYSPUNCH buffer
400 bytes are saved for the SYSLIN buffer

If SYSIN block size is greater than 500 bytes then (2 x SYSIN block size - 1K) is subtracted from the storage specified by SIZE.

If SYSPRINT block size is greater than 129 bytes then (2 x SYSPRINT block size - 258) is subtracted from the remaining storage.

If SYSPUNCH block size is greater than 400 bytes then (SYSPUNCH block size - 400) is subtracted from the remaining storage.

If SYSLIN block size is greater than 400 bytes then (SYSLIN block size - 400) is subtracted from the remaining storage.

The storage that finally remains must be equal to or greater than 44K.

SYSPUNCH/SYSLIN in Batched Compilations

Once SYSPUNCH or SYSLIN has been opened, it is left open and buffer space is allocated for all subsequent compilations in the batch, whether or not there is any SYSPUNCH or SYSLIN output for those compilations.

If SYSPUNCH or SYSLIN block sizes greater than 400 are asked for, the extra space needed for the larger buffers will be subtracted from the SIZE specified for each compilation in the batch before calculating the text and dictionary block sizes. In some cases, compilations which do not have SYSPUNCH or SYSLIN output may be allocated smaller text and dictionary block sizes than they would otherwise get if they follow a compilation which opened SYSPUNCH or SYSLIN. If SYSPUNCH or SYSLIN has been successfully opened with a big block size and a following compilation specifies SIZE < (44K + extra buffer space needed), this will be ignored and SIZE = (44K + extra buffer space needed) will be assumed.

COMPILER OPTIONS

A number of compiler options are available to the programmer. These may be passed to the compiler through the PARM parameter on the EXEC card at compilation time. Included in the information that may be specified to the compiler are the following items:

The amount of main storage allocated to the compiler for this compilation

The number of lines to be printed on each page of the source listing

Whether a source program listing is to be printed

Whether an object program listing is to be printed

Whether a list of the compiler options specified for the program is to be printed

Whether a listing of the External Symbol Dictionary (ESD) is to be provided

Whether a list of the attributes of identifiers is to be provided

Whether a list of cross-references to identifiers is to be provided

Whether the compiler is to produce an object module for input to the linkage editor, output either as a data set on SYSLIN, or SYSPUNCH, or both, as the user requires

Whether the compiler is to produce the linkage editor control statement NAME to follow either the object module or the object program deck

The level of optimization required

Whether the source code is in BCD or EBCDIC

The specification of source margins and the specification of a carriage control character for the source program listing

The choice of 48- or 60-character set for source programs

Whether diagnostic messages produced during execution are to include statement numbers from the source program

The choice of severity level at which diagnostic messages will be printed

Whether compile-time processing is required

Whether a compile-time processor input listing is to be printed

Whether compile-time processing is to be followed by compilation

Whether the extended dictionary is to be used

Whether a nesting block level is to be printed next to each statement number

Whether a sequenced card deck is to be punched as the macro-processor output

Whether the object program is to be executed on a System/360 Model 91

Compiler option	Abbreviated name	Standard default
SIZE=yyyK|yyyyyy	SIZE	999999
LINECNT=xxx	LC	50
SOURCE|NOSOURCE	S|NS	SOURCE
LIST|NOLIST	L|NL	NOLIST
OPLIST|NOOPLIST	OL|NOL	OPLIST
EXTREF|NOEXTREF	E|NE	NOEXTREF
ATR|NOATR	A|NA	NOATR
XREF|NOXREF	X|NX	NOXREF
DECK|NODECK	D|ND	NODECK
LOAD|NOLOAD	LD|NLD	LOAD
OBJNM=aaaaaaaa	N	No action
OPT=nn	O	01
BCD|EBCDIC	B|EB	EBCDIC
SORMGIN=(mmm,nnn[,ccc])	SM	(2,72)
CHAR60|CHAR48	C60|C48	CHAR60
STMT|NOSTMT	ST|NST	NOSTMT
FLAGW|FLAGE|FLAGS	FW|FE|FS	FLAGW
MACRO|NOMACRO	M|NM	NOMACRO
SOURCE2|NOSOURCE2	S2|NS2	SOURCE2
COMP|NOCOMP	C|NC	COMP
EXTDIC|NOEXTDIC	ED|NED	NOEXTDIC
NEST|NONEST	NT|NNT	NONEST
MACDCK|NOMACDCK	MD|NMD	NOMACDCK
M91|NOM91		NOM91

The list of compiler options following the equals sign in the PARM parameter must be enclosed in quotation marks; the separate options are separated by commas. The list must not exceed 40 characters including commas, but excluding the quotation marks. Because of the 40 character limitation, the compiler has been designed to accept the abbreviated compiler option names given in the table below, as alternatives to the longer mnemonics. Where an option includes a numerical specification, as in SIZE, LINECNT, OPT, and SORMGIN, only significant digits need be specified by the user. There is no required order for specifying the compiler options, but if conflicting options are specified the last specification in the list will be used. The default for all options (except OBJNM) can be set at system generation time to suit the requirements of the installation. In the absence of any such setting, the standard defaults shown in the table below will be effective. The DELETE parameter of the system generation PL1 macro instruction (described in the publication IBM System/360 Operating System, System Generation) may be used at system generation time to specify any compiler options not to be used at compilation time.

SIZE=yyyyyy|yyyK|999999 - The SIZE option indicates to the compiler the amount of main storage (in bytes) available for the compilation. The programmer specifies this amount as one of the following:

yyyyyy: This gives the number of bytes available for the compilation. For blocked input, the amount yyyyyy must be greater than 45056 + (2*SYSIN BLKSIZE if BLKSIZE>500) + (2*SYSPRINT BLKSIZE if SYSPRINT is blocked). If it is not, the compiler will terminate abnormally. For unblocked input, if the amount yyyyyy is less than 45056, the default size is taken.

yyyK: This gives the number yyy of K units (K = 1024 bytes) available for the compilation. If the number of bytes specified is less than 44K, a message is printed and the default size is taken.

999999: This instructs the compiler to obtain as much main storage as it can. If this amount is less than 44K, the compiler will comment and then will attempt to continue; later it may terminate abnormally. Because the compiler calculation of the amount of storage available is only an approximation, SIZE=999999 should not be used if the amount of storage available is less than 48K.

The partition (MFT) or region (MVT) in which the compilation step is executed must be at least 8K larger than the amount specified in the SIZE option. Failure to ensure this may result in a system abnormal termination before the compiler can provide any diagnostic aid.

The following table shows the text and dictionary block sizes used for various core availabilities as specified by the SIZE option. The available size is the amount specified in the SIZE option minus the extra space needed for larger input and output buffers. The text and dictionary

block sizes are determined from the amount available after any extra storage needed for larger input/output buffers has been subtracted from the amount specified in the SIZE option.

Available Size (bytes)	Block Size (bytes)
45,056 - 57,343	1,024
57,344 - 73,727	2,048
73,728 - 135,167	4,096
135,168 - 172,031	8,192
172,032 or more	16,384

LINECNT=xxx - the LINECNT option informs the compiler how many lines are to be printed on a page of listing. The number specified includes heading lines and blank lines. If a number is not specified, a standard default of 50 lines is assumed.

SOURCE or NOSOURCE (Source Program Listing) - the SOURCE option specifies that the source program is to be written on the device indicated on the SYSPRINT DD card. The source listing is in the same character set as the source records. The statement number of the first statement in each line is printed to the left of that line, but a statement number is not given on a line containing only a continued statement. NOSOURCE indicates that no source listing is required. If neither option is specified, the compiler assumes the standard default SOURCE. A description of the listing is given in the section called "Compiler Output."

LIST or NOLIST (Object Program Listing) - the LIST option specifies that generated machine instructions are to be listed in a format resembling the listing output of the operating system/360 assembler program. (This output is not suitable for use as input to the assembler.) Each line contains the location counter value and text in hexadecimal, the mnemonic symbol for the operation code, and a variable field in assembler language format. Source program identifiers are used wherever possible, and comments are inserted to assist in correlation with the source program. When the LIST option is specified, a STATIC INTERNAL storage map is also printed. The NOLIST option indicates that no object program listing or STATIC INTERNAL storage map is to be provided. If no option is specified, the standard default, NOLIST, is assumed by the compiler. A description of the object program listing is given in the section called "Compiler Output."

OPLIST or NOOPLIST - the OPLIST option (the standard default) causes the list of compiler options to be printed. The NOOPLIST option provides that no such list shall be printed.

EXTREF or NOEXTREF (External Listing) - the EXTREF option causes a listing of the external symbol dictionary (ESD) to be provided. The NOEXTREF option specifies that no listing is required and is the standard default assumed if no option is specified. A description of the ESD listing is given in the section called "Compiler Output."

ATR or NOATR (Attribute Table) - the ATR option gives:

1. A table showing, for each identifier:

   Statement number of the statement in which the identifier is declared

   Identifier and all containing structures

   A list of attributes pertaining to the identifier

2. A listing showing the length, in bytes, of every aggregate (array or structure) variable declared or allocated with fixed extents within the program.

The list of identifiers is sorted, ignoring the qualification. The NOATR option specifies that no listing is required and is the standard default assumed if no option is specified. A description of the attribute listing is given in the section called "Compiler Output."

XREF or NOXREF (Cross Reference Table) - the XREF option produces output showing, for each identifier:

Statement number of the statement in which the identifier is declared

Identifier and all containing structures

List of all statements in which a reference is made to the identifier

The list of identifiers is sorted, ignoring the qualification. The NOXREF option indicates that no cross reference listing is required, and is

the standard default assumed if no option is specified. A description of the listing is given in the section called "Printed Listings."

DECK or NODECK (Object Program Deck) - the DECK option specifies that the compiled program (i.e. the object module) is to be output in the form of a sequenced card deck in linkage editor format. The data set is defined by a SYSPUNCH DD statement. No job control statements, (DD cards, etc.) are generated by the compiler. NODECK specifies that no object program deck is required; if NODECK is specified, the DD statement SYSPUNCH is not required. If no option is specified, the standard default, NODECK, is assumed. A description of the deck is given in the section called "Compiler Output."

LOAD or NOLOAD (Object Program Output) - the LOAD option causes object program output to be produced as a sequential data set defined by a SYSLIN DD statement in standard linkage editor input format. NOLOAD specifies that no load module is to be created. LOAD is assumed as the standard default if no option is specified. If NOLOAD is specified, the DD statement SYSLIN is not required.

OBJNM=aaaaaaaa - this option causes the compiler to produce the linkage editor control statement NAME following either the object module, the object program deck, or both. If neither LOAD nor DECK has been specified (explicitly or by default), the use of this option will have no effect. The NAME statement produced will be of the following form:

posn.1

bNAMEbaaaaaaaa(R)

where aaaaaaaa represents the name given, which may have up to eight characters, and b represents one or more blanks.

Care should be taken when using this option in conjunction with cataloged procedures, otherwise the program name used in the EXEC statement will be incorrect. The principal purpose of this option is to enable the user to create a partitioned data set of load modules from a series of batched compilations, by means of the linkage editor.

OPT=nn (Optimization) - two alternative levels of optimization are available to the user. The decimal integer constant (nn) indicates the level required. The choice depends upon whether the prime concern of the user is execution speed or object-time storage space.

OPT=00: in this case, the object-time storage requirement is kept to a minimum.

OPT=01: in this case, the execution speed is improved at the expense of object-time storage space.

The difference in the compilation time for the two options is negligible. The standard default assumption if neither is specified is OPT=01.

BCD or EBCDIC Source Code - the BCD or EBCDIC option allows the programmer to state in which character code his source program is punched. The BCD and EBCDIC punched card codes and graphics for the PL/I 60-character set are shown in Figure 4. If neither option is specified, the standard default assumption is EBCDIC.

SORMGIN=(mmm,nnn[,ccc]) - the source margin option allows the programmer to specify:

mmm,nnn - margin for source statements

ccc - carriage control character position (optional)

The values for these arguments are subject to the constraints:

1. mmm ≤ nnn ≤ 100

2. ccc must be outside the range (mmm, nnn).

The carriage control characters are:

- b Skip one line before printing
- 0 Skip two lines before printing
- - Skip three lines before printing
- + Suppress space before printing
- 1 Skip to Channel 1 (i.e. start new page)

If the source margin is not specified, a standard default of SORMGIN=(2,72) is assumed. The compile-time processor produces output requiring margins (2,72) and changes SORMGIN accordingly. The default value for the carriage control character is installation-defined, that is, the user must decide at system generation whether a default is required and what it is to be.

Punched Card Code	BCD		EBCDIC		Punched Card Code	BCD		EBCDIC	
	Graphic	PL/I	Graphic	PL/I		Graphic	PL/I	Graphic	PL/I
No punches					12-7	G	G	G	G
12-8-3	.	.	.	.	12-8	H	H	H	H
12-8-4	)	)	<	<	12-9	I	I	I	I
12-8-5	[	%	(	(	11-1	J	J	J	J
12-8-6	<	<	+	+	11-2	K	K	K	K
12-8-7	¦	¦	¦	¦	11-3	L	L	L	L
12	& +	+	&	&	11-4	M	M	M	M
11-8-3	$	$	$	$	11-5	N	N	N	N
11-8-4	*	*	*	*	11-6	O	O	O	O
11-8-5	]	&	)	)	11-7	P	P	P	P
11-8-6	;	;	;	;	11-8	Q	Q	Q	Q
11-8-7	Δ	¬	¬	¬	11-9	R	R	R	R
11	-	-	-	-	0-2	S	S	S	S
0-1	/	/	/	/	0-3	T	T	T	T
0-8-3	,	,	,	,	0-4	U	U	U	U
0-8-4	% (	(	%	%	0-5	V	V	V	V
0-8-5	¬				0-6	W	W	W	W
0-8-6	\	#	>	>	0-7	X	X	X	X
0-8-7	¢		?	?	0-8	Y	Y	Y	Y
8-2	b	@	:	:	0-9	Z	Z	Z	Z
8-3	# =	=	#	#	0	0	0	0	0
8-4	@ '	'	@	@	1	1	1	1	1
8-5	:	:	'	'	2	2	2	2	2
8-6	>	>	=	=	3	3	3	3	3
12-0	?	?			4	4	4	4	4
12-1	A	A	A	A	5	5	5	5	5
12-2	B	B	B	B	6	6	6	6	6
12-3	C	C	C	C	7	7	7	7	7
12-4	D	D	D	D	8	8	8	8	8
12-5	E	E	E	E	9	9	9	9	9
12-6	F	F	F	F					

Figure 4. BCD and EBCDIC Punched Card Codes and Graphics for PL/I 60-character Set

CHAR60 or CHAR48 - this option allows the source language to be written in one of two character sets: 60- or 48-characters. The standard default assumed if no character set is specified is the 60-character set. Note that all characters in the 60-character set are acceptable to the compiler, even when the CHAR48 option has been selected. However, there are two points to be borne in mind by anyone intending to use 60-character set input after selecting the CHAR48 option. Firstly, the restrictions of the 48-character set must be observed in the 60-character set input, otherwise errors may result. Secondly, such a mode of operation is inefficient, and should not, therefore, be made a general practice.

STMT or NOSTMT - the STMT option specifies to the compiler that extra code is to be produced which will allow diagnostic messages printed during execution of the compiled program to contain statement numbers from the source program, in addition to offsets relative to PL/I entry points; these offsets alone are given if NOSTMT is specified. The standard default assumed if no option is specified is NOSTMT.

Note: The average PL/I program can be executed without significant degradation in speed or space requirements. However, the use of the STMT option will cause both the execution time and the object-code space requirements to increase in direct proportion to the number of statements in the program. The benefits obtained as a result of the debugging aid that this option gives, should offset and justify these increases.

FLAGW|FLAGE|FLAGS - a high level of diagnostic capability is available in the (F) compiler. Compile-time processor diagnostic messages are written after the SOURCE2 listing and before the source program listing. All other diagnostic messages are written in a

group following the source program listing. In both cases, messages are listed in order of their severity. There are four classes of diagnostic messages, which are graded in order of severity:

Type of Message	Specification
warning (lowest)	FLAGW
error	FLAGE
severe error	FLAGS
terminal error (highest)	

A Warning is a message that calls attention to a possible error, although the statement to which it refers is syntactically valid. In addition to alerting the programmer, it may assist him in writing more efficient programs in the future.

An Error message describes an attempt to correct an erroneous statement; the programmer is informed of the correction. Errors do not normally terminate processing of the text.

A Severe error message indicates an error which cannot be corrected by the compiler. The incorrect section of the program is deleted, but compilation is continued. Where reasonable, the ERROR condition will be raised at object time, if execution of an incorrect source statement is attempted. If a severe error occurs during compile-time processing, compilation will be terminated after the SOURCE listing has been produced.

A Terminal error message describes an error which, when discovered, forces the termination of the compilation.

The choice of the severity level at and above which diagnostic messages appear on the output is an option which may be selected by the programmer. FLAGW is assumed if no level is specified.

A source program message consists of the (F) compiler's identification code (IEM) followed by a number which is unique to this message. Where applicable, the statement number to which the message refers is given next. The message then follows. It will normally refer to the number of the statement which has produced the diagnostic message with the number corresponding to that on the source program listing. It may also contain a numeric parameter, a source program identifier, or a segment of source program text. During compile-time processing, messages will contain input line numbers, source text identifiers, or segments of source text.

Example:

```
IEM0096I 23 SEMI-COLON NOT FOUND WHEN
           EXPECTED IN STATEMENT NUMBER
           23. ONE HAS BEEN INSERTED.
```

A detailed list of diagnostic messages is given in Appendix G of this publication.

MACRO or NOMACRO - the MACRO option indicates that compile-time processing is required. The NOMACRO option causes the compile-time processor to be bypassed. If no option is specified, the compiler assumes the standard default NOMACRO.

SOURCE2 or NOSOURCE2 - the SOURCE2 option indicates that the input to the compile-time processor is to be listed on the device indicated on the SYSPRINT DD card. The SOURCE2 listing is in the same character set as the input text. Each line is preceded by a line number. NOSOURCE2 suppresses the listing of compile-time processor input. If no option is specified, the compiler assumes the standard default SOURCE2. If MACRO has not been specified, the SOURCE2 option is ignored.

COMP or NOCOMP - the COMP option specifies that compilation is to proceed after compile-time processing has been completed. NOCOMP specifies that compilation is not required. If neither option is specified, the compiler assumes the standard default COMP. If MACRO has not been specified, COMP|NOCOMP is ignored.

EXTDIC or NOEXTDIC - The EXTDIC option provides a compiler dictionary with a capacity 1.5 times that of the normal dictionary if the block size is 1K bytes, and 3.5 times that of the normal dictionary if the block size is greater than 1K. This allows successful compilation of some larger programs that would otherwise overflow the dictionary capacity. As the use of EXTDIC reduces compilation speed, it should be specified only when the program will not compile with the normal dictionary. The default is NOEXTDIC.

Programs that are large enough to require the EXTDIC option will compile very much more efficiently if a large storage size is available to the compiler. Enough storage should be specified, if possible, for the dictionary to be held in storage throughout the compilation. As a rough rule, the SIZE option should allow about 100,000 bytes plus 75 times the number of identifiers in the source program.

EXTDIC should not be used if SIZE<47104 bytes.

NEST or NONEST - If NEST is specified, a number is printed under the heading LEVEL showing the depth of block-nesting for the first statement on each line of the source program listing, and a number is printed under the heading NEST showing the depth of DO-group nesting for such statements which are contained within DO groups.

MACDCK or NOMACDCK - The MACDCK option causes the output records of the macro processor to be written on SYSPUNCH as sequenced card images. The default is NOMACDCK.

M91 or NOM91 - the M91 option must be specified when the object program is to be executed on an IBM System/360 Model 91. This option may be specified in the PARM field of the EXEC card at compilation time or it may be set during system generation time. The standard default for this option is NOM91.

COMPLETION CODES FOR THE COMPILER

At the end of compilation, the compiler returns a completion code to the operating system indicating the degree of success achieved in compilation of the source program and anticipated in execution of the object program. Tests of this code may be requested on the JOB and EXEC cards of the job control language, allowing later steps of a job to be suppressed if compilation has not achieved the required degree of success. The compiler completion codes are:

Code	Meaning
0	No diagnostic messages issued; compilation completed with no errors; successful execution expected
4	Warning messages only issued; program compiled; successful execution is probable
8	Error messages issued; compilation completed, but with errors; execution may fail
12	Severe error messages issued; compilation may be completed, but with errors; successful execution improbable. If a severe error occurs during compile-time processing, the compilation will be terminated and, if the SOURCE option has been specified, a listing of the PL/I program text produced by the compile-time processor will be printed
16	Terminal error messages issued; compilation terminated abnormally; successful execution impossible

JOB CONTROL PROCEDURE FOR COMPILATION

An example of a job control procedure showing the job control cards and deck required for a compilation is shown below.

```
//JOB1    JOB  123,JOHNSMITH,
               MSGLEVEL=1

//STEP1   EXEC PGM=IEMAA,PARM=
               'options'

//SYSPUNCH DD  SYSOUT=B

//SYSPRINT DD  SYSOUT=A

//SYSUT1   DD  UNIT=SYSDA,
               SPACE=(1024,(60,60)),
               SEP=(SYSPRINT,
               SYSPUNCH)

//SYSIN    DD  *

(Source Program Deck)

/*
```

Batched Compilation

A facility is available which allows more than one external procedure to be compiled within the same job step. This is achieved by preceding the second (and each subsequent) compilation by a control record of the following form:

```
 * PROCESS('options');
```

The first character of the record is an asterisk, which must appear in column 1. It denotes the end of the preceding source deck. Note that if SORMGIN=(1,72) has been specified, this facility is obtained only when there is an asterisk in column 1 and the first non-blank character string is PROCESS.

Any number of spaces, or no space, is acceptable between the asterisk and the keyword PROCESS. Spaces are permitted between:

1. The keyword PROCESS and the option list delimiter

2. The option list delimiter and the options

3. The option list delimiter and the semi-colon

Options in the list are separated by commas, as in the PARM field on the EXEC card.

The options to be listed are the compiler options to be used in compiling the source deck that follows. If any option is not listed, the installation default is assumed; there is no carry-over of the options listed on the preceding EXEC card. If the user does not wish to specify any options explicitly, he can simply place the terminating semicolon immediately after the word PROCESS. The number of characters in the option list is limited only by the length of the record, and may exceed 40 characters.

The length of the control record is limited to the length of the SYSIN record. The record must be punched in EBCDIC.

The return code given for the job step is the highest code from those which the individual compilations would return.

The advantages of batched compilation are a reduction of control program overheads, and a reduction in the number of cards to be punched. The disadvantage is that a terminal error in one compilation may result in termination of the whole batch.

The following example shows the job control statements required (in conjunction with cataloged procedure PL1LFCL): to batch compile three separate source programs; to link-edit source program 1 as one load module, source programs 2 and 3 together as a second load module, and to put both load modules into an existing cataloged data set; and to execute both load modules.

```
//BTCHCP      JOB  PL1PROG,MSGLEVEL=1
//STPCMP      EXEC PROC=PL1LFCL,
                   PARM.PL1L='LOAD,
                   NODECK,OBJNM=BATCHA'
//PL1L.SYSIN   DD  *
          source program one
* PROCESS('LOAD,ATR,NODECK');
          source program two
* PROCESS('LOAD,LIST,ATR,NODECK,
   OBJNM=BATCHB');
          source program three
/*
//LKED.SYSLMOD DD  DSNAME=PL1.PROGRAMS,
                   DISP=OLD
//BTCHEX      JOB  PL1PROG,MSGLEVEL=1
//JOBLIB       DD  DSNAME=PL1.PROGRAMS,
                   DISP=(OLD,PASS)
//STPEXA      EXEC PGM=BATCHA
//SYSPRINT     DD  SYSOUT=A
//STEPEXB     EXEC PGM=BATCHB
//SYSPRINT     DD  SYSOUT=A
```

COMPILER OUTPUT

The compiler can generate listings of source statements, compile-time statements, attributes, cross references, External Symbol Dictionary entries, aggregate lengths and object code. Source program and compile-time diagnostic messages are also produced during compilation.

PRINTED LISTINGS

The first page has the following heading: VERSION x RELEASE zz on the left-hand side of the page (where x indicates the version of the compiler, and zz the System/360 operating system release number); OS/360 PL/I COMPILER (F) in the center of the page; and DATE yy.ddd on the right-hand side of the page (where yy=year and ddd=day). Page numbering, which appears at the top right-hand corner of each page, begins on this page. Following the heading is the invocation parameter list containing the options specified for

the compilation. The diagnostic messages, if any, associated with the specification of compiler options appear next.

If the compiler options specified include OPLIST, these diagnostic messages are followed by a complete list of the compiler options used for the compilation. This list includes all options validly specified at the time of invocation, as well as those options assumed by default.

Each page thereafter has a heading and a page number. If the compiler is invoked by an EXEC statement, the heading used is the first record of PL/I source text read from the input source (SYSIN).

The listings consist of the following items in the order given; items 1 through 10 depend on the appropriate compiler options being selected and specified on the EXEC statement card; items 5 through 9 are not produced if a terminal error halts the compilation.

1. List of compiler options specified
2. Compile-time processor input listing
3. Compile-time processor diagnostic messages
4. Source program listing
5. Attribute and/or cross reference table
6. Aggregate length table
7. Storage requirements listing
8. External symbol dictionary listing
9. Assembly listing
10. Source program diagnostic messages

The time taken for the compilation is also shown on the listing.

Compiler Options

The compiler options listed are those used for the particular compilation. A description of the options is given in the section called "Compiler Processing."

Compile-Time Processor Input Listing

If the MACRO and SOURCE2 options are specified, the input to the compile-time processor is written on the output device specified in the SYSPRINT DD statement, one input record per line. Each line is assigned a number which appears to the left of the line.

Example:

```
1 % DCL A CHAR, B FIXED;

2 % A='B+C'; % B=2;

3 X=A;
```

Source Program Listing

If the SOURCE option is specified, the source program is written on the output device specified in the SYSPRINT DD statement. The contents of one card are printed on one line; if a carriage control character is specified, these may be skips between lines or to a new page. The number of the first statement starting on that line is printed at the left-hand side of that line.

Example:

```
1 PGMNAME:

  PROCEDURE;

2 DECLARE    A STATIC,

3            B AUTOMATIC; A=B;
```

For numbering purposes, statements contained within compound statements (IF and ON) are counted, as well as the compound statements themselves. In addition, when an END statement closes multiple groups or blocks, all implied END statements are included in the count.

Example:

```
1  P:    PROC;

2  X:    BEGIN;

3  IF    A=B

4  THEN  A=1;

5  ELSE  DO;

6        A=0;

7        C=B;

8        END X;

10       D=E;

11       END;
```

If the input to the compiler is produced by the compile-time processor, the SORMGIN option will be (2,72) and columns 73-80 will contain auxiliary information. This information can be used by the programmer to determine how the input to the compiler was generated from the original input.

These columns, which are listed as part of the SOURCE listing, are used as follows:

COL	USE
73-77	Contains the input line number from which this output line was generated. These numbers correspond to the line numbers printed with the SOURCE2 listing.
78-79	Contains a two-digit number giving the maximum depth of replacement which occurred for this line. If no replacement occurred, this field will be blank.
80	If an error occurs while replacement is being attempted (e.g., a % is found in a replacement value), this column will contain the letter "E". Otherwise it will be blank.

If the NEST option is specified then:

1. the block level at the begining of the statement indicated is printed next to the statement number for that line.
2. an iterative DO level count is printed next to the level count.

Example:

```
STMT LEVEL NEST

  1              A: PROC OPTIONS (MAIN);
  2    1            B: PROC (L);
  3    2               DO I = 1 TO 10;
  4    2    1             DO J = 1 TO 10;
  5    2    2             END;
  6    2    1             BEGIN;
  7    3    1             END;
  8    2    1          END B;
  9    1          END A;
```

Storage Requirements

This listing is printed out whenever the SOURCE option is in effect and gives the following information in bytes:

Storage area for each procedure
Storage area for each begin block
Storage area for each on-unit
Length of program CSECT
Length of static CSECT

Attribute and Cross Reference Table

If the ATR option is specified, the identifiers used in the program are listed in alphameric order, together with their explicitly declared and default attributes.

The attributes INTERNAL, NORMAL, and REAL are not listed; these attributes can be assumed unless the conflicting attributes EXTERNAL, ABNORMAL, and COMPLEX are listed.

For file identifiers, only explicitly declared attributes are listed, except for EXTERNAL, which is listed even when applied by default.

For an array, the dimension attribute is printed first. The dimensions are printed as in the declaration except that expressions are replaced by asterisks. Constants and expressions for string lengths are represented in the length attribute in the same way as constants and expressions for array dimensions are represented in the dimension attribute.

If an identifier is declared in a DECLARE statement, it is preceded by the number of that statement. Statement labels and entry labels are preceded by the defining statement number. No statement number precedes a contextually declared identifier.

If the XREF option is specified in addition to the ATR option, each identifier has the numbers of all the statements in which it occurs printed immediately below its attributes.

If the XREF option is specified without the ATR option, the table is printed without the attributes.

Example:

```
  1                QQ:     PROCEDURE;
  2                        DCL 1 A, 2 B(3), 2 C, 3 D;
  3                        DCL G(10,10) CHAR(2);
  4                        DCL 1 E, 2 F LIKE C;
  5                        DCL I(10) CHAR(2) DEF G(1SUB,1SUB);
  6                        DCL 1 J, 2 K BIT(1), 2 L BIT (9);
  7                        DCL S(V);
  8                        DCL H(10,10) CHAR(2) DEF G;
  9                        DCL U(*) CONTROLLED;
 10                        DCL 1 M BASED(P), 2 N POINTER, 2 T;
 11                        DCL Y(2:4,II);
 12                        DCL X(2:II);
 13                        DCL W AREA;
 14                        DCL Z(II:JJ,2);

 15                        ALLOCATE M IN (W) SET (P);
 16                        GET LIST(P->T);
 17                        ALLOCATE M IN (W) SET (Q);
 18                        GET LIST(Q->T);
 19                        P->N=Q;
 20                        PUT LIST(Q->T);
 21                        R=4;
 22                        ALLOCATE U(R);
  .                           .
  .                           .
  .                           .
150                       END QQ;
```

The foregoing procedure will produce the following attribute and cross reference table.

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
2	A	AUTOMATIC,UNALIGNED,STRUCTURE
2	B	(3)IN A,AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)
2	C	IN A,AUTOMATIC,UNALIGNED,STRUCTURE
2	D	IN C IN A,AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)
4	D	IN F IN E,AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)
4	E	AUTOMATIC,UNALIGNED,STRUCTURE
4	F	IN E,AUTOMATIC,UNALIGNED,STRUCTURE
3	G	(10,10)AUTOMATIC,UNALIGNED,STRUCTURE(2),CHARACTER
8	H	(10,10)AUTOMATIC,DEFINED,UNALIGNED,STRUCTURE(2),CHARACTER
5	I	(10)AUTOMATIC,DEFINED,UNALIGNED,STRUCTURE(2),CHARACTER
	II	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 14,12,11
6	J	AUTOMATIC,UNALIGNED,STRUCTURE
	JJ	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 14
6	K	IN J,AUTOMATIC,UNALIGNED,STRUNG(1),BIT

```
6           L               IN J,AUTOMATIC,UNALIGNED,STRING(9),BIT

10          M               BASED(P),UNALIGNED,STRUCTURE
                            15,17

10          N               IN M,BASED(P),ALIGNED,POINTER
                            19

            P               AUTOMATIC,ALIGNED,POINTER
                            15,16,19

            Q               AUTOMATIC,ALIGNED,POINTER
                            17,18,19,20

1           QQ              ENTRY,DECIMAL,FLOAT(SINGLE)

            R               AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)
                            21,22

7           S               (*)AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)

            SYSIN           FILE,EXTERNAL
                            16,18

            SYSPRINT        FILE,EXTERNAL
                            20

10          T               IN M,BASED(P),ALIGNED,DECIMAL,FLOAT(SINGLE)
                            16,18,20

9           U               (*)CONTROLLED,ALIGNED,DECIMAL,FLOAT(SINGLE)
                            22

            V               AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)
                            7

13          W               AUTOMATIC,ALIGNED,AREA
                            15,17

12          X               (2:*)AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)

11          Y               (2:4,*)AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)

14          Z               (*:*,2)AUTOMATIC,ALIGNED,DECIMAL,FLOAT(SINGLE)
```

The attributes FLOAT(SINGLE) and FLOAT(DOUBLE) refer to the precision of the floating-point data. These precisions are defined in Figures 32, 49, and 49.1.

Aggregate Length Table

Specification of the ATR option or the XREF option produces an aggregate length table, which gives, where possible, the length in bytes of all major structures and non-structured arrays used in the program.

Each entry in the table consists of the identifier of an aggregate preceded by a statement number and followed by the length of the aggregate in bytes.

The statement number is the number either of the DECLARE statement for the aggregate or, in the case of a CONTROLLED, non-BASED aggregate, of an ALLOCATE statement for the aggregate. An entry appears for every ALLOCATE statement involving a non-BASED aggregate since such statements may have the effect of changing the aggregate length during execution. Allocation of a BASED aggregate does not have this effect, and only one entry, which is that corresponding to the DECLARE statement, appears for any BASED aggregate.

The length of an aggregate may not be known at compilation, either because the aggregate contains elements having adjustable lengths or dimensions, or because the aggregate is dynamically defined. In these cases, the words 'ADJUSTABLE' or 'DEFINED' appear in the 'Length in Bytes' column.

The entry for a COBOL mapped structure has the word '(COBOL)' appended.

The procedure in the foregoing attribute and cross reference table example produces the following aggregate length table.

AGGREGATE LENGTH TABLE

STATEMENT NO.	IDENTIFIER	LENGTH IN BYTES
2	A	16
4	E	4
3	G	200
8	H	200
5	I	DEFINED
6	J	2
10	M	8
7	S	ADJUSTABLE
22	U	ADJUSTABLE
12	X	ADJUSTABLE
11	Y	ADJUSTABLE
14	Z	ADJUSTABLE

External Symbol Dictionary Listing

If the option EXTREF is specified, all entries in the external symbol dictionary (ESD) are listed. The information appears under the following headings:

SYMBOL - an eight-character field containing the name identifying the ESD entry

TYPE - two characters from the following list, to identify the particular type of ESD entry:

Character	Type
SD	Section Definition
CM	Named Common
ER	External Reference
PR	Pseudo-register
LD	Label Definition

(For a detailed description of the different types of ESD entries, refer to the publication IBM System/360 Operating System, Linkage Editor.)

ID - a 4-digit hexadecimal number which identifies the symbol

ADDR - the hexadecimal representation of the compiled address of the symbol

LENGTH - the length in bytes of the section (applicable only to SD, CM, and PR type entries), expressed as a hexadecimal number

The sizes of the program and STATIC INTERNAL control sections are printed at the end of the ESD; they are also printed at the end of the compilation. This makes the sizes available even if EXTREF is not specified. The size of each DSA is also printed at the end of the compilation. Printing of these sizes is under the control of the SOURCE option.

Description of Contents of ESD Listing

The first seven entries are in standard format, as shown in Figure 5.

SYMBOL	TYPE	ID	ADDR	LENGTH
PGMNAME	SD	0001	000000	00033A
PGMNAMEA	SD	0002	000000	00005F
IHEQINV	PR	0003	000000	000004
IHESADA	ER	0004	000000	
IHESADB	ER	0005	000000	
IHEQERR	PR	0006	000000	000004
IHEQTIC	PR	0007	000000	000004

Figure 5. Specimen ESD Listing

The seven entries are:

1. Name of the program control section. This is the first label in the external procedure statement.
2. Name of the STATIC INTERNAL control section. This is the same label as in 1, (left-filled with asterisks if necessary) to seven characters, with an eighth character, A.
3. Pseudo-register for the invocation count (the count of the number of times a block is invoked recursively).
4. Entry point of the library routine used to obtain AUTOMATIC storage for a PL/I block.
5. Entry point of the library routine used to obtain storage for some AUTOMATIC variables.
6. Pseudo-register for the error handling routine.

7. Pseudo-register used for tasking.

The remainder of the listing is variable, but the general organization is as follows:

1. Section definition for one-word control section IHEMAIN, which contains the address of the principal entry point to the external procedure. This item is present only if the external procedure has the option MAIN.
   Section definition for the three-word control section IHENTRY (always present), which provides the linkage with the library at execution time.

2. LD type items for all names of entry points to the external procedure, except the first, which has been used as the name of the program control section.

3. A PR type entry for the display pseudo-register for each block in the compilation. The names are generated from the external procedure name which is left-filled to seven characters with asterisks. The eighth character is selected from a table of printable characters which are assigned to names in the order in which the blocks appear in the compilation.

   If each character in the table has been used once and more names are still required, the first character of the name is replaced from another (smaller) table and the eighth character recycles. The first and, if necessary, the second character of the name are allowed to cycle through this second table.

   If the first or second character overwritten in this way was a character in the external procedure name (as opposed to an asterisk used in padding), a warning message is printed out, since this could mean that separate external procedures with distinct names could contain identical generated names.

   Example:

```
X: PROC;
  Y: PROC;
    Z: BEGIN;
END X;
```

   The display pseudo-registers for X, Y, and Z would have the names:

```
******XB

******XC

******XD
```

4. ER type entries for all library routines and external routines called by the compilation. The list of names of library routines contains the names of routines which are directly called by the compiled code (first-level routines) and the names of modules which are called by the first-level routines.
   ER type entries for all library routines not directly called by the compiled code but required for successful compilation.

5. SD type entries for EXTERNAL FILE constants and STATIC EXTERNAL variables, except in the case of STATIC EXTERNAL variables without any initial text, which have CM type entries. File constants, both internal and external, have a pseudo-register associated with them, used by the library for addressing a dynamically allocated data set control block. For EXTERNAL FILE constants the pseudo-register name is the same as the file name, and for internal ones it is generated as for display pseudo-registers.

6. PR type entries for all CONTROLLED variables. For external variables the declared name is used, and for internal variables a name is generated as for the display pseudo-registers.

STATIC INTERNAL Storage Map

A STATIC INTERNAL storage map is produced only when the compiler option LIST is specified. The size of this storage area is given at the end of the compilation.

The first 52 bytes of the STATIC INTERNAL Control Section are of a standard form and are not listed. They contain the following information:

```
DC      F'4096'
DC      AL4(SI.+X'1000')
        .
        .
        .
DC      AL4(SI.+X'7000')
DC      VL4(IHESADA)
DC      VL4(IHESADB)
DC      A(DSASUB)
DC      A(EPISUB)
DC      A(IHESAFA)
```

SI. is the address of the STATIC INTERNAL control section, and IHESADA, IHESADB and IHESAFA are library routines. DSASUB and EPISUB are compiler routines for getting and freeing DSAs.

The remainder of STATIC is listed in the following form:

1. 6-character hexadecimal offset
2. Up to 8 bytes of hexadecimal text
3. Comment indicating type of item to which the text belongs (only appears against first line of text for item)

The possible comments are as follows (xxx indicates the presence of an identifier):

DED FOR TEMP or DED	Data Element Descriptor for temporary or for a programmer's variable
FED	Format Element Descriptor
DV..xxx	Dope Vector of a STATIC variable
DVD..	Dope Vector Descriptor
D.V. SKELETON	Dope Vector skeleton for an AUTOMATIC or CONTROLLED variable
RDV..	Record Dope Vector
A..xxx	Address of EXTERNAL control section or entry point, or of an internal label
ARGUMENT LIST	Argument list skeleton
CONSTANTS	
SYMTAB	Symbol table entry
SYM..xxx	Symbolic name of label constant or label variable
FILE..xxx	File name
ON..xxx	Programmer-declared ON-condition
ATTRIB	File attributes
xxx	STATIC variable. If the variable is not initialized, no text appears against the comment, and there is also no STATIC offset if the variable is an array. (This can be calculated from the dope vector if required)

If the LINECNT specification for the listing is 72 or less, and the actual number of lines to be printed (including skips) exceeds the LINECNT, double-column format is used. If LINECNT is greater than 72, or if the actual number of lines to be printed (including skips) is less than the LINECNT, single-column format is used.

A sample of a STATIC INTERNAL storage map is given in Appendix D.

Object Program Listing

If the LIST option is specified, a listing of the object code is produced, together with its equivalent in a form similar to assembler language. If the LINECNT specification for the listing is 72 or less, and the actual number of lines to be printed (including skips) exceeds the LINECNT, double-column format is used. If LINECNT is greater than 72, or if the actual number of lines to be printed (including skips) is less than the LINECNT, single-column format is used. Comments, of the form shown below, are included to help the programmer to identify the functions of different segments of the object program.

* STATEMENT NUMBER n - identifies the start of the code generated for source listing statement number n
* PROCEDURE xxx - identifies the start of the procedure labeled xxx
* REAL ENTRY xxx - heads the initialization code for an entry point to a procedure labeled xxx
* PROLOGUE BASE - identifies the start of the initialization code common to all entry points to that procedure
* PROCEDURE BASE - identifies the address loaded into the base register for the procedure
* APPARENT ENTRY xxx - identifies the point of entry into the procedure for the entry point labeled xxx
* END PROCEDURE xxx - identifies the end of the procedure labeled xxx
* BEGIN BLOCK xxx - indicates the start of the BEGIN block with label xxx
* END BLOCK xxx - indicates the end of the BEGIN block with label xxx
* INITIALIZATION CODE FOR xxx - indicates that the code following performs initial value assignment or dope vector initialization for the variable xxx

Wherever possible, a mnemonic prefix is used to identify the type of operand in an instruction, and where applicable this is followed by a source program identifier. The following prefixes are used:

A..	Address constant
AE..	Apparent entry point (point in the procedure to which control passed from the prologue)
BLOCK.	Label created for an otherwise unlabeled block (followed by the number of the block)
C..	Constant (followed by a hexadecimal dictionary reference)
CL.	A label generated by the compiler (followed by a decimal number identifying the label)
DED..	Data Element Descriptor
DV..	Dope Vector
DVD..	Dope Vector Descriptor
FVDED..	Data Element Descriptor of function value
FVR..	Function value
IC.	Invocation count pseudo register
ON..	ON-condition name
PR..	Pseudo-register
RDV..	Record Definition Vector
RSW..	Return switch
SI.	Address of STATIC INTERNAL control section
SKDV..	Skeleton dope vector, followed by hexadecimal dictionary reference
SKPL..	Skeleton parameter list, followed by hexadecimal dictionary reference
ST..	Symbol table entry
SYM..	Symbolic representation of a label
TCA..	Temporary Control Area. A word containing the address of the dope vector of the specified temporary
TMP..	Temporary, followed by hexadecimal dictionary reference
TMPDV..	Temporary dope vector, followed by hexadecimal dictionary reference
VO..	Virtual origin
WP1.	Workspace, followed by decimal number of block which allocates workspace
WP2.	Workspace, followed by decimal number of block which allocates workspace
WS1.	Workspace, followed by decimal number of block which allocates workspace
WS2.	Workspace, followed by decimal number of block which allocates workspace
WS3.	Workspace, followed by decimal number of block which allocates workspace

A listing of the various storage areas is not produced, but the addresses of variables can be deduced from the object program listing.

Example:

A=B+10E1; in the source program produces:

```
0002CA   78 00 B 058  LE    0,B

0002CE   7A 00 B 064  AE    0,C..08F4

0002D2   70 00 B 060  STE   0,A
```

A and B are STATIC INTERNAL variables at an offset of X'60' and X'58', respectively, from the start of the control section.

A sample of an object program listing is given in Appendix D. The size of this listing is always printed at the end of the compilation.

Diagnostic Messages

Diagnostic messages are produced whenever the compiler detects an error (or possible error) in the program and are sorted in order of severity. The four levels of severity and types of message are fully described in the section "Compiler Options."

Each error message is preceded by a code in the form IEMnnnnI, where IEM is the identification code for the PL/I (F) compiler, and nnnn is a decimal number identifying the message.

For a detailed list of the diagnostic messages refer to Appendix G of this publication.

Time of Compilation

At the end of a compilation, the time taken, in minutes and hundredths of minutes, is printed out, provided that the appropriate control-program timing options have been selected at the time of system generation. The time given is the time during which the compiler is actually in control. For details of these options, refer to 'System Requirements'.

OBJECT MODULE OUTPUT

The object module is in a form suitable for direct input into the linkage editor. It is made up as follows:

1. ESD entries for the object module.
2. TXT and RLD entries for the compiled program. TXT and RLD entries are intermingled, although RLD entries are always preceded by their related text.
3. TXT and RLD entries for the STATIC INTERNAL control section.
4. TXT and RLD entries for a one-word control section IHEMAIN, which contains the address of the principal entry point to the external procedure. This control section is produced only when the external procedure has the MAIN option; it is used by the library routines IHESAPA, IHESAPC, IHETSAP and IHETSAO when passing control to the program.
5. TXT and RLD entries for a three-word control section IHENTRY. This control section is referenced by the END card and contains a branch to one of the four library routines IHESAPA, IHESAPC, IHETSAP or IHETSAO. The last two routines are used for tasking.
6. TXT and RLD entries for all STATIC EXTERNAL control sections and EXTERNAL file constant control sections. The sections appear in increasing order of ESD identification.
7. END statement for the compilation. This specifies the IHENTRY control section as the one to be entered.
8. NAME statement for the compilation. This card appears only when the OBJNM option is specified.

Serialization of Object Decks

Object decks generated as a result of the use of the DECK option will be serialized in the following manner.

Columns 73 to 76 will contain the first four characters of the first entry label of the external procedure. If the label has fewer than four characters, it will be left-adjusted in the field and the remainder padded with blanks.

Columns 77 to 80 will contain a four-digit decimal serial number. Numbering starts at 0001 and the increment is 1 each time.

ABNORMAL END OF COMPILATION

If a program check occurs during a compilation, a diagnostic message is printed giving the type of interruption.

LINKAGE EDITOR PROCESSING

The linkage editor processes PL/I object modules, resolves any references to subroutines and functions, and forms a load module. To communicate with the linkage editor program, the programmer supplies an EXEC statement and DD statements that define all data sets. He may also supply control statements for the linkage editor. The program name for the linkage editor is IEWL. If the linkage editor is executed as a job step, the parameter PGM=IEWL is used in the EXEC statement.

Linkage Editor Input and Output

The primary input to the linkage editor is in the form of object modules and/or linkage editor control statements. (A load module, by itself, cannot be the primary input.) While processing an object module, the linkage editor finds any references to functions or subroutines in the object module, and uses an automatic library call to resolve them. The automatic call library consists of either object modules and linkage editor control statements, or load modules, but not both.

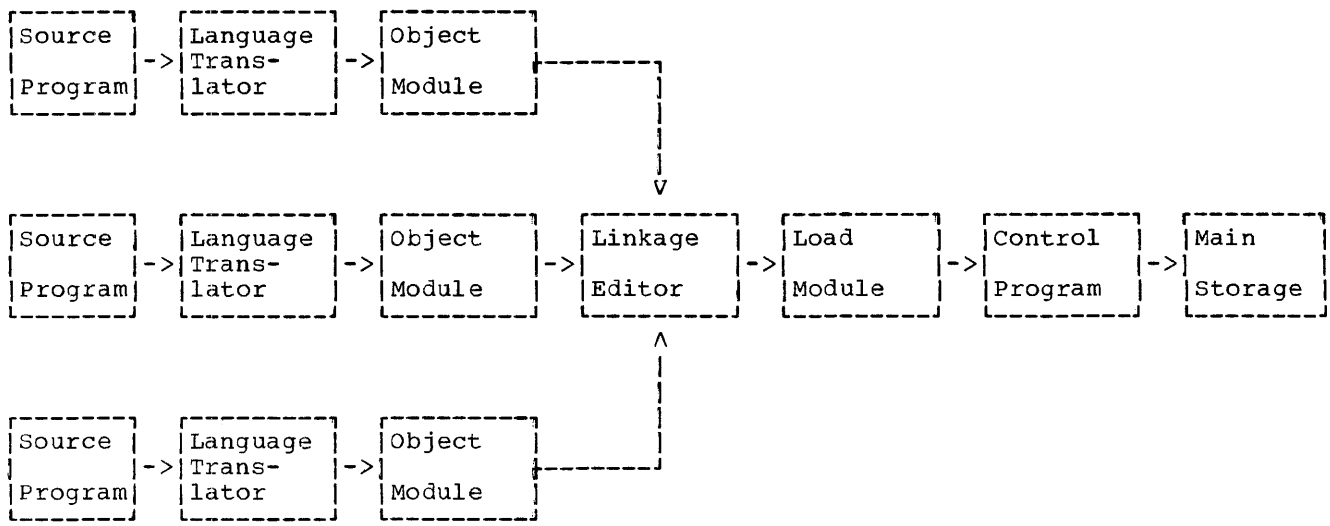


Figure 6. Linkage Editor Input/Output Flow

The programmer can specify additional libraries and data sets for use by the linkage editor; the LIBRARY card is used for this purpose (see the publication IBM System/360 Operating System, Linkage Editor.)

The output of the linkage editor is always placed in a library. Error messages and optional diagnostic messages are written on either an intermediate storage device or the printer. In addition, a work data set is required by the linkage editor to do its processing. Figure 6 shows the input/output flow in linkage editor processing.

Unit Names and ddnames

The programmer communicates information about data sets to the linkage editor through DD statements identified by specific ddnames, similar to the ddnames used by the compiler. The programmer must use these ddnames and he must meet the functions and requirements for data sets specified in Figure 7.

Any data sets defined by SYSLIB or SYSLMOD, and any additional data sets, must be partitioned data sets. The ddname given to refer to any additional libraries is not fixed by the linkage editor. Linkage editor control statements are used to inform the linkage editor about the additional libraries.

The data sets used by the linkage editor are assigned to the device classes listed in Figure 7.

Specifying Additional Libraries

The user can specify additional libraries by means of the linkage editor control statements INCLUDE and LIBRARY. These control statements can appear in either the input stream or the libraries. For details of these statements refer to the publication IBM System/360 Operating System, Linkage Editor.

Link-Editing a PL/I Subroutine into a Private Library

Space can be conserved by link-editing PL/I subroutines into a private library with their external references unresolved. To do this, the linkage editor option NCAL is specified; the SYSLIB DD statement is then unnecessary. It is also possible to maintain a library of PL/I object modules.

Either of these techniques requires considerably less disk space than link-editing all PL/I object modules with their external references resolved.

Linkage Editor Priority

If modules with the same name appear in the input to the linkage editor, the linkage editor accepts the first module which appears.

ddname	Function	Possible Device Classes
SYSLIN	Primary input data, normally the output of the compiler	SYSSQ, or the input stream device (specified by DD *)
SYSLIB	Automatic call library (usually the PL/I Library)	SYSDA
SYSUT1	Work data set	SYSDA
SYSPRINT	Diagnostic messages	SYSSQ, or SYSOUT device
SYSLMOD	Output for load module	SYSDA
(user-specified)	Additional libraries and data sets	SYSDA

Figure 7. Linkage Editor ddnames

Overlaying of PL/I Programs

In order to overlay PL/I programs, the user must have some knowledge of the control sections produced by the compiler. The following control sections are produced for each external procedure:

One control section containing the executable portion of the program and all constants.

One control section containing STATIC INTERNAL data.

One control section for each STATIC EXTERNAL item or EXTERNAL file name. (This is a named common control section only for a non-string scalar variable declared without the INITIAL attribute.)

One control section IHEMAIN (only if OPTIONS(MAIN) is specified for the procedure).

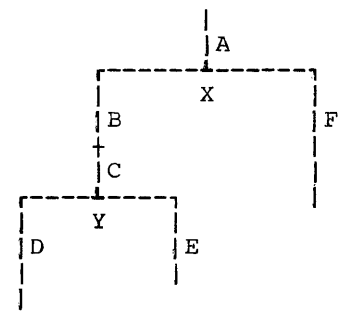
One control section IHENTRY.

There is a linkage editor requirement that each segment of an overlay structure consists of one or more complete control sections. No control section can be split between different segments.

The input to the linkage editor must contain all external procedures required in the overlay structure, together with the linkage-editor control cards describing the required structure. The external procedures may be in the form of:

1. Object modules in the input stream
2. Object and load modules in a library. These modules are called by means of the linkage-editor INCLUDE statement.

For a full description of the overlay facilities, see IBM System/360 Operating System: Linkage Editor. The two examples given below show how the simple overlay structure illustrated in the diagram can be produced from each of the forms of external procedure described above. Both examples use cataloged procedures; the first one also uses batched compilation methods.



Example 1. Object Modules in the Input Stream: The cataloged procedure PL1LFCLG is used, with two additional DD statements for SYSIN in the compilation and link-edit job steps.

```
//jobname    JOB   etc.
//stepname   EXEC  PL1LFCLG,
                      PARM.LKED='OVLY'
//PL1L.SYSIN  DD   *

     A:PROC OPTIONS(MAIN); ... END A;
*      PROCESS...
     B:PROC; ... END B;
*      PROCESS...
```

```
     C:PROC; ... END C;
*      PROCESS...
     D:PROC; ... END D;
*      PROCESS...
     E:PROC; ... END E;
*      PROCESS...
     F:PROC; ... END F;
/*
//LKED.SYSIN  DD   *
       INSERT A
       OVERLAY X
       INSERT B
       INSERT C
       OVERLAY Y
       INSERT D
       OVERLAY Y
       INSERT E
       OVERLAY X
       INSERT F
/*
```

Note that the DD statement LKED.SYSIN allows data in the input stream to be concatenated with the object file.

Example 2. Object and Load Modules in a Library: The cataloged procedure PL1LFCL is used to compile source modules in a library LIB.S and to use the object modules from this as input to the link-edit step. The load modules produced are placed in a library of load modules, LIB.L. Both LIB.S and LIB.L are cataloged data sets. Each external procedure requires:

```
//stepname    EXEC    PL1LFCL,PARM.LKED=
                      'XREF,LIST,NCAL'
//PL1L.SYSIN    DD    DSNAME=LIB.S(member),
                      DISP=SHR
//LKED.SYSLMOD DD     DSNAME=LIB.L(member),
                      DISP=OLD
```

To create an overlay structure from modules in LIB.L, the cataloged procedure PL1LFLG is used with the INCLUDE statement:

```
//stepname    EXEC    PL1LFLG,PARM.LKED=
                      OVLY
//LKED.PRIVLIB DD     DSNAME=LIB.L,
                      DISP=SHR
//LKED.SYSIN    DD    *
       INCLUDE PRIVLIB(A,B,C,D,E,F)
       INSERT A
       OVERLAY X
       INSERT B
       INSERT C
       OVERLAY Y
       INSERT D
       OVERLAY Y
       INSERT E
       OVERLAY X
       INSERT F
       ENTRY IHENTRY
/*
```

Use of the NCAL option (see below) ensures that only those modules required are placed in the load module library. The LKED.SYSLMOD DD statement overrides the DD statement with the same ddname in the cataloged procedure. The other DD statements are additions to those in the cataloged procedure.

The basis of the above method is that all object and load modules appear in the input stream before the first OVERLAY statement. All control sections are therefore placed in the root segment until explicitly moved to another segment by an INSERT statement. Each INSERT statement moves into the appropriate overlay segment, only the control section containing executable code.

Since all modules required from the PL/I library are obtained using the automatic library call procedure to resolve external references, all library modules and data are in the root segment, where they are available to all other segments. This is basically necessary, since PL/I does not support exclusive calls, i.e., calls between segments which do not lie on a single path. For example, in the above structure, procedures in the segment containing D could call procedures in the segments containing A, B, C, and D, but not in the segments containing E or F. Procedures in the segments containing B and C could call procedures in the segments containing A, B, C, D and E, but not in the segment containing F. A procedure in, say, the segment containing B may not call a procedure in the segment containing A which in turn calls a procedure in the segment containing F.

However, certain modules may not be required by all segments, in which case they can be moved out into a lower segment. The method of doing this is to compile the procedures using the compiler option EXTREF and then to examine the lists of external references produced. For example, if in the above structure a module IHExxx is called only by the segment containing E, it may be moved into that segment by a control statement of the form INSERT IHExxx immediately following the statement INSERT E. Similarly, if module IHEyyy is called only by the segments containing D and E it may be moved into the segment containing B or C. Any module called by both the segment containing F and any one or more of the segments containing B, C, D, and E must remain in the root segment.

Similarly, data control sections may be moved from the root structure to segments lower in the structure. For example:

1. To move the STATIC INTERNAL control section for procedure F into the segment containing F, the statement

INSERT F must be followed by INSERT ******FA.

2. To move the STATIC EXTERNAL control section PQR (which is referenced by B and C only) into the segment containing B and C, the statement INSERT C must be followed by INSERT PQR.

Values assigned to these STATIC data items are not saved when the segment is overlaid; when the segment is reloaded the original values are restored. STATIC data should therefore be left in the root segment unless it consists of:

1. Values set by the INITIAL attribute, and then unchanged (that is, read-only)

2. Values that need not be retained between different loadings of the segment.

An alternative method of producing an overlay structure is to obtain object decks and intermingle these with OVERLAY cards. This method requires much more careful handling. STATIC INTERNAL control sections must be moved up into the root segment, unless they are read-only. Named common control sections (which are produced only for non-initialized, non-string EXTERNAL scalar variables) are automatically moved up so that they are accessible to all segments requiring them, but all other EXTERNAL items will need to be moved up by the use of INSERT statements.

Further information on the use of overlay structures can be obtained from the publication IBM System/360 Operating System, Linkage Editor.

Other Linkage Editor Facilities

In addition to those already discussed, other linkage editor control statements are available:

1. NAME statement: Specifies the name of the load module created from the preceding input modules.

2. ALIAS statement: Specifies alternative names for the load module and can also specify alternative entry points.

3. CHANGE statement: Changes a control section name, an entry name, or an external reference.

4. REPLACE statement: Deletes a control section, an entry name, or an external reference; or deletes a control section to be replaced by another; or changes an entry name or an external reference.

For a detailed description of these facilities, see the publication IBM System/360 Operating System, Linkage Editor.

OPTIONS FOR LINKAGE EDITOR PROCESSING

The linkage editor options can be specified in the EXEC statement. The options that are most applicable to the PL/I programmer are:

```
PARM='LIST,MAP|XREF,LET|XCAL,NCAL,OVLY'
```

LIST - the LIST option specifies that all control statements processed by the linkage editor are listed in card-image format on the diagnostic output data set.

MAP or XREF - the MAP option tells the linkage editor to produce a map of the load module; this map indicates the relative location and length of control sections. If XREF is specified, a map of the load module is produced and a cross reference list indicating all external references in each control section is generated. If neither MAP nor XREF is specified, neither the map nor the cross reference listing is generated.

LET or XCAL - the LET option specifies that the linkage editor should mark the load module ready for execution even though certain error or abnormal conditions were found during linkage editing. (It should be noted that, because of the selective loading techniques, certain external references to library modules may not be resolved.) The XCAL option marks the load module executable even though improper branches were made between control sections; this occurs only if the OVERLAY feature of the linkage editor is being used.

NCAL - the NCAL option informs the Linkage Editor that no external references are to be resolved by the automatic library call mechanism. The subprograms in the library are not inserted in the load module. However, the load module is marked executable.

Other options can also be specified for the linkage editor. For a detailed description of all linkage editor options, see the publication IBM System/360 Operating System, Linkage Editor.

JOB CONTROL PROCEDURE FOR LINKAGE EDITOR

An example of a linkage editor procedure that handles the object program deck from the PL/I (F) compiler is given below.

```
//JOB2     JOB 345,JOHNSMITH,
              MSGLEVEL=1

//LINK     EXEC PGM=IEWL,
              PARM='XREF,LIST'

//SYSPRINT DD SYSOUT=A

//SYSLIB   DD DSNAME=SYS1.PL1LIB,
              DISP=SHR

//SYSUT1   DD UNIT=SYSDA,
              SPACE=(1024,(200,20))

//SYSLMOD  DD DSNAME=PRIVATE(PROGRAM),
              DISP=OLD

//SYSLIN   DD *

(Object program deck)

/*
```

LINKAGE EDITOR OUTPUT

The linkage editor produces a map of the load module if the MAP option is specified, or a cross reference list and a map if the XREF option is specified. The linkage editor also produces diagnostic messages, which are fully described in the publication IBM System/360 Operating System, Linkage Editor.

Size of Load Module

The size of the load module produced by the linkage editor must not exceed 512K bytes. Programs producing a load module larger than this should be segmented.

Module Map

The module map shows all control sections in the output module and all entry names in each control section. The control sections are arranged in ascending order according to their assigned origins. All entry names are listed below the control section in which they are defined.

If the module is in an overlay structure, the control sections are arranged by segment in the order in which they were specified.

Each control section that is included from a library during automatic library call is indicated by an asterisk.

After the control sections, the module map shows the contents of the pseudo-register vector (PRV).

At the end of the module map is:

1. The relative address of the instruction with which processing of the module begins.
2. The total length of the module in bytes
3. The total length of the pseudo-register vector.

In the case of an overlay load module, the length is that of the longest path. The addresses shown in the module map are those assigned by the linkage editor prior to loading for execution.

Cross Reference Table

The cross reference table consists of a module map and a list of cross references for each control section. Each address constant is listed with its assigned location, the symbol referred to, and the name of the control section in which the symbol is defined. For overlay programs, this information is provided for each segment. In addition, the number of the segment in which the symbol is defined is provided.

If the program is in an overlay structure, a list of cross references is provided for each segment.

If a symbol is unresolved after processing by the linkage editor, it is identified by $UNRESOLVED in the list. However, if an unresolved symbol is marked by the never-call (NCAL) function, it is identified by $NEVER-CALL.

Disposition Data

Information indicating the options and attributes specified is provided with each load module produced.

There are, in addition, a number of disposition messages to indicate the conditions of the load module in the output module library. For full details see the publication IBM System/360 Operating System, Linkage Editor.

DIAGNOSTIC MESSAGES

Certain conditions that are present when a module is being processed can cause an error or warning message to be printed. An error or warning message consists of a message code and a message text.

The message text contains combinations of the following:

The code IEW, which identifies linkage editor messages

The message classification (either error or warning)

Cause of the error

Identification of the symbol, segment number (when in overlay), or member in error

Instructions to the programmer

Actions taken by the linkage editor

If an error is encountered during processing, the message code for that error is printed with the applicable symbol, symbols, or record in error. After processing has been completed, the diagnostic message associated with that message code is printed.

LOAD MODULE EXECUTION

Execution Device Specification

When the system is generated, device names are assigned by the operating system and the installation, and the programmer chooses devices by specifying either the installation or operating system names.

The programmer can choose which device to use for his data sets and can specify the name of the device or class of devices in the UNIT parameter of the DD statement.

DCB Parameter

The data control block (DCB) parameter in the DD statement is used to specify record formats, record lengths, blocking, and other information regarding the uses of the data set. It may be used for data sets when a load module is executed. For information concerning the subparameters in the DCB used by PL/I load modules and assumptions made if the DCB parameter is not specified, see the section "Managing Data."

JOB CONTROL PROCEDURE FOR COMPILING, PROCESSING BY LINKAGE EDITOR, AND EXECUTING

An example follows of the card deck required for compiling, processing by the linkage editor, and executing a PL/I program:

```
//CLG      JOB  789,JOHNSMITH,
                MSGLEVEL=1

//STEP1    EXEC PGM=IEMAA,
                PARM='Options'

//SYSPRINT DD   SYSOUT=A

//SYSLIN   DD   DSNAME=&&LOADSET,
                UNIT=SYSDA,
                DISP=(MOD,PASS),
                SPACE=(80,(250,100))

//SYSUT1   DD   UNIT=SYSDA,
                SPACE=(1024,(60,60)),
                SEP=(SYSPRINT,
                SYSLIN)

//SYSIN    DD   *

(Source Program Deck)

/*

//LKED     EXEC PGM=IEWL

//SYSPRINT DD   SYSOUT=A

//SYSLMOD  DD   DSNAME=&&LOADSET,
                SPACE=(1024,(50,20,1)),
                DISP=(NEW,PASS),
                UNIT=SYSDA

//SYSLIB   DD   DSNAME=SYS1.PL1LIB,
                DISP=SHR

//SYSUT1   DD   UNIT=SYSDA,
                SEP=(SYSLMOD,SYSLIB),
                SPACE=(1024,(200,20))

//SYSLIN   DD   DSNAME=*.STEP1.SYSLIN,
                DISP=(OLD,DELETE)
```

```
//STEP3  EXEC  PGM=*.LKED.SYSLMOD,
               COND=((5,LT,STEP1),
               (5,LT,LKED))

//SYSPRINT DD  SYSOUT=A

//SYSIN    DD  *

(Data)

/*
```

OBJECT PROGRAM OUTPUT

Stream-Oriented Output

Figure 9 illustrates various forms of external data representation as produced by PUT statements employing the EDIT, LIST, and DATA keywords. It should be noted that the output of data fields to PRINT files by list- or data- directed techniques will automatically be aligned upon preset tab positions, the initial data field of a line being aligned upon the left margin, position 1. The preset tab positions are: 25, 49, 73, 97, and 121. (See the section "The Tab Control Table" for details of how to alter these values.) A numeric data field for list- or data- directed output will not be split over two lines when this can be avoided. A field which exceeds the residual space on the current line will begin a new line, and only if the new line cannot accommodate the complete field will it be carried over to subsequent lines.

Execution of the PL/I procedure shown in Figure 8 will produce the object time output in Figure 9.

Record-Oriented Output

This form of output is concerned with writing complete records, instead of one item at a time. One record is the contents of one variable. The external data representation is the same as the internal form; there is no data conversion. RECORD-oriented output is described in 'Managing Data'.

```
OUTPUT: PROCEDURE OPTIONS(MAIN);

        DCL  A FIXED,
             B FLOAT,
             C FLOAT COMPLEX,
             D BIT(10),
             E CHAR(10),
             ARRAY(2,2);

        A,B = 12345;
        C = A+12345I;
        D = '1100111'B;
        E = 'ABC''DEFG';

        DO I = 1 TO 2; DO J = 1 TO 2;
        ARRAY(I,J) = I+J;
        END; END;

        PUT PAGE LIST('EXAMPLES OF LIST/DATA/EDIT OUTPUT');

LIST:   PUT SKIP(2) LIST('LIST DIRECTED EXAMPLES:');

             PUT SKIP(1) LIST(A,B,C,D,E);

DATA:   PUT SKIP(2) LIST('DATA DIRECTED EXAMPLES:');

             PUT SKIP(1) DATA(A,B,C,D,E);
             PUT SKIP DATA(ARRAY);

EDIT:   PUT SKIP(2) LIST('EDIT DIRECTED EXAMPLES:');

             PUT SKIP(1) EDIT(A,B,C,D,E)
                             (F(10,2),E(10,0,5),C(E(10,0,5)),X(5),2A);

END;
```

Figure 8. Example of PL/I Procedure Using STREAM I/O

```
EXAMPLES OF LIST/DATA/EDIT OUTPUT

LIST DIRECTED EXAMPLES:
   12345              1.23450E+04          1.23450E+04+1.23450E+04I                    '1100111000'B
ABC'DEFG

DATA DIRECTED EXAMPLES:

E='ABC''DEFG  ';
ARRAY(1,1)= 2.00000E+00 ARRAY(1,2)= 3.000000E+00 ARRAY(2,1)= 3.000000E+00 ARRAY(2,2)= 4.00000E+00;

EDIT DIRECTED EXAMPLES:
  12345.00 12345E+00 12345E+00 12345E+00      1100111000ABC'DEFG
```

Figure 9. Example of LIST/DATA/EDIT Output to PRINT Files using the PUT Statement

Object Time Diagnostic Messages

When an exceptional or error condition arises during the execution of a PL/I program, a message of the form IHEnnnI - text is printed. The first three characters IHE are standard and indicate a PL/I library message. The message number is specified by the digits nnn. The final character, I, is a system standard action code, indicating an informative message for the programmer. These messages are printed on an output data set which is specified by the SYSPRINT DD statement provided by the programmer. If the SYSPRINT DD statement is absent, the object time messages appear on the operator's console, except for the ON CHECK system action messages and the COPY option output, which are not printed at all in this case. The text of the message, which may consist of any number of characters, is separated from the I by one blank. PL/I input/output messages include the word FILE and the file-name at the start of the message, to assist in the identification of the file causing the error.

If the statement number compiler option has been specified, each message will also contain IN STATEMENT nnnnn prior to AT location message. nnnnn gives the number of the statement in which the condition occurred.

Messages are generated for two reasons:

1. An error occurred for which no specific ON condition exists in PL/I. A diagnostic message is printed out and the ERROR condition is raised.

2. An ON condition is raised, by compiled code or by the library, and the action required is system action, for which the language specifies COMMENT as part of the necessary action.

The complete list of object time diagnostic messages appears in Appendix G of this publication.

Operator Messages

A message is transmitted to the operator when the DISPLAY statement is specified in the PL/I program. The two forms in which an operator message may appear are as follows:

1. Without the REPLY option - the message is in the form of a character string specified by the programmer.

2. With REPLY option - the message is in the form of a character string specified by the programmer, but it is preceded by a two-digit code generated by the operating system. The operator must use this code as the prefix to his reply message.

SYSTEM OUTPUT

Diagnostic messages and completion codes are issued by the operating system, indicating coding errors found in job control statements and system macro instructions, and errors detected during processing by the linkage editor. The completion codes indicate conditions causing the control program to abnormally terminate execution of a job. For details of the system messages and completion codes, refer to the publication IBM System/360 Operating System, Messages, Completion Codes and Storage Dumps.

USE OF CATALOGED PROCEDURES

To ease the work of the PL/I programmer, IBM has supplied cataloged procedures to assist in the compilation, linkage editor processing, and execution of PL/I source programs. The job control statements needed to invoke the procedures, and the deck structures used with the procedures, are described in the following paragraphs.

In the examples given in this section, the operand is specified as:

PROC = cataloged procedure name

but the 'PROC=' can be omitted if required.

It is recommended that each installation review the procedures supplied by IBM, and modify them to obtain more efficient use of the devices available, to allow for installation conventions, to accommodate local changes to the compiler options, to alter the REGION fields to correspond to available storage, or to set block sizes for output data sets.

Compilation with Deck Output

The cataloged procedure for compilation with deck output is PL1DFC. The programmer invokes this procedure by using an EXEC statement with the first parameter containing the name PL1DFC. An example is shown as follows:

```
//STEP1  EXEC  PROC=PL1DFC
```

The cataloged procedure, PL1DFC, consists of the control statements shown in Figure 43 in Appendix E.

Invoking the PL1DFC Cataloged Procedure: Figure 10 shows control statements that might be used to invoke the procedure.

```
//COMPIL    JOB  PL1PROG,MSGLEVEL=1
//STEP1     EXEC  PROC=PL1DFC
//PL1D.SYSIN DD  *
(Source Program)
/*
```

Figure 10. Invoking the Cataloged Procedure PL1DFC

Compilation with Object Module Output

The cataloged procedure for compilation with object module output is PL1LFC. The programmer invokes this procedure by using an EXEC statement with the first parameter containing the name PL1LFC. An example is shown as follows:

```
//STEP1     EXEC  PROC=PL1LFC
```

The cataloged procedure, PL1LFC, consists of the control statements shown in Figure 59 in Appendix E.

Invoking the PL1LFC Cataloged Procedure: Figure 11 shows control statements that might be used to invoke the procedure.

```
//COMPLM    JOB  PL1PROG,MSGLEVEL=1
//STEP1     EXEC  PROC=PL1LFC
//PL1L.SYSIN DD  *
(Source Program Deck)
/*
```

Figure 11. Invoking the Cataloged Procedure PL1LFC

Compilation and Link-Editing

The cataloged procedure for compiling and link-editing a PL/I source program is PL1LFCL. The programmer invokes this procedure by using an EXEC statement with the first parameter containing the name PL1LFCL. For example:

```
//STEP1     EXEC  PROC=PL1LFCL
```

The cataloged procedure for compiling and link-editing consists of the control statements shown in Figure 60 in Appendix E.

Invoking the PL1LFCL Cataloged Procedure: Figure 12 shows the statements that might be used to invoke the procedure PL1LFCL.

```
//JOBCLE    JOB  PL1PROG,MSGLEVEL=1
//STEP1     EXEC  PROC=PL1LFCL
//PL1L.SYSIN DD  *
(Source Program)
/*
```

Figure 12. Invoking the Cataloged Procedure PL1LFCL

Compilation, Link-Editing and Execution

Another cataloged procedure, PL1LFCLG, passes a source program through three procedure steps - compilation, link-editing, and execution. The cataloged procedure is

invoked by specifying the name PL1LFCLG as the first parameter in an EXEC statement. For example:

```
//STEP1    EXEC  PROC=PL1LFCLG,
                 REGION.GO=xxxxK
```

where xxxxK represents the REGION specification (leading zeros need not be specified).

Figure 61 in Appendix E shows the statements that make up the cataloged procedure PL1LFCLG.

Invoking the PL1LFCLG Cataloged Procedure: Figure 13 shows statements that might be used to invoke the procedure PL1LFCLG.

```
//COMPLEG   JOB  PL1PROG,MSGLEVEL=1

//STEP1     EXEC PROC=PL1LFCLG,
                 REGION.GO=xxxxK

//PL1L.SYSIN DD  *

(Source Program)

/*
```

Figure 13. Invoking the Cataloged Procedure PL1LFCLG

Link-Editing and Execution

The cataloged procedure for link-editing and executing a previously compiled PL/I program is PL1LFLG. This procedure is invoked by an EXEC statement with the first parameter containing the name PL1LFLG. For example:

```
//STEP1    EXEC  PROC=PL1LFLG,
                 REGION.GO=xxxxK
```

where xxxxK represents the REGION specification (leading zeros need not be specified).

Figure 62 in Appendix E shows the statements that make up the cataloged procedure PL1LFLG.

Invoking the PL1LFLG Cataloged Procedure: Figure 14 shows statements that might be used to invoke the procedure PL1LFLG.

```
//jobname   JOB  PL1PROG,MSGLEVEL=1

//STEP1     EXEC PROC=PL1LFLG,
                 REGION.GO=xxxxK

//LKED.SYSIN DD  parameters defining
                 the input data set
                 to the Linkage
                 Editor
```

Figure 14. Invoking the Cataloged Procedure PL1LFLG

Link-Editing and Execution of Several Separate Compilations

Several programs can be separately compiled, then link-edited together and executed as one program. This is done by using the cataloged procedure PL1LFC for each individual compilation except the last, then using the procedure PL1LFCLG for the last, all within the same job.

FILES AND DATA SETS

The term "file," as defined for PL/I, may be equated for the most part with the System/360 operating system term "data set." However, there is not necessarily a one-to-one correspondence between the number of files being used within a program and the number of data sets being used. That is, a program may declare two files, ALPHA and BETA, both of which are concurrently open and both referring to the same data set. This may be effected in two ways:

1. When opened, both files ALPHA and BETA use the same TITLE option value, i.e., they refer to the same DD statement.

2. Two DD statements are used, one for each file, and each nominates the same data set.

From the above, it may be seen that the PL/I program does not itself nominate data sets directly. The TITLE option of the OPEN statement is employed to specify the name (ddname) of the DD statement to be associated with a given file; in turn, the DD statement specifies its associated data set (dsname). Should the TITLE option be omitted from the OPEN statement, or should the file be opened implicitly by another I/O statement, a default TITLE is obtained from the file identifier; the initial eight characters, padded with blanks if necessary, are used.

If two or more files are assigned to the same data set or SYSOUT class (via different DD statements) and are simultaneously open for sequential output, then, if the PCP or MFT system is being used, the records in the files will be intermixed or the job will be abnormally terminated. If MVT is being used, the records will remain separate. This practice is extremely dangerous and is not recommended.

There are three data set organizations which are supported by PL/I object programs. Specification of data set organization is made by means of an option within the ENVIRONMENT attribute, which is discussed further in Appendix B.

The organization of a data set determines the manner in which data is stored within the data set and the permitted techniques of accessing it. The PL/I organizations used in the System/360 Operating System are:

CONSECUTIVE

INDEXED

REGIONAL

CONSECUTIVE organization permits only sequential access, while REGIONAL and INDEXED organization permit both direct and sequential access.

The permitted combinations of data set organization, PL/I statements, access methods, etc. are detailed in Figures 15, 16, and 17.

CONSECUTIVE ORGANIZATION

In CONSECUTIVE organization, the data set consists of records whose placement within the data set is determined solely by the order in which they were initially added. Such a data set does not employ keys; records may be retrieved only in sequential order. Device support for CONSECUTIVE data sets includes magnetic tapes, card readers/punches, direct access storage devices, and paper tape readers. However, for UPDATE files, only direct access devices may be used.

Both STREAM and RECORD I/O facilities may employ data sets of this organization. STREAM I/O is limited to this organization.

The facility for reading BACKWARDS is available only for files with records in F,FB,FBS, or U format on 9-track magnetic tapes, or 7-track magnetic tapes recorded without the use of the data converter. If the data converter is used, only string records should be read BACKWARDS (Record formats are described later in this chapter.)

INDEXED ORGANIZATION

In INDEXED organization, the data set consists of keyed records whose placement within the data set is determined by the value of the key. Since a key is associated with each record, direct retrieval, replacement, addition, or deletion of

Organization	Access	File Declaration		Mode	Statement	Options	Data Mgmt.	Record Format
CONSECUTIVE	SEQUENTIAL	[ENV (CONSECUTIVE)]	[BUFFERED] [BACKWARDS][1]	INPUT	READ	[INTO|IGNORE|SET]	QSAM	F|FB|FBS
				OUTPUT	WRITE	FROM		
					LOCATE	[SET]		
			[BUFFERED]		READ	[INTO|IGNORE|SET]		U|V|VB VS|VBS
				UPDATE[2]	REWRITE	[FROM]		
			UNBUFFERED [BACKWARDS][1]	INPUT	READ	[INTO|IGNORE] [EVENT]	BSAM	F|U|V
				OUTPUT	WRITE	FROM [EVENT]		
			UNBUFFERED		READ	[INTO|IGNORE] [EVENT]		
				UPDATE	REWRITE	FROM [EVENT]		

[1] BACKWARDS is available only for records of F, FB, FBS, and U format.

[2] UPDATE is not available for records of VS and VBS format.

Note: The syntax notation used in this table is the same as that used in IBM System/360 Operating System, PL/I Reference Manual.

●Figure 15. CONSECUTIVE Data Set Organization and Applicable Language Features

records is possible, as well as sequential access of such records. The keyed records of an INDEXED data set are located by means of several levels of indexes which are initially constructed when the data set is created.

Keys for INDEXED data sets consist of character strings, 1 through 255 characters in length, defined by means of the DCB subparameter KEYLEN(described later in this chapter). Should the length of a character string key specified in a KEY or KEYFROM option differ from that specified by KEYLEN, it is truncated or padded with blanks to KEYLEN.

Device support for INDEXED data sets is restricted to direct access storage devices.

Basically, there are three areas within an INDEXED data set:

Prime data area
Overflow area
Index area

The prime data area is the area into which records are inserted during creation. Subsequent addition of records, when using DIRECT access for UPDATE, may cause records to be moved from the prime data area to an overflow area, in order to keep the indexes in order. Access to records in an overflow area is slower than to those in the prime data area; accordingly, the data set should be "reorganized" periodically (see: "Accessing of INDEXED Data Sets").

There are two types of overflow area:

1. a cylinder overflow area (a certain number of tracks per cylinder reserved during creation)

2. an independent overflow area, which requires special DD statement usage (see "Creation of INDEXED Data Sets").

The index area may be spread, physically, through the space allocated to the data set. There are two basic index types, the track index and the cylinder index; a third type, the master index, is used if the cylinder index becomes too large for efficient access.

The track index specifies the highest key value for each track of a given cylin-

Organization	Access	File Declaration	Mode	Statement	Options	Data Mgmt.	Record Format
INDEXED[1]	SEQUENTIAL	ENV (INDEXED) [BUFFERED|UNBUFFERED] [KEYED][2]	INPUT	READ	[INTO|SET[4]] [KEYTO|KEY]|IGNORE]	QISAM (SCAN)	F|FB[3]
			OUTPUT	WRITE	FROM KEYFROM	QISAM (LOAD)	
				LOCATE[4]	[SET] KEYFROM		
			UPDATE	READ	[INTO|SET[4]] [KEYTO|KEY]|IGNORE]	QISAM (SCAN)	
				REWRITE	[FROM][4]		
				DELETE			
	DIRECT	ENV (INDEXED) [KEYED]	INPUT	READ	INTO KEY [EVENT]	BISAM	
				READ	INTO KEY [NOLOCK] [EVENT]		
		[EXCLUSIVE]	UPDATE	WRITE[5]	FROM KEYFROM [EVENT]		
				REWRITE	FROM KEY [EVENT]		
				DELETE	KEY [EVENT]		
				UNLOCK	KEY		

[1] INDEXED organization is supported only for direct access storage devices.

[2] For INDEXED SEQUENTIAL INPUT and UPDATE, the file attribute KEYED must be declared if the KEYTO or KEY option on the READ statement is to be used. DIRECT access implies KEYED.

[3] Blocked records are not supported for UNBUFFERED files.

[4] Locate mode I/O operations are applicable to BUFFERED files only.

[5] WRITE is ignored if NOWRITE is specified.

Note: The syntax notation used in this table is the same as that used in IBM System/360 Operating System, PL/I Reference Manual.

Figure 16. INDEXED Data Set Organization and Applicable Language Features

der; the cylinder index specifies the highest key value of each cylinder. The master index may be created (by specification of the NTM subparameter of the DCB parameter in the DD statement) when the cylinder index exceeds the specified number of tracks. Each entry in the master index points to a track within the cylinder index. Up to three levels of master index (addressing tracks within the next lowest level master index) may be created automatically.

Records within an INDEXED data set are either "actual" (representing still valid data) or "dummy" (marked as deleted, and inaccessible to the user). Dummy records are recognized only when both of the following are present:

1. The DCB subparameter OPTCD = L in the DD statement used to create the data set.

Organization	Access	File Declaration		Mode	Statement	Options	Data Mgmt.	Record Format
REGIONAL[1]	SEQUENTIAL	ENV (REGIONAL(1) | REGIONAL(2) | REGIONAL(3))	[BUFFERED] [KEYED][2]	INPUT	READ	[INTO|SET[KEYTO]|IGNORE]	BSAM[2]	REG(1) and(2): F only REG(3): F|U|V
			[BUFFERED] KEYED	OUTPUT	WRITE	FROM KEYFROM	BSAM (LOAD)	
					LOCATE	[SET] KEYFROM		
			[BUFFERED] [KEYED][2]	UPDATE	READ	[INTO|SET[KEYTO]|IGNORE]	BSAM[2]	
					REWRITE	[FROM]		
			UNBUFFERED [KEYED][2]	INPUT	READ	[INTO[KEYTO]|IGNORE] [EVENT]	BSAM	
			UNBUFFERED KEYED	OUTPUT	WRITE	FROM KEYFROM [EVENT]	BSAM (LOAD)	
			UNBUFFERED [KEYED][2]	UPDATE	READ	[INTO[KEYTO]|IGNORE] [EVENT]	BSAM	
					REWRITE	FROM [EVENT]		
	DIRECT	ENV (REGIONAL(1) | REGIONAL(2) | REGIONAL(3))	KEYED	INPUT	READ	INTO KEY [EVENT]		
				OUTPUT	WRITE	FROM KEYFROM [EVENT]		
			[EXCLUSIVE] [KEYED]	UPDATE	READ	INTO KEY [EVENT] [NOLOCK]	BDAM (BSAM: LOAD)	
					WRITE	FROM KEYFROM [EVENT]		
					REWRITE	FROM KEY [EVENT]		
					DELETE	KEY [EVENT]		
					UNLOCK	KEY		

[1] REGIONAL organization is supported only for direct access storage devices.

[2] For REGIONAL SEQUENTIAL INPUT and UPDATE, the file attribute KEYED must be declared if the KEYTO option on the READ statement is to be used (DIRECT access implies KEYED). If the KEYED attribute is not declared, QSAM is used for REGIONAL(1) INPUT and UPDATE.

Note: The syntax notation used in this table is the same as that used in IBM System/360 Operating System PL/I Reference Manual.

Figure 17. REGIONAL Data Set Organization and Applicable Language Features

2. A character whose value is (8)'1'B in the initial position of the record.

Should either condition not apply, a given record is not recognized as a dummy (deleted) record. Should the OPTCD = L subparameter be employed, the user is cautioned that unless care is taken to prevent the unintentional presence of such an initial character (for example, because of negative FIXED BINARY data), records may vanish from the data set.

REGIONAL ORGANIZATION

There are three types of REGIONAL data organization: REGIONAL(1), REGIONAL(2), and REGIONAL(3). REGIONAL(1) and (2) support unblocked F-format records; REGIONAL(3) supports unblocked records of U, F, or V-format.

In all three REGIONAL organizations, the data set consists of regions numbered consecutively, starting at zero. In REGIONAL(1) and (2), each region contains just one record. In REGIONAL(3), each region is a whole track, containing as many records as will fit onto the track. The actual number will depend upon the size of the records and the device characteristics.

REGIONAL organizations support DIRECT access of records by directly locating the region in which the record is placed and then, if necessary, performing a sequential search for the specified record in the vicinity of the located region. REGIONAL organizations also support SEQUENTIAL access of records.

A major advantage of REGIONAL organization is that it permits program control over the relative physical placement of records. Accordingly, optimization of record access in terms of particular application requirements and device capabilities can be obtained by judicious programming.

Source Keys and Recorded Keys

Two types of key are used with REGIONAL organizations: source keys and recorded keys.

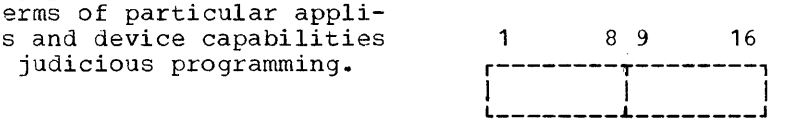
The source key is a character string specified in the KEYFROM option of a WRITE or LOCATE statement or in the KEY option of a READ, REWRITE, or DELETE statement. In the WRITE statement, its purpose is to determine whereabouts in the data set a new record will be written; in the READ, REWRITE, or DELETE statement, its purpose is to identify the record which is to be read, rewritten, or deleted. The last eight characters of the source key (or all of it, if there are less than eight) represent the region number of the record. In REGIONAL(1), any additional characters in the source key are ignored.

In REGIONAL(1), the region number alone is sufficient to determine the location in which a record will be placed and to identify it for retrieval. In REGIONAL(2) and (3) (as will be explained below), a further means of identification is needed; it takes the form of a character string which is actually written in the data set attached to the record, and is known as the recorded key. The recorded key consists of the initial n characters of the source key, where n is the number specified in the DCB subparameter KEYLEN. If the source key specified in the KEY or KEYFROM option is not equal in length to the number specified by KEYLEN, the source key is either padded with blanks or truncated (on the right) before being written with the record as the recorded key or compared with existing recorded keys in the search for a particular record.

The relationship between the portion of the source key which represents the region number of the record (i.e., the last eight characters of the source key) and the portion which constitutes the recorded key (i.e., the first n characters, where KEYLEN = n) should be carefully considered. Some possibilities are illustrated below. Note that all the examples refer to REGIONAL(2) or (3). In REGIONAL(1), there is no recorded key, and any characters in the source key other than the final eight are ignored.

Example 1:

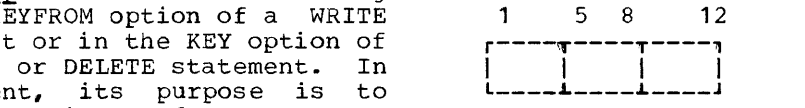
Source key of 16 characters; KEYLEN = 8



Characters 9 through 16 represent the region number; characters 1 through 8 are used as the recorded key.

Example 2:

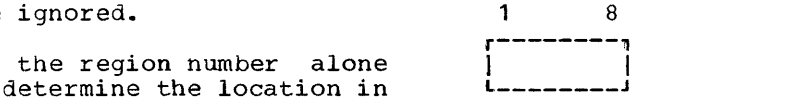
Source key of 12 characters; KEYLEN = 8



Characters 5 through 12 represent the region number; characters 1 through 8 are used as the recorded key.

Example 3:

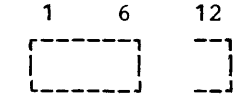
Source key of 8 characters; KEYLEN = 8



Characters 1 through 8 represent the region number; characters 1 through 8 are also used as the recorded key.

Example 4:

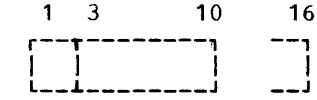
Source key of 6 characters; KEYLEN = 12



Characters 1 through 6 represent the region number; characters 1 through 6 are padded on the right with blanks to make 12 characters, which are then used as the recorded key.

Example 5:

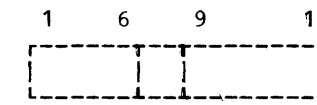
Source key of 10 characters; KEYLEN = 16



Characters 3 through 10 represent the region number; characters 1 through 10 are padded on the right with blanks to make 16 characters, which are then used as the recorded key.

Example 6:

Source key of 16 characters; KEYLEN = 6



Characters 9 through 16 represent the region number; characters 1 through 6 are used as the recorded key. Characters 7 and 8 are not used.

From the above illustrations, it can be seen that it is optional whether the region number portion of the source key is included in the recorded key. Similarly, the KEY option may specify a source key longer than the recorded key, thereby causing a search for a key matching only the initial n characters of the source key, where n is defined by KEYLEN; this facilitates reading data sets created by other processors (e.g., COBOL, which has a different source language keying mechanism).

SEQUENTIAL access of REGIONAL data sets will retrieve records in the order in which they are physically located in the data set. The recorded key will not affect the sequence, but can be obtained by use of the KEYTO option.

Equality between the value specified by the KEYFROM option and that subsequently assigned to a character string by means of the KEYTO option can only be guaranteed if the length of the KEYFROM value is equal to that specified by KEYLEN.

REGION L(1) Data Sets

A REGIONAL(1) data set consists of unblocked F-format records without associated recorded keys. The data set is divided into regions each of which is large enough to contain just one record, hence each region number corresponds to a relative record position within the data set. The location of a particular record can be determined by the source key alone. The final eight characters of the source key are the character-string representation of an unsigned decimal integer not exceeding 16777215, and may contain only the characters 0 through 9 and blank. If more than eight characters are given in the KEY or KEYFROM option, only the last eight are used; otherwise all characters are used.

If the KEYTO option specifies a character-string variable of more than eight characters, the source key value returned is padded at the left with blanks; if it has fewer than eight characters, the value is truncated on the left without raising any condition.

REGIONAL(2) Data Sets

A REGIONAL(2) data set contains unblocked F-format records with associated recorded keys. As in REGIONAL(1), the data set is divided into regions each of which will hold just one record. Unlike REGIONAL(1) however, in REGIONAL(2) organization a particular record will not necessarily occupy the relative position specified by the region number in the source key. When writing a new record in a data set opened for DIRECT access, the location in which the record will be written is determined as follows. A search is commenced at the beginning of the track containing the region number specified by the source key. The record is written in the first available position on that track or, if that track is full, on a subsequent track. The record may, therefore, be written either before or after the specified region number, depending on the availability of space. The search for a vacant position can, however, be limited by the DCB subparameter LIMCT. (The use of LIMCT in REGIONAL(2) is described below.) Similarly, when accessing a record in a REGIONAL(2) data set opened for DIRECT access, the record is located as follows.

Commencing at the beginning of the track containing the specified region number, a sequential search is made for a record whose recorded key matches the source key (truncated or padded as required). Again, the extent of the search can be restricted by the use of the DCB subparameter LIMCT.

The DCB subparameter LIMCT limits the number of regions which are to be sequentially searched. In REGIONAL(2), the search commences at the beginning of the track containing the region number specified by the source key, and continues to the end of the track containing the region whose number is given by:

region number specified in source key + LIMCT value - 1

Example:

Each track of a device holds ten records. The final eight characters of the source key represent the number 17. LIMCT has the value 15.

The sequential search will commence at the start of track 2 (since region number 17 is on this track) and continue through to the end of track 4 (since region number 31, given by source key region number + LIMCT value - 1, is on track 4).

It can be seen from the foregoing that, in REGIONAL(2), the nearer a record is to the beginning of a track, the more quickly can the record be accessed.

REGIONAL(2) organization does provide a degree of device independence. Provided that access to the data set is DIRECT and that all keys are unique, the programmer does not need to modify his algorithms in consideration of the device type. If, however, a data set created or updated by DIRECT access is subsequently read by SEQUENTIAL access, then the order in which the records are read will vary slightly according to the type of device.

The final eight characters of the source key are the character-string representation of an unsigned decimal integer not exceeding 16777215, and may contain only the characters 0 through 9 and blank. If more than eight characters are given in the KEY or KEYFROM option, only the last eight are used as the region number; otherwise all characters are used.

If the KEYTO option specifies a character-string variable of different length from that of the recorded key, the recorded key returned is padded with blanks at the right or truncated at the right; no condition is raised.

REGIONAL(3) Data Sets

A REGIONAL(3) data set may contain unblocked records of U,F, or V-format, with associated recorded keys. The data set is divided into regions, but unlike REGIONAL(1) and (2), in REGIONAL(3) a region is a whole track on the direct-access device. The region number in the source key thus corresponds not to a relative record position, but to a relative track on the device. The number of records within any region will depend on the record sizes and the device characteristics.

When writing a new record in a data set opened for DIRECT access, the location in which the record will be written is determined as follows. A search is made for an available position, commencing at the start of the track whose relative position is represented by the region number in the source key. The record will be written in the first available position on that track, or, if that track is full, on a subsequent track. The search can, however, be limited by use of the DCB subparameter LIMCT. (The use of LIMCT in REGIONAL(3) is described below.) Similarly, when accessing a record in a REGIONAL(3) data set opened for DIRECT access, the record is located as follows. Commencing at the beginning of the track whose relative position is represented by the region number in the source key, a sequential search is made for a record whose recorded key matches the source key (truncated or padded as required). Again, the extent of the search can be restricted by use of the DCB subparameter LIMCT.

The DCB subparameter LIMCT limits the number of regions which are to be sequentially searched. In REGIONAL(3), each region is a track, hence the value specified in LIMCT quite simply specifies the number of tracks which are to be searched.

The final eight characters of the source key are the character-string representation of an unsigned decimal integer not exceeding 32767, and may contain only the characters 0 through 9 and blank. If more than eight characters are given in the KEY or KEYFROM option, only the last eight are used as the region number; otherwise all characters are used.

If the KEYTO option specifies a character-string variable of length different from that of the recorded key, the recorded key returned is padded with blanks at the right or truncated at the right; no condition is raised.

Note: The EVENT option should be used with caution when V- or U-format records are being added to a REGIONAL(3) data set.

Dummy Records within REGIONAL Data Sets

Records within a REGIONAL data set are either "actual" (representing valid data) or "dummy" (inserted during creation or when actual records are deleted). Dummy records vary in format according to the REGIONAL type and, in the case of REGIONAL (3), according to the record format as well. See also the section "Accessing of REGIONAL Data Sets."

REGIONAL(1) - initial character of record has the value (8)'1'B; the remainder of the record is undefined.

REGIONAL(2) - initial character of recorded key has the value (8)'1'B, the remainder of the key being undefined; the initial character of the record is an internal representation of the sequence number of the record within a track, the remainder of the record being undefined.

REGIONAL(3) - if F-format records, as for REGIONAL (2). If U- or V-format records, the initial character of the key has the value (8)'1'B, the remainder of the key being undefined. If V-format, the control bytes of the record are valid, but the remainder is undefined.

DATA SET DEFINITION

Whereas the attributes of a file (which may be considered to be a "data set usage") are declared within a PL/I program, the attributes of its associated data set are specified by means of the DD statement or the ENVIRONMENT attribute. (For the format of the options available in the ENVIRONMENT attribute, refer to Appendix B.) In this manner a large degree of device independence is achieved. The DD statement permits the specification of object time device type, unit assignment, record format, etc.

Since PL/I itself does not define data set attributes, there are no default attributes, with the exception of PRINT files which have a default of the V-format, with unblocked records of 129 bytes. Information regarding specific data sets is communicated to the Operating System via DD statements. It is not, however, necessary for each programmer to understand the details of DD statements: in some instances the use of cataloged procedures will allow the programmer to be oblivious of the particular I/O device configuration. These cataloged procedures will be prepared by personnel familiar with requirements at a given installation (see Appendix E). Also, use of cataloged data sets will reduce the amount of configuration knowledge required.

SPECIFYING DATA SETS

Data sets in the operating system can be specified by parameters in the DD statement. This section describes the use of such DD statements.

The DD statement provides the following functions:

1. Naming of the data set (DSNAME)
2. Extent allocation, i.e., device and space (UNIT, VOLUME, SPACE, LABEL)
3. Disposition (DISP, SYSOUT)
4. Data set attributes (DCB)

The functions are summarized in the following paragraphs. For full details refer to the publication IBM System/360 Operating System, Job Control Language.

Naming of the Data Set (DSNAME)

The DSNAME parameter allows the programmer to specify the name of a newly defined data set or to refer to a previously defined data set. This parameter should be supplied for all but temporary data sets.

Note: Data set names that begin with the letters SYS and have a P as the nineteenth character of the name should not be used. Data sets with such names are created for temporary data sets on PCP systems, but not for temporary data sets on MVT or MFT systems, and are deleted when the IEHPROGM utility is used and the SCRATCH utility control statement is specified with the VTOC, PURGE, and SYS keywords.

Extent Allocation (UNIT, VOLUME, SPACE, LABEL)

The UNIT parameter allows the programmer to state the type and quantity of input/output devices to be allocated for the data set. The type of unit is specified by a symbol that refers to a collection of one or more devices.

The VOLUME parameter is used to specify the identification of the volume on which the data set resides. Other information includes instructions to the system for volume mounting actions.

The SPACE parameter permits the specification of the type and amount of space required to accommodate the data set on a direct-access device. This space is obtained either from the volume designated in the VOLUME parameter or from the volume mounted on one of the devices specified in the UNIT parameter.

The LABEL parameter specifies the type and contents of the label or labels associated with the data set.

Disposition (DISP,SYSOUT)

The disposition (DISP) parameter indicates the current status of the data set and what is to be done with it when the step is completed. Parameters are provided to delete the data set, to pass the data set to another step within the same job, to enter the data set name into the catalog, or to remove the data set name from the catalog.

DISP must be carefully specified when the EXCLUSIVE attribute is used. This attribute protects records against access by different tasks in the same program, provided the references to a data set are made by means of the same file. DISP=OLD ensures that a data set is used by one job only; DISP=SHR allows a data set to be used by different jobs. If a data set is to be updated by more than one job, DISP=SHR must not be specified.

The system output (SYSOUT) parameter is used to schedule a printing or punching operation for the data set described by this DD statement.

Data Set Attributes (DCB)

The DCB parameter allows the programmer, at execution time, to complete information declared for his file at the time of compilation via a FILE declaration or OPEN statement.

File Attributes and the DD Statement

The DCB parameter of the DD statement permits object-time definition of various data set characteristics. Whereas the other DD statement parameters deal chiefly with the physical residence of a data set, or aid in data set identification, the DCB parameter may specify information required to process the data records themselves. Information supplied to the object program by means of the DCB parameter can only be "additive" information, i.e., it may not override information already established for a data set during compilation. DCB subparameters which specify DCB fields already completed during compilation (by means of the file declaration or OPEN attributes) will be ignored.

DCB subparameters which may be specified for STREAM-oriented I/O are listed in Figure 18. The DCB subparameters which may be specified in the PL/I language for this mode of I/O are:

RECFM

BLKSIZE

LRECL

BUFNO

The first three are specifiable by means of the record option of the ENVIRONMENT file declaration attribute (see Appendix B); the fourth is specified by means of the BUFFERS option. In addition, the LINESIZE attribute of the OPEN statement for PRINT files may specify the initial three subparameters to be established by default; see the section called "PRINT Files."

DCB subparameters which may be specified for RECORD I/O include those available for STREAM I/O. Additional subparameters are listed in Figure 19.

Data set information supplied within the PL/I program may not be overridden by DD statement parameters, or data set label information (in the instance of direct-access devices, the DSCB). Accordingly, files which are to remain open to object-time definition of data set characteristics must not specify these characteristics within a program. (It should be noted that data sets which are defined by a "DD *" statement are supplied with full DCB subparameters by the Operating System.) Data control blocks generated by the PL/I "open" process at object time are of a format which allows complete device-type independence when applicable. During various executions of a program, a file may be associated with a card reader, paper tape, magnetic tape, or a direct-access device.

Keyword	Specifies	PL/I Option
BLKSIZE	Maximum, or fixed, number of bytes per physical record (block)	ENV (Record) LINESIZE
LRECL	Maximum, or fixed, number of bytes per logical record	ENV (Record) LINESIZE
RECFM	Record format and characteristics (usage of printer/punch control character)	ENV (Record) LINESIZE| PRINT
BUFNO	Number of buffers for physical record transmission	ENV (BUFFERS)
CODE	Paper tape: code in which the data was punched	-
DEN	Magnetic tape: tape recording density	-
TRTCH	Magnetic tape: tape recording technique for 7-track tapes	-
MODE	Card reader or punch: mode of operation, column binary or EBCDIC	-
STACK	Card reader or punch: stacker selection	-
PRTSP	Spacing of printer between each line (0, 1, 2, or 3)	- -
OPTCD	Optional data management services and data set attributes	-

Figure 18. DCB Subparameters for STREAM I/O

Keyword	Specifies	Applicable Organization
KEYLEN	Number of characters in recorded key	INDEXED REGIONAL (2) and (3)
LIMCT	Maximum number of records or tracks to be searched	REGIONAL (2) and (3)
RKP	Relative key position within record	INDEXED
NTM	Number of tracks in cylinder index per master index entry	INDEXED
CYLOFL	Number of overflow tracks in each cylinder	INDEXED
DSORG	Creation of INDEXED or REGIONAL data set	INDEXED REGIONAL (1),(2), and (3)
NCP	Number of channel programs allocated to a file	All

Figure 19. Additional DCB Subparameters for RECORD I/O

The format of the DCB subparameters is given in the following paragraphs. Refer to Appendix B for the format of the ENVIRONMENT option.

RECORD FORMAT INFORMATION

The notation used in the format descriptions of record format is interpreted as follows:

Upper case letters and numbers must be coded by the programmer exactly as shown.

Lower case letters represent variables for which the programmer must substitute specific information or specific values.

Items or groups of items within brackets [] are optional. They may be omitted at the programmer's discretion.

A logical OR sign | indicates a choice of option.

Record Types

The PL/I programmer may create and access data sets which consist of one of the following types of record:

Fixed-length (F-format) records
Undefined-length (U-format) records
Variable-length (V-format) records

Whichever record type is used, data is transmitted to and from main storage, and written on external storage in blocks. A block is considered to be either the data between two interrecord gaps in the case of magnetic storage devices such as disks and magnetic tape, or the number of bytes which can be transmitted to or from a unit record device, such as a line printer or card reader, in one input/output operation.

Fixed-Length Records: A fixed-length (F-format) record consists of a predetermined number of bytes, and is transmitted in a block containing one or more of these records. The length of a block containing fixed-length records is an exact multiple of the record length used. If unblocked records are specified, each record is transmitted and stored as a separate block.

Undefined-Length Records: An undefined-length (U-format) record simply consists of a string of bytes containing data. Each record is transmitted or stored as a separate block. The interrecord gap between blocks delimits undefined-length records when held on an external storage medium, such as magnetic tape or disk.

Variable-Length Records: A variable-length (V-format) record consists of a four-byte control field containing the number of bytes in the record, including the control field, followed by bytes containing data. A variable-length record is transmitted in a block formed by one or more such records. The length of the block depends on the number and length of consecutive variable-length records which can be accommodated within the maximum block size specified. The block itself contains a four-byte length field to determine its actual length, including the control field.

Variable-length records can also be specified as spanned (VS-format or VBS-format) records. Spanned records are written over block boundaries in order to enable maximum-length blocks to be used in conjunction with variable-length records, thereby permitting efficient use of external storage. A spanned record is a logical record which can be divided into segments which are written on separate consecutive blocks. Each segment contains a four-byte control field to indicate the length of the segment, including the control field, and a flag byte to designate a segment as the first, intermediate, or last segment of a logical record which spans blocks. If a complete logical record does not span blocks, i.e., it is contained entirely within one block, it is designated as the last or only segment of the logical record. Spanned records are created and retrieved automatically, and their use in no way affects the programmer. Data sets containing spanned records must be CONSECUTIVE. They can be opened as INPUT or OUTPUT files, but not as UPDATE files.

Block Size

BLKSIZE=nnnnn: the decimal integer specified must not exceed 32,760, and may be limited by the device used, such as a unit record device. Blocks to contain V-format records must exceed the maximum record length by four bytes, for control purposes. Blocks to contain VS- or VBS-format records may be shorter than the maximum record length.

Logical Record Length

LRECL=nnnnn: the decimal integer specified must not exceed 32,760 bytes for F-format records, and 32,756 for V-format records. The LRECL subparameter specifies the length of fixed-length records, and the maximum length of V-format, VS- and VBS-format records. It is not specified for U-format records for which the maximum length is 32,760 bytes. If not specified for the V-, VS-, or VBS-format, or F-format records, the BLKSIZE specified is assumed. For VS-, VBS- or V-format records, a four-byte control field must be included in the LRECL quantity. An extra byte must be added for all record formats which include a printer/punch control character. This includes all PRINT files, which use ASA printer control characters.

Note that for INDEXED data sets which contain blocked records, if RKP=0 the value of LRECL must include the number of bytes required for the recorded key.

Record Format

RECFM= U [T][A|M]

V [B] [S] [T] [A|M]

F [B][S][T][A|M]

The record format is specified as follows:

U: Undefined-length records

V: Variable-length records

F: Fixed-length records

The additional subfields are indicated by the following:

B: Blocked records

S: (F-format records only) Standard: no blocks of a length less than the specified blocksize, except possibly the last, are present in the data set

S: (V-format records only) Spanned records: variable-length records can span maximum-length blocks. A spanned record is written on one or more consecutive blocks. The length of the spanned record can exceed the length of the maximum-length block.

For variable-length VS-format records the maximum block size must always be stated. VS-format records that exceed the maximum block size are segmented, and the segments are placed in consecutive blocks. Each block can contain only one record or segment of a record. For example, if a record format is specified as VS(80,200), an ENV or DCB record that includes 180 bytes of data will appear in the data set as two blocks of 80 bytes (8 control bytes and 72 data bytes) and one block of 44 bytes (8 control bytes and 36 data bytes).

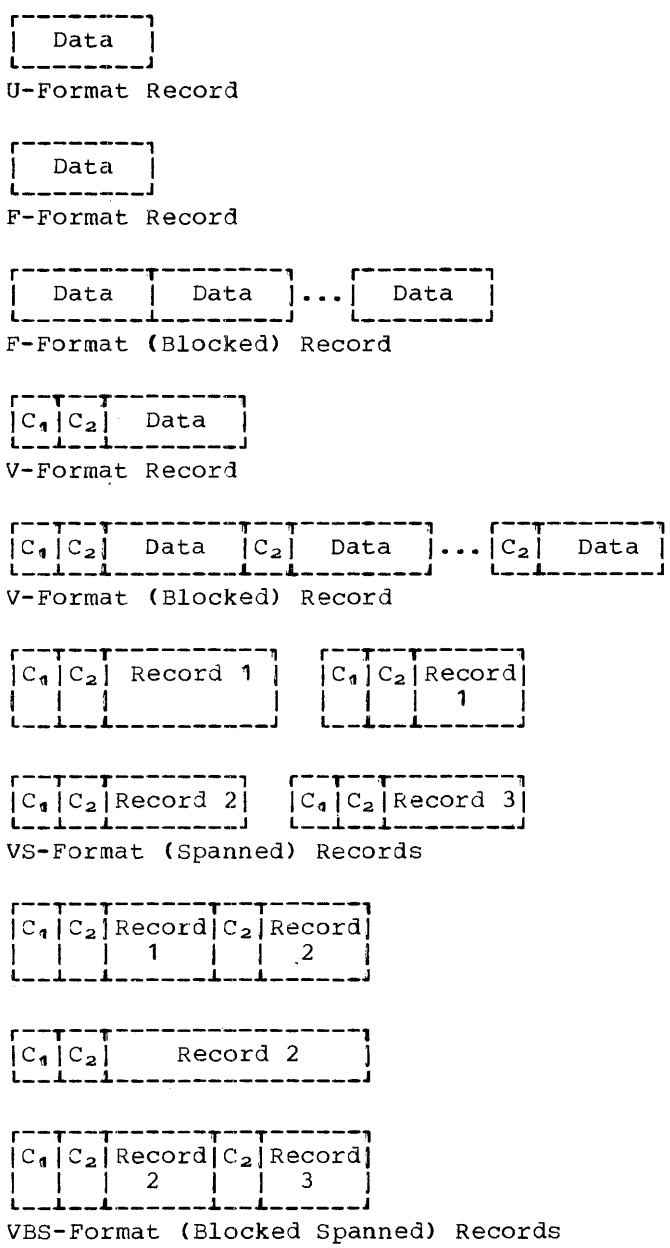
Varying-length VBS-format records are similar to VS-format records except that they are blocked, that is, each block contains as many complete records or segments of records as it can accommodate; each block is substantially the same size, although there can be a variation of up to four bytes, since each segment must contain at least one byte of data. For example, a block might contain the last segment of one record, one or more complete records, and the first segment of another record.

T: Track Overflow. Track overflow is a feature of System/360 operating system which can be incorporated at system generation time; it requires the record overflow feature on the direct access storage control unit. Track Overflow allows a record to overflow from one track to another. It is useful in achieving a greater data packing efficiency, and allows the size of a record to exceed the capacity of a track.
Note: Track overflow is not available for REGIONAL(3) data sets with U- or V-format records or for INDEXED data sets.

A: ASA printer/punch control characters are present within the records

M: Machine code printer/punch control characters are present within the records

The following figures illustrate the various record formats:



The control field C_1 (four bytes) specifies the total byte count of the block (including C_1); the field C_2 (four bytes) specifies, for each logical record, the byte count of the logical record (including C_2). In the figure depicting spanned (VS-format) records, the field C_2 contains the number of bytes including C_2, in the segment of a spanned record that is written on a particular block. Field C_2 also contains a flag to indicate the relative position of any segment of a spanned record within a logical record which is written on one or more blocks. It designates a segment as the first, intermediate, last, or only segment of a record. The flag is set and used automatically when this type of record format is specified.

If the file is a PRINT file, the initial byte of the data portion of each logical record is reserved for an ASA printer control byte.

Blocked records are not accepted from paper tape. Fixed-length records on paper tape have a fixed number of characters after translation; the number of characters before translation is not fixed.

Use of the Various Record Formats in STREAM I/O

The three available record formats, undefined, variable-length, and fixed-length, must each be considered for its application to the features of STREAM I/O. The following points should be considered when making a choice for OUTPUT files.

Variable-Length Records: The variable-length record format is the most versatile of the three available formats. In order to prevent the output of possible 'noise' records, i.e., records of less than the System/360 operating system standardized minimum length, the records for output files are padded to contain at least ten data characters. With the addition of the eight bytes required for the control fields, this gives a total of 18 data characters. Accordingly, LINESIZE should not specify a value less than 10 (9 for PRINT files, because a control character is required).

Fixed-Length Records: For output files -- as the printing options/format items have the ability to truncate a record (line), each record is padded with blanks when the following record is prepared for output.

Blocked output is not supported on unit record devices, where each logical record begins on a new card or line. For example, specification of recordsize=20, blocksize=80 for a card punch will result in only the first 20 columns of each card being punched. If these cards are then read in, the stream will contain 60 blank characters between each of the original records.

Also, the length of logical records should not exceed the physical size of each unit record, as the excess characters will be truncated without notification.

Use of the Various Record Formats in RECORD I/O

Data transmitted by a single RECORD I/O statement is always contained within a single record, the size of which should be large enough to hold the data aggregate being transmitted, yet small enough to conform to device restrictions on the length of records which may be transmitted. Determination of the record length required for the various formats of data aggregates may require the information available in the section "Structure Mapping" in Appendix C of this publication; similar calculations are required for determination of space required for array data aggregates.

It should be noted that when a data aggregate is moved into a record buffer, or transmitted immediately from the area in which the data aggregate is stored, no "padding bytes" are added before or after those bytes defined by the aggregate (or string). The current length of a variable-bit string will be such as to terminate the string on a byte boundary.

Another consideration affecting the transmission of data aggregates is the fact that the implementation includes locate mode I/O. This allows data to be processed in the input/output buffers. Therefore a programmer who is creating files to be read using READ statements with the SET option should bear in mind that:

1. The first byte in a block read from a file will be aligned in the buffer on a doubleword boundary.

2. If any data within any record in the block has an alignment requirement, this requirement must be preserved in the buffer when the record is read in.

For all record formats, this first byte is, with one exception, the first byte of data in the first record of the block. The exception is for FB-format records in an INDEXED file where RKP=0; here the first byte of the key of the first record is aligned on the doubleword boundary. The following information, if present, precedes this double word boundary:

1. Keys for F-format INDEXED files or for REGIONAL files.

2. Control fields for V- and VB-format records.

The user must be aware that, for the subsequent records in the buffer, the presence of keys and control fields will affect the alignment of the data elements in these records.

There are several constraints regarding record format in RECORD I/O which depend upon the data set organization and the manner in which it is accessed.

CONSECUTIVE Records: V-, VB-, VS,- and VBS-format data sets may not be read BACKWARDS.

INDEXED Records: Only F- and FB-format records are available.

REGIONAL(1) and (2) Records: Only F-format records are available.

REGION L(3) Records: Record formats U,F, and V are available, i.e., records cannot be blocked.

Additionally, there are various overheads which apply to the use of different formats within different data set organizations. The choice between BUFFERED and UNBUFFERED for SEQUENTIAL files may depend upon the time and space overheads outlined, as may the choice between the record formats U and V for REGIONAL(3) data sets. In the following details, the term "hidden buffers" refers to work buffers allocated by the object program library to provide special areas for the actual transmission of records to and from the data set, even though the file is declared UNBUFFERED or DIRECT.

UNBUFFERED access of CONSECUTIVE data sets employs hidden buffers if V-format records are used, in order to handle the V-format control bytes.

An INDEXED data set always requires hidden buffers in order to handle a special 10 byte control field.

A REGIONAL data set accessed sequentially, and declared KEYED, requires hidden buffers (except for REGIONAL(1)) to handle both the key and the record, which must be transmitted to and from contiguous storage areas.

Use of Spanned Records

The blocksize (BLKSIZE) used for spanned records can be selected independently of the record lengths likely to be encountered. It should be chosen with regard to attaining efficient use of external storage combined with efficient use of processor-I/O overlap. For instance, the block size selected for use on a 2311 disk store should be the correct part of one track, to ensure that the entire track is used and that efficient processing is achieved.

Spanned Records and LOCATE I/O

When READ SET is used with spanned records, each record is built up from its segments in an additional area (buffer) supplied by the PL/I library, and the pointer is set to address this area; the record is not processed in the data management buffer. Similarly, with LOCATE, a record for output is built up in a library area before being passed to data management for segmentation and transmission. Consequently, the advantage gained by using locate mode is lessened if the records are spanned.

Number of Buffers

BUFNO=n: the decimal constant must be less than 256.

If a file is declared BUFFERED, but the number of buffers is not specified or is specified as zero, the number is assumed to be 2.

Performance is not necessarily improved by specifying a large number of buffers. Generally, the use of more than two gives little improvement. In some circumstances, however, more may be required; for example, chained scheduling requires at least three.

Number of Channel Programs

NCP=n: the decimal constant must not exceed 99.

Specifies the number of channel programs allocated to the file when it is opened. The number of incomplete input/output operations may not exceed the number of channel programs. This option is available only for DIRECT access to INDEXED data sets or for UNBUFFERED SEQUENTIAL access to CONSECUTIVE or REGIONAL data sets. The default value assumed if the option is not specified is 1.

An attempt to exceed the appropriate number of incomplete input/output operations will have one of two results. For UNBUFFERED SEQUENTIAL access to CONSECUTIVE or REGIONAL data sets, the ERROR condition will be raised. For DIRECT access to an INDEXED data set, the operations, will be queued until a channel program becomes available.

Chained Scheduling

OPTCD=C

Specifies chained scheduling, which improves input/output performance by reducing the time required to transmit records to and from external devices. If this option is specified with QSAM, it is recommended that at least three buffers are specified for the file; with BSAM, the number of channel programs should be at least three.

If BSAM or QSAM are used, then chained scheduling will not be used for INPUT or UPDATE with U-format records.

Source Code (Paper tape)

CODE=I|F|B|C|A|T|N, where the codes specify:

I: IBM BCD perforated tape and transmission code (8 tracks)

F: Friden (8 tracks)

B: Burroughs (7 tracks)

C: NCR (8 tracks)

A: ASCII (8 tracks)

T: Teletype (5 tracks)

N: No conversion (F-format records only)

If no code is specified, I is assumed.

Density (Magnetic Tape)

DEN=0|1|2|3

DEN Value	Bytes per Inch: 7-track	Bytes per Inch: 9-track
0	200	-
1	556	-
2	800	800
3	-	1600

The density assumed, if none is specified, is:

7-track	200
9-track (one density)	800
9-track (dual density)	1600

Conversion (Magnetic tape - 7-track)

TRTCH=C|E|T|ET

C: Data conversion feature is used

E: Even parity

T: Translation from BCD to EBCDIC required

ET: (Both E and T, above)

This subparameter is required only for 7-track magnetic tape data sets.

Use of a tape recording technique other than TRTCH=C restricts the character set in which data can be written if it is subsequently to be reread and result in the same bit-configuration in storage.

In PL/I terms, for STREAM and RECORD input/output of character strings or pictured data, acceptable modes are TRTCH=C|T (with TRTCH=ET acceptable if characters are restricted to the 48-character set). For RECORD input/output of arithmetic data, TRTCH=C must be used.

Mode (Card Reader, Punch)

MODE=C|E

C: Column binary card images

E: EBCDIC card code

Stacking (Card reader, punch)

STACK=1|2

1: All cards read or punched are to be fed into stacker pocket 1

2: All cards read or punched are to be fed into stacker pocket 2.

Stacker pocket 1 is assumed if neither is specified.

Print Spacing (Printer)

PRTSP=0|1|2|3

Specifies spacing of 0 through 3 lines after printing each line; a value of 3 will cause 2 blank lines to appear between each printed line.

This parameter is ignored if the record format uses ASA control characters.

UCS Printer - Suppress TRANSMIT

OPTCD=U

If a printer has the universal character set feature, the TRANSMIT condition is normally raised when an invalid character is passed to the printer. Specifying OPTCD=U suppresses the condition. Either way, a blank character is printed.

Validity Check

OPTCD=W

W: Perform write validity check when writing on a direct-access device.

Deleted Records

OPTCD=L

Specifies that records within an INDEXED data set are to be recognized as deleted if the initial character (byte) of the record has the value (8)'1'B.

Keys

KEYLEN=n

Specifies the number of characters to be written, or retrieved, as the recorded key of records in data sets of the INDEXED or REGIONAL(2) and (3) organizations. Note that this value is fixed during data set creation, and that all records within a data set will have keys of the same length, the maximum length being 255 characters.

Extended Search Limit

LIMCT=n

Limits the extent of the search through a REGIONAL(2) or (3) data set beyond the region number specified in the source key. If LIMCT is not specified, the search for a particular record, or for space to add a new record, will continue from the specified region to the end of the data set.

In REGIONAL(2) LIMCT specifies records, but the search will continue to the end of the track containing the record whose region number is given by:

source key region no. + LIMCT - 1

In REGIONAL(3), LIMCT specifies tracks: the number of tracks searched will be the number specified by LIMCT.

Relative Key Position

RKP=n

Keys for INDEXED records may be separate from or embedded in the records. RKP=n specifies the position (n) of the first byte of the key relative to the beginning of the record (byte 0). RKP=0 implies that the key is not embedded. Note that n is always 1 less than the byte number of the first character of the key; that is, if RKP=1 then the first character of the key is in byte 2.
Embedded keys obviate the need for the KEYTO option for sequential input, but not the KEYFROM option for output (the data specified by the KEYFROM option may be the embedded key itself, however). Unblocked records always have a separate key (recorded key) attached to the record, even when there is already an embedded key; such records will therefore require double the space for key information. The maximum value for n is (recordsize - keylength).

Master Indexes

OPTCD=M

NTM=n

Specifies the number of tracks within a cylinder index which are to be filled before a master index entry is to be created. It also specifies the number of tracks within a master index (of which there may be three levels), which, when exceeded, will cause an entry in the next higher master index level. Such a facility is advantageous for large data sets, in order to avoid extensive serial searches through large low-level indexes. The maximum value for n is 99. Note that the OPTCD subparameter list must include the "M" option in order to create master indexes.

Independent Overflow Area

OPTCD=I

Specifies that an independent overflow area is to be used and must be defined in a separate DD statement.

Cylinder Overflow Area

OPTCD=Y

CYLOFL=n

Specifies number of tracks within each cylinder to be reserved, during creation, for overflow records. If an independent overflow area is requested (via another DD statement), a cylinder overflow area need not be specified. The maximum number of such tracks is 99, but the limit varies with the particular device: there must be at least one track on a given cylinder to hold the track index, and one to hold prime data.

Data Set Organization

DSORG=IS|DA

IS: specifies that an INDEXED data set is being created

DA: specifies that a REGIONAL data set is being created

PRINT Files

Declaration of the PRINT attribute for a file causes the initial data byte within each record to be reserved for an ASA printer control character. These control characters are set by the PAGE, SKIP, or LINE options and format items. Unless overridden by these options or format items, the printer control character is set for single spacing. The following control characters are used:

Page eject	1
Single space	b
Double space	0
Triple space	-
Suppress space	+

All these characters cause the spacing action to occur before printing the given record (it should be noted that a record is, in System/360 operating system, a line). The ASA control characters for 'skip to channel n' are not employed. It is possible to use them, however, by specifying in the DD statement that the ASA character is present, but not declaring the file to be a PRINT file and not specifying record format information in the ENVIRONMENT attribute: in this case, the initial byte of each record is available to the program. Also, by the above means, it is possible to use the initial byte for printer control if the RECFM subparameter specifies the use of machine carriage control characters by the letter M.

While a PRINT file is always an output file, it is possible to read such a file, but not as a PRINT file. This is effected simply by not declaring the file as PRINT, and opening it for INPUT: the initial data byte of each record in the data set will then be available for inspection.

PRINT files, since they have a default LINESIZE of 120 characters, need not have any record information specified for them. The complete default record information will become:

```
BLKSIZE = 129
LRECL   = 125
RECFM   = VA
```

Note: The LINESIZE for a PRINT file can be varied during execution by closing the file and reopening it with a different LINESIZE option. If this action is taken on a file which is being output to any medium other than a printer, (the use of MVT and MFT-II may cause a file intended for direct printing to be held temporarily on a direct-access device) the record and block size established for the file originally cannot be changed, and any change to LINESIZE must be accommodated within the existing record and block size. For example, a PRINT file declared with V-format records with a maximum length of 60 bytes, a block size of 69 bytes, and opened with a LINESIZE of 60 bytes may not be reopened with a LINESIZE of 100 bytes, as this will raise the UNDEFINEDFILE condition. To avoid this, declare the file explicitly with a maximum record length and block size large enough to accommodate the increased LINESIZE. The block size should exceed the LINESIZE by at least 9 bytes. If the record length is not declared explicitly, it is set to the length specified in LINESIZE when the file is first opened.

When the SYSPRINT file is opened, the PL/I program automatically positions the file output at the start of a new physical page. A blank page will appear if the first PUT statement using the file:

1. Has the PAGE option, or
2. Is a PUT EDIT with PAGE as the first format item

If a SIGNAL ENDPAGE (filename) statement is present for which there is no corresponding ON ENDPAGE (filename) on-unit, the statement is ignored, and the standard system action to start a new page is suppressed. This prevents the standard system action from attempting to start a new page on a file which may not have been opened.

RECORD I/O is not applicable to PRINT files, but the use of printer spacing characters (ASA or machine code, the latter being device dependent) is available through the DD statement by means of the DCB subparameter of RECFM, or by using one of the options CTIASA/CTL360 in the ENVIRONMENT attribute. The user is then responsible for ensuring that the initial character of the record is a valid printer control character, which may be a 'skip to channel n'.

The ASA channel skip codes are:

Channel	Code
1 through 9	1 through 9
10 through 12	A through C

Also, punch stacker select uses the following characters:

punch pocket 1	V
punch pocket 2	W

Should any other code be employed, it will be interpreted as either single-space or pocket 1, depending upon the device in use.

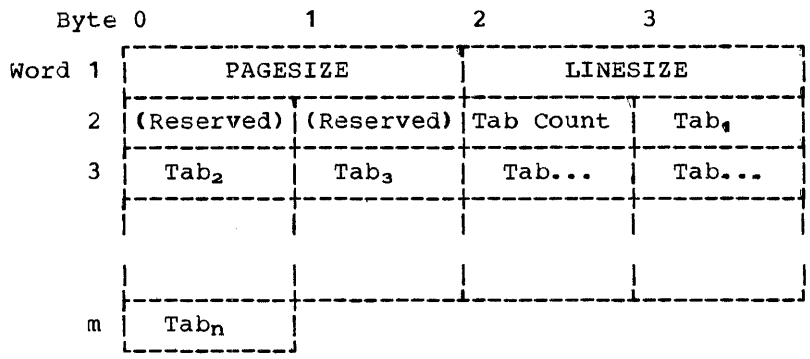


Figure 20. Format of the Tab Control Table

The Tab Control Table

DATA and LIST directed output to PRINT files will automatically align data on preset tab positions. The positions are determined from the tab control table, which is the PL/I Library module IHETAB. This table can be altered, as described below, and either replaced within the library permanently or inserted in the linkage editor input stream for use in a single program only. the format of the tab control table is shown in Figure 20.

PAGESIZE: Half-word binary integer defining the default PAGESIZE

LINESIZE: Half-word binary integer defining the default LINESIZE

Reserved Bytes: Reserved for left and right margin facilities

Tab Count: Specifies number of tab position entries within table (maximum of 255). If the tab count = 0, then the tab positions are not used; each data item is put out according to the STREAM file output mode used.

Tab_1-Tab_n: Tab positions within the print line: initial position is numbered 1, greatest position is numbered 255. The value of each tab should be greater than that of the tab preceding it in the table; if not, it is ignored during table scanning, which proceeds through the table from Tab_1 to Tab_n for each data field. Data fields are begun at the tab position specified, or, in the case of the first data field of a line, at the left margin default (position 1).

In its unaltered form, the table is set up as follows:

60		120	
0	0	5	25
49	73	97	121

Changing the Tab Settings

To change the tab settings, a user must change the values in the assembler language control section listed below. To become an installation standard, this must be link-edited into the PL/I library (data set SYS1.PL1LIB) to replace module IHETABS. For use in one program only, the object deck is link-edited with that program.

```
TAB         TITLE   'IHETAB'
IHETAB      CSECT
            ENTRY   IHETABS
IHETABS     DS      0D
PAGESIZE    DC      AL2(60)
LINESIZE    DC      AL2(120)
            DC      H'0'
NOTABS      DC      AL1(ENDTABS-*-1)
TAB1        DC      AL1(25)
TAB2        DC      AL1(49)
TAB3        DC      AL1(73)
TAB4        DC      AL1(97)
TAB5        DC      AL1(121)
ENDTABS     EQU     *
            END
```

CREATING AND ACCESSING DATA SETS

The facilities available for initially creating data sets are outlined in the following paragraphs, together with details for the subsequent usage of created data sets. It should be noted that creation of a data set is indicated by opening a file for OUTPUT, while subsequent access is indicated by either INPUT or UPDATE.

STREAM DATA SETS

Creation of STREAM Data Sets

There are no special considerations with regard to the data set specification beyond those already indicated under "File Attributes and the DD Statement."

Accessing of STREAM Data Sets

After a STREAM data set is created, it may, with one exception, be opened only for INPUT. The exception is that it may be opened for OUTPUT using the parameter DISP=MOD, in which case the new records are added at the end of the existing records in the data set. In any other circumstances, opening a STREAM data set for OUTPUT will result in the data set being overwritten.

RECORD DATA SETS

Creation and accessing of RECORD-oriented data sets varies according to the three data set organizations available. Details are given below for each of the three types: CONSECUTIVE, INDEXED, and REGIONAL.

Use of the EVENT Option

The EVENT option gives the user the ability to overlap input/output operations with internal processing. This option may be used with statements for all RECORD files except:

1. SEQUENTIAL BUFFERED (PL/I language restriction)
2. SEQUENTIAL INDEXED ((F) compiler implementation restriction).

When an I/O statement is executed with the EVENT option, input/output of the record is initiated but is not completed before control is returned to the user's problem program. The user can take advantage of the time taken for the I/O operation by incorporating statements whose execution is independent of the I/O operation. When, at some later stage, it is necessary to ensure tnat the I/O operation is complete, the WAIT statement is used.

Event Variable: The Event Variable, when used in conjunction with a RECORD I/O statement, can be in either an active or an inactive state. It becomes active when an I/O statement with which it is associated is executed. It remains active until any I/O conditions associated with the I/O operation have been raised or until the I/O operation has been completed (either normally or abnormally). The Event Variable also becomes inactive after a GO TO out of an ON-unit entered as a result of a condition being raised. An Event Variable which is active must not be used again in any I/O statement, a CALL or assignment statement, or in the COMPLETION pseudovariable, until it has become inactive.

Raising Conditions: All I/O conditions are raised at the time of execution of the WAIT statement. The UNDEFINEDFILE condition is raised in the normal way when an implicit OPEN error occurs. Any statement errors that occur cause the I/O operation to be terminated and an ERROR condition raised; in this case, the ERROR condition is raised when the I/O statement is executed, and the Event Variable is unchanged.

CONSECUTIVE

Creation of CONSECUTIVE Data Sets

There are no special considerations with regard to the data set specification beyond those already indicated under "File Attributes and the DD Statement." Note that LOCATE is only valid for BUFFERED files.

Accessing of CONSECUTIVE Data Sets

After a CONSECUTIVE data set is created, it may, with one exception, be opened only for INPUT or UPDATE. The exception is that it may be opened for OUTPUT using the parameter DISP=MOD, in which case the new records are added at the end of the existing records in the data set. In any other circumstances, opening a CONSECUTIVE data set for OUTPUT will result in the data set

being overwritten. Reading of such a data set may be either forwards or backwards if it resides on tape; to read backwards, the file must be opened with the BACKWARDS attribute. If a file is first read (or written) forwards, and then read backwards, the LEAVE option in the ENVIRONMENT attribute should be specified, to prevent the normal rewind at close time, or when volume switching occurs on a multivolume data set.

Replacement of Records: When a CONSECUTIVE data set is opened for SEQUENTIAL UPDATE, the only I/O statements which may be used, apart from OPEN and CLOSE, are READ and REWRITE. READ SET and REWRITE without options can only be used for BUFFERED files. A REWRITE statement may not be issued until a READ statement has been issued; and a REWRITE statement will always update the last record read. It is not permissible to have intervening READ statements between a READ and a REWRITE referring to the same record in the data set. For example:

```
1  READ FILE(F) INTO(A);
     .
     .
     .
5  READ FILE(F) INTO(B);
     .
     .
     .
9  REWRITE FILE(F) FROM(C);
```

In the above example, statement number (9) updates the record which was read at statement number (5). The record which was read at statement number (1) can never be updated after statement number (5) has been executed.

Consider also the following:

```
1  READ FILE(F) INTO(A) EVENT(E1);
     .
     .
     .
5  READ FILE(F) INTO(B) EVENT(E2);
     .
     .
     .
9  WAIT (E1);
10 REWRITE FILE(F) FROM(A);
```

Statement number (10) would cause the ERROR condition to be raised, since between the REWRITE statement and its corresponding READ statement (1), there is an intervening READ statement (5).

When blocked records are used with UPDATE files, it must be remembered that if one record in a block is rewritten, then all the records in that block are rewritten. The user must take this into account when updating records obtained by means of READ....SET.

INDEXED

Creation of INDEXED Data Sets

Special DD statement requirements are as follows:

One, two, or three DD statements are required. The number depends on the sizes and relative positions of the three parts of the INDEXED data set, namely the prime data area, the overflow area, and the index area. The SPACE and DSNAME fields of the DD card have special formats for INDEXED data sets.

The following DCB subparameters must be specified:

```
KEYLEN=n
DSORG=IS
```

The following DCB subparameters may be specified:

```
RKP=n
OPTCD=L|M|Y|I (or any combination)
NTM=n
CYLOFL=n
```

Space for the data set must be allocated in cylinders, unless the absolute track (ABSTR) technique is used. If ABSTR is used, the number of tracks must be equivalent to an integral number of cylinders, and the first track used must be the first track on a cylinder which is not the first cylinder on a volume.

Programming considerations are as follows:

For blocked INDEXED data sets, the DCB subparameters LRECL and BLKSIZE must be specified as follows:

Without embedded key (i.e., RKP=0), LRECL = size of data area + size of key

With embedded key (i.e., RKP≠0), LRECL = size of data area

In both cases, BLKSIZE = LRECL x blocking factor.

Creation of an INDEXED data set is only available in the SEQUENTIAL OUTPUT mode. The EVENT option is not support-

ed for SEQUENTIAL access to INDEXED data sets. The LOCATE statement is only valid for BUFFERED files.

When creating INDEXED data sets, the values of the keys presented must be in ascending collating sequence order; i.e., the binary value of each successive key must be greater by at least 1 than that of the previously presented key. Therefore, duplicate keys cannot be added to the data set.

Data set specifications made during creation may not be altered during subsequent processing of the data set; e.g. KEYLEN is fixed during creation. Also provision for overflow areas, whether cylinder or independent, must be made when the data set is created. If the prime data area is not filled during creation, then, with one exception, the unused area is not usable for overflow records, nor for any records subsequently added during direct access. The exception is that the unfilled portion of the last track used may be filled if required. It is possible to reserve space for later use within the prime data area by writing "dummy" records during creation: provided that the initial byte of the record has the value (8) '1'B, and that the option OPTCD=L has been specified, these dummy records can subsequently be replaced by actual records with the same key.

Refer to the publication IBM System/360 Operating System, Job Control Language, and IBM System/360 Operating System, Job Control Language Charts, for more detailed discussion of the above considerations.

Accessing of INDEXED Data Sets

After an INDEXED data set is created, it may be reopened for INPUT or UPDATE, for either SEQUENTIAL or DIRECT access. If opened for DIRECT access, records may then be added and deleted. It should be noted that once created, an INDEXED data set may not be opened for OUTPUT to add further records; it must be opened for UPDATE.

The user is cautioned that only one DIRECT UPDATE file that adds records to an INDEXED data set should be open at any one time. If two files are simultaneously open for SEQUENTIAL and DIRECT processing of the same INDEXED data set, then the following records might not be accessed by the SEQUENTIAL file:

1. Records added to the end of the data set.
2. Records written directly in the overflow area of the data set.
3. Records written on the overflow area when forced out of the prime data area by a record being added to this data area.

SEQUENTIAL Access

INDEXED data sets accessed in the SEQUENTIAL fashion may be opened for either INPUT or UPDATE, once created. Sequential access is in the order defined by the values of the recorded key, i.e., ascending collating sequence order. Records are retrieved in this order irrespective of the sequence in which they were added to the data set. Those records which have been deleted are not retrieved, and may be overwritten or lost when other records are added to the data set.

It is possible to use the KEY option on READ statements to cause repositioning, in either a forward or backward direction, among the records being sequentially accessed either for INPUT or UPDATE. This facility is available only if the KEYED attribute is specified. Repositioning may be either to a specified record or, if GENKEY has been specified in the ENVIRONMENT option list, to the first non-dummy record of a particular key class. In either case a record is retrieved followed by the next higher keyed record if the subsequent READ does not have a KEY option. If the specified KEY is not found in the data set, the KEY condition is raised. A subsequent READ without the KEY option will retrieve the first record in the data set.

The EVENT option is not supported for SEQUENTIAL access to INDEXED data sets. READ SET and REWRITE without the FROM option are only valid for BUFFERED files.

When opened for SEQUENTIAL UPDATE, only the statements READ and REWRITE (other than OPEN and CLOSE) may apply to the file. The order of operation must be READ then REWRITE, but not every record read need be rewritten. When blocked records are used with UPDATE files, it must be remembered that if one record in a block is rewritten, then all the records in that block are rewritten. The user must take this into account when updating records obtained by means of READ....SET. Records may not be added to the data set when in this mode, but they may be deleted simply by setting the initial character of the record to the delete-code (i.e. a character of the value (8)'1'B) and then rewriting it. This can also be done by using the DELETE statement

which marks as deleted the last record read. Records which are marked "deleted" are not made available during sequential input.

If blocked records are to be deleted, RKP must be greater than 0. This is necessary because a deleted record is indicated by (8)'1'B in the first byte and, for blocked records, this byte is part of the recorded key; hence if RKP=0, the access of subsequent records in the block may be impossible.

After several usages of an INDEXED data set in which records have been added or deleted, it may become desirable to copy the data set from one volume to another in order to purge the data set of records marked deleted, but still physically present, and to collect records from overflow areas into the prime data areas. Such reorganization will allow more efficient future access to the data set.

DIRECT Access

INDEXED data sets accessed in the DIRECT fashion may be opened for INPUT or UPDATE, once created. A DIRECT UPDATE file may have records added, deleted, or replaced, as well as retrieved.

Retrieval of Records: Records marked "deleted" are not made available.

Addition of Records: If the key is unique, the record is added to the data set, either in the prime area (possibly replacing one marked deleted), or in an overflow area; indexes are automatically updated as necessary. Duplicate keys cause the KEY condition to be raised, as will failure to find available space for the addition of records.

If a new record is added to a data set whose overflow areas are already full, a record will be irretrievably lost. The position of the new record, in relation to the existing records on the track, will determine whether it is the new record or an existing one which is lost. If the new record would follow the last existing record on the track, the new record will be lost. Otherwise, the last existing record on the track will be lost. In either case, the KEY condition will be raised.

Deletion of Records: The specified record is located, marked deleted, and rewritten into the data set. Note that deletion of records is only available if the DCB parameter of the DD statement specified OPTCD=L.

If blocked records are to be deleted, RKP must be specified as greater than zero. Otherwise, owing to the technique by which records are marked as deleted -- by setting the initial byte of the record, which for blocked records includes the recorded key, to the value (8)'1'B -- the locating of subsequent records in the data set may be impossible.

Replacement of Records: The specified record is overwritten by the replacement record. Records may be replaced without having been read, unless the data set contains blocked records, in which case the sequence must be READ then REWRITE. In the case of blocked records, a WRITE, DELETE, or READ statement may not be issued while a previous READ statement is still outstanding; that is, before its corresponding REWRITE statement has been issued. The EXCLUSIVE attribute should be specified to synchronize the READ-REWRITE cycle and WRITE statements.

REGIONAL

The three types of REGIONAL data set organization are treated together in the following paragraphs.

Creation of REGIONAL Data Sets

Special DD statement requirements are as follows:

The following DCB subparameters must be specified:

```
KEYLEN=n (except for REGIONAL(1))
DSORG=DA
```

The following DCB subparameters may be specified:

```
LIMCT=n (except for REGIONAL(1))
```

Space allocation (via the DD statement parameter SPACE) may be in terms of records, tracks, or cylinders; use of the ABSTR subparameter is permitted also. When allocating space for a REGIONAL data set, it is possible to request secondary allocation, such that when the initial allocation is filled, a second allocation is automatically made; further allocations are also made when necessary. (Secondary allocation is only available when the data set is being created, and then only when opened for SEQUENTIAL OUTPUT; opening for DIRECT OUTPUT causes the initial

allocation, only, to be "formatted" for direct placement of records.)

Programming considerations are as follows:

Creation of REGIONAL data sets may be performed with either DIRECT or SEQUENTIAL access; the mode must be OUTPUT.

DIRECT Creation: Upon opening the data set, it is initialized in accordance with the record format employed.

F-format (REGIONAL(1), (2), and (3)): Each track within the initial allocation is initialized with dummy records (and keys, if (2) or (3)).

U- and V-format (REGIONAL(3)): Eacn track within the initial allocation is "cleared" by writing a special record termed the capacity record. This record is set to indicate that the complete track is available for the addition of records.

Creation employing the DIRECT access technique permits records to be placed anywhere within the data set, in any order. Subsequent retrieval of these records in a SEQUENTIAL fashion, however, will access the records in a physically sequential order.

When there is a danger of more than one record having the same recorded key, the DCB subparameter KEYLEN should be increased so that the region number is also recorded; provided that the region number of the two records is not also the same, there will then be no retrieval problem.

SEQUENTIAL Creation: Keyed records are presented for addition to the data set according to the following rules.

REGIONAL(1) and (2): The positioning value of the key must increment by at least 1 for each successive key; if the value increments by a value greater than 1, say n, then n - 1 dummy records are added to the data set before the given record is added.

REGIONAL(3): The positioning value of the key remains constant for addition of records to a given track. When switching to a new track is desired, the positioning value is incremented by at least 1. If the value increments by a value greater than 1, say n, then n - 1 intervening tracks are cleared (if formats U or V) or written with dummy records (if format F, in which case the current track is completed with dummy records).

If a LOCATE statement was used in the addition of KEYED records to any type of REGIONAL data set, then, if the RECORD condition is raised, the key value presented at subsequent operations must not be less than the current one.

Creation employing the SEQUENTIAL access techniques causes records to be added to the data set in a physically sequential order, according to ascending relative record or track value. Subsequent retrieval of the data set in a SEQUENTIAL fashion will access these records in the same order (as well as any records which may have been inserted between them). When a sequentially created REGIONAL data set is closed, the current extent is completed with dummy records, or the remaining tracks are cleared.

The user is cautioned that, for REGIONAL(3) data sets, the addition of duplicate keys is not detected; subsequent DIRECT access will read the first record with the required key on the given track (or tracks, if "extended-search" has been requested, i.e., if the DD subparameter LIMCT is absent or specifies a value greater than one).

Also note that when a given track of a REGIONAL(3) data set becomes filled by sequential addition of records with the same region number (relative track value), the current track number is automatically incremented by 1. If an attempt is then made to add another record with the same region number, the KEY condition is raised (key sequence error) and the excess record is not added to the data set.

Accessing of REGIONAL Data Sets

After a REGIONAL data set is created, it may be reopened for INPUT or UPDATE, SEQUENTIAL or DIRECT. If opened for DIRECT access, records may then be added or deleted. It should be noted that once created, a REGIONAL data set may not be opened for OUTPUT to add further records; it must be opened for UPDATE.

SEQUENTIAL Access

REGIONAL data sets accessed in the SEQUENTIAL fashion may be opened for either INPUT or UPDATE, once created. Sequential

access is in the order of ascending relative record (REGIONAL(1) and (2)) or track value; the value of the keys does not affect the order of retrieval. It should be noted that all records within a REGIONAL(1) data set, whether dummy or actual, are retrieved in sequence; if dummy records are present, the user should be prepared to recognize them if necessary. Deleted (dummy) records in REGIONAL(2) or (3) data sets are not made available.

It is not possible to employ the KEY option when retrieving REGIONAL data sets sequentially.

Replacement of Records: When a REGIONAL data set is opened for SEQUENTIAL UPDATE, the only I/O statements which may be used, apart from OPEN and CLOSE, are READ and REWRITE. A REWRITE statement may not be issued until a READ statement has been issued; and a REWRITE statement will always update the last record read. It is not permissible to have intervening READ statements between a READ and a REWRITE referring to the same record in the data set. For example:

```
1  READ FILE(F) INTO(A);
   .
   .
   .
5  READ FILE(F) INTO(B);
   .
   .
   .
9  REWRITE FILE(F) FROM(C);
```

In the above example, statement number (9) updates the record which was read at statement number (5). The record which was read at statement number (1) can never be updated after statement number (5) has been executed.

Consider also the following:

```
1  READ FILE(F) INTO(A) EVENT(E1);
   .
   .
   .
5  READ FILE(F) INTO(B) EVENT(E2);
   .
   .
   .
9  WAIT (E1);
10 REWRITE FILE(F) FROM(A);
```

Statement number (10) would cause the ERROR condition to be raised, since between the REWRITE statement and its corresponding READ statement (1), there is an intervening READ statement (5).

DIRECT Access

REGIONAL data sets accessed in the DIRECT fashion may be opened for either INPUT or UPDATE, once created. An UPDATE file may have records added, deleted, or replaced, as well as retrieved. Facilities available for DIRECT access vary according to the REGIONAL type, and the record format in the case of REGIONAL(3).

The user is cautioned that, for REGIONAL(1) data sets, records added with a source key specifying the relative record position of an existing record will overwrite the existing record without notification. Also, for REGIONAL(2) and (3), the use of the "extended search" feature may cause a record to be added beyond the track containing the specified region (REGIONAL(2)), or beyond the specified track (REGIONAL(3)); this can be prevented by use of the DCB subparameter LIMCT (see the paragraph "Extended Search Limit," earlier in this chapter). If LIMCT is specified as 1, insertion is confined to the track containing the specified record (REGIONAL(2)) or the specified track (REGIONAL(3)). Another point to note is that it is quite possible to add, in a REGIONAL(2) or (3) data set, a new record whose recorded key is the same as that of an existing record. In such a case, the record which is physically farthest from the beginning of the data set may never be retrieved by DIRECT access.

Retrieval of Records:

REGIONAL(1):	All records, whether dummy or actual, may be retrieved.
REGIONAL(2):	Records marked dummy cannot be retrieved.
REGIONAL(3):	Records marked dummy cannot be retrieved.

Addition of Records:

REGIONAL(1):	"Addition" is actually replacement of existing records, whether dummy or actual (no condition is raised in either case).
REGIONAL(2):	"Addition" is actually replacement of dummy records on the track containing the relative record number specified (or subsequent tracks, if extended-search has been requested, i.e., LIMCT > 1 or absent).
REGIONAL(3):	If F-format records, as for

REGIONAL(2). If U- or V-format records, records are added to available space on specified track (or subsequent tracks, if extended-search has been requested, i.e., LIMCT > 1 or absent).

Deletion of Records:

REGIONAL(1): The specified record is overwritten with a dummy record; the space may be reused.

REGIONAL(2): The specified record is overwritten with a dummy record, the key being rewritten as a dummy key; the space may be reused.

REGIONAL(3): If F-format records, as for REGIONAL(2). If U- or V-format records, the record is overwritten with a dummy record, the key being rewritten as a dummy key; the space may not be reused.

Replacement of Records:

REGIONAL(1): The specified record, whether a dummy or an actual record, is rewritten.

REGIONAL(2): The specified record is rewritten; a record with the specified key must exist.

REGIONAL(3): As for REGIONAL(2), for all record formats.

If a REGIONAL file has the EXCLUSIVE attribute, then, in order to lock a particular record, different tasks must use the same region number to refer to the record.

PROCESSING MODES IN RECORD I/O

In RECORD I/O, the user has the choice of processing a record in a work area or in a buffer. This choice is exercised by selecting one of two processing modes: move mode or locate mode. There are advantages in processing an entire file with one mode, but it is permitted to use a combination of both modes in the same file.

MOVE MODE

Characteristics

Input: Data is moved (possibly via a buffer) from a file to a work area, where it is processed. (The work area is the record variable.)

Output: Data is moved from the work area to a file, possibly via a buffer.

Move mode may be simpler to use than locate mode, as there are no buffer alignment problems. In cases where there are numerous references to the contents of a record, it can also be faster, as it does not have the indirect addressing overhead of locate mode.

Language Forms

The basic statements used in move mode are:

```
READ  FILE  (filename)  INTO  (record variable);
WRITE FILE  (filename)  FROM  (record variable);
REWRITE FILE (filename) FROM  (record variable);
```

File Attributes

The full set of RECORD-oriented I/O file attributes is permitted.

LOCATE MODE

Characteristics

Input: Data is moved from a file to an input buffer, where it is processed.

Output: The data record is constructed by the user in an output buffer. The buffer is then transmitted to a file.

The locate mode input statement sets the address of the record (in the buffer) in a pointer variable. The record can then be identified for processing by using this pointer in conjunction with a based varia-

ble that defines the structure of the record. As many based variables as are required can be declared for a file; the record can be associated with any of them.

Locate mode can be used to read self-defining records. These are records where information in one part of the record is used to indicate the structure of the rest of the record. This information could be, for example, a count of the number of repetitions of a subfield, or a code identifying which one of a class of structures should be used to interpret the record.

Locate mode should also provide faster execution, as there is no need to move data from a buffer to a work area, and may have a smaller storage requirement, because no storage is needed for separate work areas.

Whenever a statement that causes data transmission to or from a file is executed, any logical record accessed previously by a locate mode statement is no longer available for processing.

Language Forms

The basic statements used in locate mode are:

```
READ FILE (filename) SET (pointer
  variable);
LOCATE based variable FILE (filename)
  [SET (pointer variable);]
REWRITE FILE (filename);
```

File Attributes

Locate mode can only be used with SEQUENTIAL BUFFERED files. The KEYED and ENVIRONMENT attributes may be specified; the other file attributes permitted depend on the statement used:

Statement	File Attribute
READ (with SET)	INPUT/UPDATE
LOCATE	OUTPUT
REWRITE (without FROM)	UPDATE

If any of these attributes is not explicitly or implicitly declared, or if any other attribute is present, the ERROR condition is raised.

SYSTEM REQUIREMENTS

To operate successfully, the PL/I (F) Compiler and its associated load modules must have the required amount of main storage and input/output devices, as specified in the following paragraphs.

MINIMUM SYSTEM/360 REQUIREMENTS FOR THE (F) COMPILER

System/360 operating system operates in a device-independent environment. In particular, the (F) Compiler may operate with different combinations of devices. However, certain restrictions should be noted.

The PL/I (F) Compiler requires at least a System/360 Model 30 with a minimum of 64K bytes of storage. At least 45,056 (44K) bytes should be allocated in the SIZE option. If additional storage is available and allocated, the compiler will in general run more efficiently and provide a faster compilation.

At least one direct access device must be used for residence of the operating system, and for the use of the SYSUT1 data set. The same volume may be used for both purposes. The use of certain source program and operating system features requires a storage minimum greater than that given above. For example, PL/I tasking uses the MVT system, which requires at least 256K bytes of storage.

Instruction Sets

The standard, decimal, and floating-point instruction sets are required by the compiler and object programs, irrespective of declared data attributes.

Timing Clock

The timer feature is required in order to provide elapsed time of compilation, or to support the timer built-in function. In the absence of the clock, no time information will be printed out.

Printer Character Sets

For flexibility in character sets and graphic representation, the following types of chains, trains, or type bars are available for use with output printers.

1403 Printer Chains or Trains

Character Set	Chain Type
48-character set	HN
BCD	HN
60-character set	PN or QN

1443 Printer Type Bars

Character Set	Bar Type
48-character set	52H
BCD	52H
60-character set	63-character bar

Operator's Console Character Sets

The operator's console should have the IBM 1052 printer keyboard Model 7 with dual case printing element, feature code 9572.

SYSTEM/360 OPERATING SYSTEM REQUIREMENTS

Primary Control Program of the Operating System

The primary control program of the operating system provides all sequential scheduling features of the job control language as specified in the publications IBM System/360 Operating System, Job Control Language, and IBM System/360 Operating System, Operator's Guide. It affords data management capability and contains a supervisor which provides for:

Efficient overlapping of central processing unit operations and input/output channel activity

Error checking and standard input/output error recovery procedures

Supervision and processing of interruptions

Supervision of requests for various services provided by the system

This control program provides for a single input job stream and the sequential processing of job steps through single task operations.

Multiprogramming with a Fixed Number of Tasks (MFT)

This is an extension of the primary control program. It provides for concurrent control of up to four independent tasks representing separate jobs. Each job occupies its own fixed partition of main storage.

Multiprogramming with a Variable Number of Tasks (MVT)

This provides, in addition to the facilities provided by the primary control program:

Priority scheduling of jobs submitted from single or multiple job streams

Concurrent scheduling and execution of up to 15 separately protected jobs

The MVT system is used in PL/I tasking .

COMPILER SUPPORT

After the compiler initially receives control from the calling program of the operating system by means of a supervisor-assisted linkage, communication is maintained with the operating system through the compiler control routines.

The functions provided by the primary control program of the operating system, together with the data management routines BSAM and QSAM, are required by the compiler. In addition, BPAM data management routines are required by the compile-time processor if the INCLUDE statement is used.

Object-time interfaces with the operating system control program are made through the PL/I Library routines.

The facility for providing details of the time taken for each compilation must be specified at system generation time through the control program options (see "Object Program Support"). On completing a compilation, a message will be generated giving the actual time taken. Under MVT, the time given is CPU time only, as I/O time cannot be measured.

OBJECT PROGRAM SUPPORT

Control Program Options

Figure 21 shows the control program options which may be added to the primary control program in order to provide greater performance and/or programming flexibility for the various features of PL/I. These exclude input/output features.

PL/I Feature	Control Program Option*
WAIT	Multiple WAIT
Tasking	MVT
TIME, DATE	Timing: A. Time
TIME, DATE DELAY	Timing: B. Interval timing

* See the publication IBM System/360 Operating System, Storage Estimates for further details of these options.

Figure 21. Control Program Options

Usage of Data Management Access Methods for STREAM I/O

STREAM I/O within PL/I is supported by the queued sequential access method (QSAM). In order to conserve space and time, the GET and PUT macros are used in the locate mode whenever possible. Automatic data-transmission computing-time overlap is provided, as are blocking-deblocking functions. In order to achieve an efficient degree of I/O overlap, two buffers are normally allocated to each data set when it is opened. The number of buffers allocated is, however, under the control of the DD statement, or the BUFFERS option of the ENVIRONMENT attribute.

Usage of Data Management Access Methods for RECORD I/O

The access method employed depends upon the following factors:

Organization	Access	Mode	Buffering	Record Format	Access Method	Notes on Use of Access Method
CONSECUTIVE	SEQUENTIAL	INPUT UPDATE OUTPUT	BUFFERED	All	QSAM	Locate mode (except paper tape)
			UNBUFFERED	F|U|V	BSAM	-
INDEXED	SEQUENTIAL	INPUT UPDATE	BUFFERED or UNBUFFERED	F|FB	QISAM	Scan mode ESETL/SETL
		OUTPUT				Load mode
	DIRECT	INPUT UPDATE	-		BISAM	-
REGIONAL(1), (2), and (3)	SEQUENTIAL	INPUT UPDATE	BUFFERED or UNBUFFERED	F (REGIONAL (1) and (2))	BSAM	QSAM (under certain circumstances)
		OUTPUT				BDAM load mode
	DIRECT	INPUT OUTPUT UPDATE	-	F|U|V (REGIONAL(3))	BDAM	REGIONAL(1) * Relative record without keys REGIONAL(2) * Relative record with keys REGIONAL(3) * Relative track with keys
* OUTPUT causes data set to be formatted using BSAM (BDAM load mode) at open time						

•Figure 22. Usage of Data Management Access Methods for RECORD-Oriented I/O

Organization
Record format
File attributes

The various combinations of the above, and the resultant usage of data management, are illustrated in Figure 22.

PROGRAM SEGMENTATION

The MAIN Option

Although external procedures must be compiled separately, several compilations may be processed by the linkage editor to form a single load module. In order to execute such a load module, the external procedure of one of the compilations must possess the option MAIN.

If more than one compilation has this option, then the linkage editor will cause control to be passed to the first of these compilations to appear in its input stream. Note that this is a function of the linkage editor itself and is not a formal PL/I facility.

An attempt to execute a PL/I program in which no procedure has been given the MAIN option will result in termination of execution, accompanied by a message on the console or system output listing.

Communication Between Separate Compilations

Communication between separate compilations can be achieved by means of EXTERNAL declarations or by passing arguments in CALL statements or function references. It should be noted that the PL/I language makes the following restrictions:

> AUTOMATIC variables cannot be EXTERNAL (but may be passed as arguments)
>
> Descriptions of an external variable in separate compilations must not be contradictory. If they are, the compiler will not be able to detect this.

The linkage editor will load the INITIAL value of a STATIC EXTERNAL variable from the first object module encountered in its input stream which declared the variable.

Estimation of Program Size

In order to estimate with any accuracy the maximum overall storage requirements for a PL/I program, several factors must be taken into account. The information provided by the compiler and the linkage editor is a guide only to the space required by the executable control section and the STATIC INTERNAL control section in the load module. Several other items must be added to this to obtain a realistic estimate of the maximum object-time requirements. These include the following:

1. Pseudo-Register Vector (PRV). The overall length of the PRV is contained in the linkage editor module map, which is produced when the MAP option is specified. The maximum size of a PRV is 4096 bytes. (Refer to Appendix C of this publication for a more detailed description of pseudo-register vectors.) The primary workspace requirement for the library is approximately 1100 bytes, which should be added to the PRV requirement, plus 512 bytes if optimization level OPT=1 has been chosen.

2. Interrupt and Error Handling. For each PL/I interrupt, 1100 bytes are required for library workspace. In addition, library modules which are dynamically invoked as a result of interrupts may require up to 2600 bytes (irrespective of the number of interrupts). All storage required for interrupt and error handling is released when the interrupt has been cleared.

3. Dynamic Storage Areas (DSAs). A general guide to the length of each DSA is to add 200 bytes per active block to the space required for the AUTOMATIC variables declared within the block and their dope vectors. This is only a guideline; there may be considerable variations between different programs.

4. CONTROLLED Variables. Each allocated generation of each CONTROLLED variable will require 12 bytes in addition to the space required for the contents of the variable and its dope vector.

5. File Manipulation. The main factors to be considered in estimating the storage required when opening and closing files are as follows:

   a. File Control Block (FCB). Approximately 150 bytes for CONSECUTIVE and REGIONAL data set organizations; approximately 300 bytes for INDEXED organization.

Organization	PL/I Attributes	OPEN* (bytes)		CLOSE* (bytes)		NORMAL* (bytes)	
		Non-Tasking	Tasking	Non-Tasking	Tasking	Non-Tasking	Tasking
CONSECUTIVE	STREAM	1900	1900	1300	1700	-	-
	SEQUENTIAL BUFFERED	2400	2400	2400	2800	1100	1100
	SEQ.BUFF.(spanned input)	1900	1900	1900	2300	600	600
	SEQ.BUFF.(spanned output)	1800	1800	1800	2200	500	500
	SEQ.UNBUFFERED	5100	5100	5100	5500	3800	3800
INDEXED	SEQUENTIAL	3600	3600	3600	4000	2300	2300
	DIRECT	3100	3900	3100	4300	1800	2600
REGIONAL	SEQUENTIAL INPUT/UPDATE	5100	5100	5100	5500	3800	3800
	SEQUENTIAL OUTPUT	3900	3900	3900	4300	2600	2600
	DIRECT	3100	3900	3100	4300	1800	2600

* These figures show the extra storage required by linked PL/I library I/O modules in both non-tasking and tasking environments. The first two columns show the maximum space occupied by the library modules during the opening of a file with the given attributes. The second two columns show the maximum space occupied during the closing of the same file. The third two columns show the space occupied by the library modules while the file is open. If several files with the same attributes are processed, then the storage requirements are unaffected. If, however, another file with different attributes is open when a file is opened or closed, then the figure for that file must be added to the figure in the OPEN and CLOSE column. For instance, if a SEQUENTIAL INDEXED file is open in a non-tasking environment at the time that a CONSECUTIVE SEQUENTIAL BUFFERED file is being closed, then the total storage required by the linked modules will be 2300+2400=4700 bytes. The figures do not include the storage required by data management.

● Figure 23. Storage Required by PL/I Library Modules when Opening and Closing Files

b. Library modules. The greatest storage requirement generally occurs when a file is opened or closed. Figure 23 shows the storage required at these times. In all cases, the space required by data management must be added to the figures given.

c. Buffers. In addition to the factors given above, space must be allowed for buffers or I/O control blocks. This requirement can become very large if large blocksizes are used.

All space required by files is released and becomes reusable after the file has been closed.

NON-PL/I MODULES IN PL/I PROGRAMS

Combination of PL/I With Other Languages

The combination of modules written in other languages with modules written in PL/I is possible only in a rather limited way, since most other languages are not structured to support the advanced features of PL/I. In order to achieve more than a limited use, the user would require an intimate knowledge of the object code structure and requirements of a PL/I program. Appendix C is provided as an introduction to the subject. Appendix D of the present manual contains an example illustrating one way in which an assembler language program can be written in order to combine it with a PL/I procedure.

Variable-Length Argument List

When an assembler subroutine is called by a PL/I program, by means of a CALL statement or a function reference, a variable number of arguments can be passed. The compiler indicates the last argument by setting X'80' in its high-order byte.

This feature provides compatibility with COBOL and FORTRAN calling sequences. It cannot be used in an exclusively PL/I environment because PL/I does not allow a variable number of arguments.

PL/I Library Subroutines

PL/I object programs make use of a large number of subroutines. These subroutines are held in the PL/I library and are incorporated into the PL/I program by the linkage editor. Library modules not directly referenced by the compiled module, but which are selectively referenced by other required modules (e.g., modules within in the data conversion package) are specified by the compiler in ESD entries. These ESD entries, in conjunction with the linkage editor control statement, LIBRARY, are used to keep the program's storage requirements to a minimum. The form of the LIBRARY statement used specifies that the linkage editor is not obliged to resolve certain external references within a given module, unless the referends are already included within the module being edited. Those modules which require this feature use the LIBRARY control statement when they are edited into the PL/I library. Thus, when these modules are subsequently processed by the linkage editor when building a PL/I program, there will be no second-level search of the library. Library subroutines are automatically selected, and no user intervention is needed.

Other library modules are used on a transient basis, and are dynamically loaded during program execution. These modules reside within the data set SYS1.LINKLIB. Modules of this type include those required for opening and closing files, RECORD I/O data management interface modules, and object-program error-handling routines and message tables.

CONDITIONAL EXECUTION OF JOB STEPS

It is possible for the programmer to specify that a particular job step shall not be executed if any of certain conditions are satisfied by return codes passed from previously executed job steps. A return code can be set during execution of a PL/I program.

Setting Conditions

The conditions under which a job step is to be bypassed are set out in the COND parameter of the EXEC statement for the step. For example, if the EXEC statement for a job step, STEPC, contained the parameter

```
COND=((22,GT,STEPA),(59,NE,STEPB))
```

then, if 22 were greater than the return code passed from STEPA, or if 59 were not equal to the return code passed from STEPB, the job step STEPC would not be executed. For further information on the use of the COND parameter, see the publication IBM System/360 Operating System, Job Control Language.

Return Code Setting by PL/I Object Program

To use this facility in a PL/I program, it is first necessary to declare the library module entry point IHESARC:

```
DCL IHESARC ENTRY(BINARY FIXED);
```

Once IHESARC has been declared, the return code can be set at any point in the program by the following CALL statement:

```
CALL IHESARC(expression);
```

The routine may be called as often as required. Each time, the expression will be evaluated and its value will be saved. On normal termination, the return code will be set to the latest value, modulo 4096. (On abnormal termination, the complete job would be terminated.)

If the facility is not used, a return code of zero will be passed.

CHECKPOINT/RESTART

Compatibility with Release 11 Checkpoint/Restart

The major differences between Release 17 and the earlier Release 11 version of Checkpoint/Restart are the increased number of Checkpoint/Restart features and the changed method the programmer must use to initiate a programmer-deferred checkpoint restart, or step restart.

Introduction

When a job step terminates because of, for example, an I/O error or a machine check, the programmer may require that processing begins again from a point within the job step or, alternatively, from the beginning of the job step. This can be done by means of the IBM System/360 Operating System Checkpoint/Restart facility.

Checkpoint/Restart is supported by the MVT and PCP supervisors, and provides the following features:

Automatic step restart
Automatic checkpoint restart
Programmer-deferred step restart
Programmer-deferred checkpoint restart

If automatic checkpoint restart or automatic step restart is specified when using MVT, MSGLEVEL=1 must be specified in the JOB statement.

Step Restarts

Programmer-deferred step restart enables the programmer to resubmit a multistep job and cause any number of steps to be bypassed, until the required step is reached, at which point execution commences. Automatic step restart causes the excution of a job step which terminates abnormally to be recommenced from the orginal starting point.

Checkpoint Restarts

Programmer-deferred checkpoint restart is the technique used if the job is to be discontinued, and resubmitted for checkpoint restart by the user at a later time; automatic checkpoint restart is the technique used if a restart at the last checkpoint taken is to be performed during the job immediately that the system recognizes that the job step has been terminated abnormally. Both checkpoint restart techniques require checkpoint data, in order to carry out the restart operation. Checkpoint data consists of all the information about a program at a point in its execution which can be used to recommence execution from that point. Multiple checkpoints can be preserved, or, alternatively, only a single checkpoint may be required, in which case each checkpoint is overwritten by a subsequent checkpoint. Deferred checkpoint restart can be performed at any of these checkpoints, if multiple checkpoints are preserved, or from the latest checkpoint, if it is the only checkpoint that is preserved. Automatic checkpoint restart will use only the last checkpoint taken. To create a checkpoint from within a PL/I program, code the statement

```
CALL IHECKPT;
```

at a strategic point in the program such that any data sets used by the program can be repositioned by the restart program to the positions held at the time the checkpoint was taken.

Note: The CALL IHECKPT statement must not be executed in a multitasking environment.

When the CALL IHECKPT statement is executed, a checkpoint data set is created, for which a unique name or checkid is generated by the system. The checkid is displayed on the operator's console for each checkpoint taken. The checkid for a particular checkpoint data set which is to be used to restart execution of the program should be noted, and later used to identify the data set. The checkpoint data set is sequential only, and may not be a member of a partioned data set.

A DD statement with the ddname SYSCHK must be provided in the job stream for each step which produces checkpoints. This DD statement contains the specifications for the checkpoint data set. The parameters for this statement are given in Figure 24.

The SYSCHK DD DISP parameter depends on whether there is a single checkpoint or whether multiple checkpoints are required. For a single checkpoint code DISP=(NEW,KEEP), for multiple checkpoints code DISP=(MOD,KEEP).

Single Checkpoint

With a single checkpoint, the information stored at a checkpoint overwrites the information stored by any previous checkpoint; thus only the latest checkpoint information is available.

Multiple Checkpoints

The information for a checkpoint is stored after the information for a previous checkpoint; thus all the checkpoints taken are available for a restart.

The other checkpoint/restart techniques mentioned, automatic step restart and programmer-deferred step restart, do not require either the CALL IHECKPT or SYSCHK DD statements. (Automatic step restart causes a step to be reexecuted immediately if it is terminated abnormally; programmer-deferred step restart permits the job to be resubmitted, but any job steps preceding the step to be reexecuted are bypassed, permitting the immediate restart of the step.)

Parameter	Tape, Drum, Disk	
	Single checkpoint	Multiple checkpoints
DSNAME=	Any name	
VOLUME=	Volume number or other reference	
UNIT=	2400, 2400-2, 2301, 2311, 2314, 2321 etc.	
DISP=	(NEW,KEEP)	(MOD,KEEP)
SPACE=	Provided by formulas for direct-access devices	
DCB=	(TRTCH=C) if UNIT=2400-2	

* The space allocated on a direct-access device must be sufficient for the job-step requirements together with the system blocks. In effect, this means that it might have to be as much as one and a quarter times main storage or region size. For example, a 256K machine or region might require 10 cylinders on a 2311 disk pack. The amount required is derived from the formulas provided in IBM System/360 Operating System: Storage Estimates.

A secondary quantity can appear in the SPACE parameter of the SYSCHK DD statement. However, if it is specified, it is not used.

● Figure 24. Basic Parameters for the SYSCHK DD Statements

Specifying Checkpoint Restart

Automatic checkpoint restart is specified implicitly by the execution of a CALL IHECKPT statement.

The remaining checkpoint restart features are controlled from the RD (Restart Definition) parameter, which can be specified in either a JOB statement or an EXEC statement. The settings of the RD parameter are listed below:

RD [.procstep] = {R|NC|NR|RNC}

where:

R (restart)	requests automatic step restart.
NC (no checkpoint)	totally suppresses both the execution of the CALL IHECKPT statement, and the implicit automatic checkpoint restart.
NR (no restart)	Checkpoints may be written, but automatic checkpoint restart is inhibited. Specify this parameter for programmer-deferred checkpoint restart.
RNC (restart, but no checkpoints)	requests automatic step restart and totally suppresses the execution of the CALL IHECKPT statement.

Notes:

1. An RD parameter coded in a JOB statement overrides one coded in an EXEC statement.
2. If the RD=value is coded in an EXEC statement for a cataloged procedure, it applies to all the steps within the procedure. RD.procstep=value can be coded instead of RD=value; it then applies only to the specified procedure step. RD.procstep=value can be coded for each step in the cataloged procedure, in procedure step order.
3. The CALL IHECKPT statement causes a request for automatic checkpoint restart and overrides the request for automatic step restart, if the parameter RD=R has been specified.
4. RD=NC and RD=RNC are provided to suppress the execution of a CALL IHECKPT statement which is contained in a

program to be executed, but is not required for a particular job.

5. RD=NC and RD=NR have no effect on a step in which there is no CALL IHECKPT statement to be executed. In this case RD=RNC has the same effect as RD=R, specifying automatic step restart.

6. If no RD parameter is specified, automatic step restart is suppresed but an automatic checkpoint restart can occur if a checkpoint is taken.

7. The RD parameter is ignored when the MFT II supervisor is used.

Restarts

Automatic restarts are specified in the RD parameter described previously.

Programmer-deferred restarts are performed by resubmitting the deck containing the job control statements for the original job, with the JOB statement containing a RESTART parameter to specify that a restart is required, and how it is to be performed. The RESTART parameter specifies the step to be restarted, and, if a checkpoint restart is required, the checkpoint identity, i.e., checkid, of the checkpoint data set to be used. It is specified as follows:

RESTART= ({ stepname / stepname.procstep / * } [,checkid])

If the restart is a programmer-deferred step restart, the checkid information must not be specified, and the enclosing parentheses may be omitted. The stepname parameter identifies the step in the job to be restarted. The form stepname.procstep is used if the step is in a cataloged procedure. The form '*' can be used to indicate that the first step in the job is to be restarted. (The first step can be a cataloged procedure step.) If a programmer-deferred checkpoint restart is to be performed, the checkid information should be present and correspond to the checkid generated for the appropriate checkpoint data set by the system; this data set must be available, and specified to the job in a SYSCHK DD statement. The SYSCHK DD statement must appear after the JOB statement, but before the EXEC statement for the first step in the job. If JOBLIB DD statements are present, the SYSCHK DD statement appears after them. This SYSCHK DD statement must have the disposition parameter coded either as DISP=(OLD,KEEP) or as DISP=(OLD,PASS).

DATA SETS FOR PROGRAMMER-DEFERRED CHECKPOINT RESTARTS

To perform a programmer-deferred checkpoint restart, the data sets open at the time that the checkpoint was taken must be available for repositioning for use by the restarted step. All the DD statements used for the original job step must be present in the restart job step. A second SYSCHK DD statement, as orginally supplied, must also be present to specify the checkpoint data set for the restarted step.

Card Input when Using PCP

A restarted program may require the omission of a certain number of cards since the system cannot reposition a data set read from a card reader. The user should omit any cards which were read prior to the checkpoint from which the restart is to be performed.

SYSIN

A SYSIN data set will be repositioned correctly to the position it occupied when the checkpoint was taken. If the card input is used under a PCP system, it will not be repositioned. The position is achieved by bypassing the number of cards that were processed when the checkpoint was issued. Care must be taken if the data set needs to be changed; in this case the new data set must contain some dummy cards up to the point to be reached for repositioning, to replace any data cards which are taken out, modified, and replaced in the data set at or beyond the repositioning point.

SYSOUT - PCP

If a PCP system is used in which output for the SYSOUT data set is transmitted directly to a printer or card punch, a checkpoint taken while the output file is open causes the restarted program to produce output on SYSOUT from the point in execution at which the checkpoint was taken.

If the SYSOUT data set is written onto an intermediate tape, the tape is repositioned so that records written after the checkpoint are overwritten by records from the restarted program.

If the SYSOUT data set is on tape, and has been opened and closed prior to the checkpoint, and is then reopened by the restarted program, the original SYSOUT data set is preserved, and the new data is written immediately after it.

SYSOUT - MVT

If an MVT system is used, in which SYSOUT data sets are written onto a direct-access device for output by the system output writer, the output is produced up to the point at which the abnormal termination occurred. The output produced by the restarted program is written on a new SYSOUT data set which is opened at the point reached by the checkpoint, and contains records written from that point. This can cause duplication of output for the part of the processing which is repeated until the point of failure of the original job step is passed.

PRESERVATION OF DATA SETS

Temporary Data Sets

A job step which produces temporary data sets cannot be restarted. Temporary data sets with no dsname, or with a dsname of the form &&....., cannot be preserved. However, temporary data sets can be preserved for restarts if the conditional form of the DISP parameter is used, and providing that permanent dsnames have been specified. The conditional DISP parameter in the DD statements for temporary data sets, e.g., DISP=(NEW,PASS,KEEP), permits the data set to be passed (or deleted) at the normal end of the job step, or to be kept if the job step terminates abnormally.

Updated Data Sets

Direct-access UPDATE data sets which are updated by use of the REWRITE statement can prevent successful restarts because of changes to records taking place after the last checkpoint and before abnormal termination of the job step. A restart in such circumstances can cause an updated record to be re-updated, giving incorrect results. To avoid this situation, two identical data sets of the same file should be used, one for input and one for output, and processing performed using a READ statement to access a record from one of the data sets, and a WRITE statement to overwrite the corresponding record on the output data set with the updated record.

MULTITASKING

In PL/I multitasking, a number of tasks, each of which can have a different priority, can exist within the execution jobstep. Control is given to the task that has the highest priority and is not waiting for any reason. If there are two or more tasks with the same priority, control is given to the first one in the system queue. Thus multitasking allows the programmer to make fuller and more efficient use of machine time, by reducing the time during which the CPU is waiting or the I/O devices are not used. It allows him to exploit the full range of PL/I language features, and thus to arrange his program to be as flexible and effective as possible. Lastly, it allows the program to which it can be applied to be coded more easily.

This section describes multitasking with regard to:

1. System and implementation requirements.
2. Management of multitasking in planning and executing a program.

Multitasking is a sophisticated feature and will only give satisfactory results when used correctly. Provided the user understands the concepts involved and arranges his program accordingly, he will be able to obtain the full advantages of this facility.

MULTITASKING REQUIREMENTS

System/360 Requirements

A storage capacity of at least 256K bytes is required for multitasking.

Operating System Requirements

PL/I multitasking uses the MVT system. A PL/I program compiled with the TASK option and executed under the MFT or PCP systems will terminate abnormally. (For

details of this and other control programs see the section, 'System/360 Operating Requirements'). All the tasks exist within the same job step; the priorities for these tasks exist only within the job step and can be varied over a range that is determined by the job-step priority. PL/I is not concerned with establishing priorities between jobs or between job steps.

The minimum PL/I multitasking overhead in space and execution time, over and above that resulting from a single-task PL/I program, is:

Space: About 3500 bytes plus 2K bytes for each subtask attached

Time: About 70 milliseconds per task attached on a model 40

Programming Requirements

Compiler Level: Multitasking requires a Version 4 PL/I (F) compiler and a Version 4 PL/I library. Programs based on earlier versions can only be executed in a multitasking environment if recompiled with the TASK option (see below); even then subprograms in these programs may need rearranging to execute successfully.

Procedure Options: All programs and external compilations that are to be executed in a multitasking environment must have been compiled with the TASK option in the external PROCEDURE statement. For example:

```
X: PROC OPTIONS (MAIN, TASK);
```

A CALL statement with the EVENT, PRIORITY or TASK options also requires a multitasking environment; the TASK option should, therefore, be specified for the external procedure. If it is not, it is assumed by default.

Use of the TASK option causes the PL/I library multitasking modules, instead of the single-task modules, to be link-edited into the load module. The load module thus contains multitasking modules only when multitasking is required.

Combination with other languages: When a routine in System/360 assembler language is to be used in multitasking, the DSA obtained in it must be at least 108 bytes long. (The minimum for a DSA in a non-multitasking environment is 100 bytes). PL/I library routines used in multitasking must be those designed for multitasking, for example, IHEITH, IHETSA.

MULTITASKING MANAGEMENT

There are a number of topics that must be fully understood for successful multitasking management. These are:

1. Programming considerations
2. Use of priorities
3. I/O handling
4. Task termination.

These topics are discussed briefly here; further information is provided by the diagnostic messages.

Programming Considerations

A task may lose control under any of the following circumstances:

1. Termination
2. I/O operations
3. A task is attached with a higher priority than the current task
4. Use of the PRIORITY pseudo-variable
5. A higher-priority task may come out of a wait state or may complete I/O
6. Use of the DELAY or WAIT statements

Event Option: Only one task at a time can wait on a particular event. If a situation occurs where two or more tasks wait on the same event, a diagnostic message is provided. This kind of situation can easily occur; for example:

```
CALL X TASK (A);
CALL X TASK (B);
```

If no event is explicitly declared in X, then, if X contains a WAIT statement, both tasks may be waiting for the same event.

Variables: The scope of variables must be watched carefully in a multitasking program. A variable that has not been explicitly declared may be altered unpredictably when control passes from one task to another. For example:

```
COMPLETION (ASSIGNMENT) = '0'B;
A = 1;
CALL A_TEST(A) TASK(ONE);
WAIT (ASSIGNMENT);
A = 2;
CALL A_TEST (A) TASK(TWO);
```

```
      .
      .
      .
A_TEST; PROC(PARM);
  DCL VARIABLE FIXED BINARY;
  VARIABLE = PARM;
  COMPLETION (ASSIGNMENT) = '1'B;
  IF VARIABLE = 1 THEN DO;
     .
     .
     .
```

The WAIT statement ensures that the program must wait for the task to assign a value to a variable known only to A_TEST, and then set the event 'ASSIGNMENT' complete. If the WAIT statement was not used then TASK(ONE) PARM would be overwritten by TASK(TWO) PARM, then A may possibly have the value 2 before the assignment to VARIABLE, in which case the statements following THEN DO would never be executed.

ON-Units: If there is an ERROR ON-unit in a task, and the condition to which the ON-unit applies is raised in a subtask of that task, a GOTO out of the ON-unit will also raise the ERROR condition, causing the ERROR ON-unit to be reentered repeatedly in a loop, because the GOTO label will not be known to the attached subtask. This problem is avoided by using a separate ERROR ON-unit in the subtask with a GOTO label that is known to the subtask. For example:

```
MAJOR: PROC OPTIONS (MAIN, TASK);
       ON ERROR GO TO FINISH;
       CALL A TASK;
          .
          .
          .
       A: PROC;
       ON ERROR GO TO FINISH;
          SIGNAL ERROR;
          .
          .
          .
          END A;
        .
        .
        .
FINISH: END MAJOR;
```

CHECK Condition: If, in a multitasking program, the use of the CHECK condition is not not carefully synchronized, the results obtained may be unpredictable. A program executed with the CHECK condition may produce different results to the same program not using CHECK, since the presence of CHECK in a program may cause the task in which it occurs to wait for I/O and hence to lose control. When this task eventually regains control, some of the variables may have new values.

SNAP option in ON Unit: If an entry point has been called with the TASK option specifying a task name, then the entry point in the SNAP print-out is followed by the task name in brackets.

User Requested Dump: A useful debugging feature is the ability to obtain a storage dump at any point in the program.

A dump is obtained by the statement:

```
CALL dump identifier [(argument)]
```

The dump identifier is one of the following

IHEDUMP - dump all core associated with active tasks, and terminate all tasks

IHEDUMJ - dump all core associated with active tasks, and continue processing

IHEDUMT - dump all core associated with current task only, and terminate this task (and its subtasks)

IHEDUMC - dump all core associated with current task only, and continue processing.

When IHEDUMJ or IHEDUMP is specified, the contents of the dump may not necessarily be the contents of main storage at the time when the CALL statement was executed. This is because while the dump for one of the tasks is being written on an output device, the other tasks are still being processed. Contempory dumps can be obtained by use of the WAIT statement.

To ensure that a complete dump is provided, a WAIT should be given to prevent the task being terminated before the dump is complete.

If a PI1DUMP DD card has been supplied a standard OS/360 SNAP dump will be printed; if the identifier is IHEDUMP or IHEDUMT there will be one dump for each active task. The dump will include information such as register values, load list, contents of PIE, and storage dump. To help the user interpret the dumps, a directory is printed out at the start of each dump. This directory includes:

1. The task name and priority, and the attaching statement, if available.
2. The contents of the SYSPRINT file buffers, if the file is open.
3. The name of the files currently open, with the addresses of the relevant control blocks.

4. The name of the current file.

5. The addresses of the save areas and of other areas of special interest.

The argument in the CALL statement is optional; if it is used the dump identifier must be declared as ENTRY (FIXED BINARY). The argument is an expression that is evaluated at execution-time; the result is a fixed binary integer that appears in the heading of the dump. This integer must be in the range 0 to 127, a number outside this range is replaced by 127.

If no DD card is supplied, or if an unrecoverable error is deteted during the output, e.g, incorrect chaining of save areas, then a standard operating sytem ABEND dump will occur, and will terminate the job step.

For more information on storage dumps and the information they provide, see IBM System/360 Operating System: Messages, Completion Codes and Storage Dumps, Form C28-6631.

Return Codes: These can be set by use of the statement CALL IHETSAC. Return codes can be set in the major task.

Use of Priorities in PL/I

The priority associated with a particular job is supplied by the programmer, using the PRTY parameter in the JOB statement. The job priority can have any value from 0 to 14 inclusive; the higher the value the higher the priority.

This priority determines the initial priority of the major task in the PL/I program, using the formula:

p=(16*(job priority)) + 10

The absolute range of values provided by this formula is 10 to 234 inclusive; the range for a particular program is from 10 to a value that depends on the job priority. This value is not only the initial priority of the major task, it is also the maximum priority that any other task in the program can have. If an attempt is made to create a task priority greater than the maximum priority, the task will be executed at the maximum priority.

The priority of any task -- major, current or subtask -- can be reduced to zero; an attempt to create a priority of less than zero will result in the task concerned being executed at zero priority. An attached task can have a priority greater or smaller than its attaching task or of the major task, provided that this priority is within the limits given above. A priority can be changed within a program but only when it is the priority of the current task or of an immediate subtask of the current task.

A priority can be assigned to a task variable before the variable is associated with an active task. If the task is attached without the PRIORITY option, the priority has the value that was assigned to the task variable.

These conventions must be interpreted carefully when the PRIORITY pseudovariable, function or option is being used to manipulate the priority of a task. In particular, the effect of the maximum priority in restricting the manipulation should be noted. In the example given below, it is assumed that none of the omitted statements can cause transfer of control or create a wait situation.

In the example, control passes through the program to statement 12. Here task T1 is attached with a priority of 45; this is greater than the maximum priority, so T1 is given the maximum priority, 42. As this is higher than that of T(Z), the major task, control at once passes to T1, that is, to statement 26.

At statement 31, task T2 is attached with a priority of 38. Control remains in T1; statement 32 is executed. The priority of T2 is now the highest priority (T(Z)=35, T1=32, T2=38), so control passes at once to statement 41. After the execution of statement 47, the tasks have the priorities: T(Z)=35, T1=32, T2=30. Control now returns to T(Z), at statement 13.

For example:

```
  //jobname JOB 123, JOHNSMITH, MSGLEVEL=1,
            PRTY=2
```

Hence maximum priority=42

Statement number		Priority or value
1	Z:PROC OPTIONS (MAIN,TASK);	T(Z)=42
	.	
	.	
	.	
	6PRIORITY=-7;	
	.	
	.	
	.	
12	CALL Y TASK(T1) PRIORITY(10);	T(Z)=35
13		T1=42
	.	
	.	
	.	

```
16 PRIORITY=2:                          T(Z)=37
17 A=PRIORITY (T1);                     A=-5
18 PRIORITY (T1)=-9;                    T1=28
     .
     .
     .
25 END Z;                               T(Z)=37

26 Y:PROC;                              T1=42
     .
     .
     .
31 CALL X TASK (T2) PRIORITY (-4)       T1=42,
                                        T2=38
32 PRIORITY=-10;                        T1=32
33 B=PRIORITY (T2);                     NE
     .                                  NE
     .                                  NE
     .                                  NE
40 END Y,                               NE

41 X: PROC;                             T2=38
     .
     .
     .
47 PRIORITY=-8;                         T2=30
     .                                  NE
     .                                  NE
     .                                  NE
55 END X;                               NE
```

Note: T(Z) is the priority of the major task
NE=Not executed

In this program, the main procedure Z terminates normally, while the procedure Y and X terminate abnormally. This is because:

1. The value of the PRIORITY pseudovariable specified in procedures Y and X is such that it causes control to be transferred out of these procedures, leaving several statements in the procedure unexecuted.

2. There is no statement in the program that causes either procedure to be re-entered before the main procedure is terminated.

Only the priorities of the current task and of an immediate subtask of the current task can be determined directly. The calculation of priorities in PL/I is relative, not absolute; absolute values, including that of the major task, are not easily available.

I/O Handling

EVENT Option: An I/O event must be waited for in the task that initiated it.

Use of the I/O EVENT option requires careful consideration. In multitasking, the waiting time in a given task is overlapped by processing in another task. Using the EVENT option requires extra CPU time, which is obtained at the expense of time that could be used for processing another task. However, EVENT and WAIT can be used with advantage provided that a processing overlap can be guaranteed, that is, while one task is waiting for completion of an event, another task is being processed.

Another instance of the difficulties met when using an I/O event occurs in the use of WRITE statements in REGIONAL(3) files. If a REGIONAL(3) file with V- or U-format records is opened for DIRECT UPDATE or OUTPUT, and records are being added to the file, then under certain conditions, the program may not execute properly. These conditions are:

> If two or more tasks are simultaneously attempting to add records to the same track of the data set, and at least one of the WRITE statements concerned has the EVENT option, then if the WRITE statements are not correctly synchronized, the results are unpredictable.

This difficulty can be avoided in several ways:

1. The EVENT option should not be used on such WRITE statements

2. Such WRITE statements should be confined to one task

3. Use of non-I/O EVENT variables and WAIT statements to synchronize the WRITE statements in the various tasks

SEQUENTIAL Files: Use of a SEQUENTIAL file other than SYSPRINT in more than one task can present difficulties. For example, a task may be interrupted in the middle of a PUT statement. If, in the task that gets control, another PUT statement is executed on the same file, the result, at best, will be garbled fields in the output buffer and, at worst, will be changes to the internal control blocks so that the original task may not continue to execute properly and will probably terminate abnormally. Again, in the updating of a SEQUENTIAL RECORD file, a REWRITE will replace the last record read (or waited for) irrespective of which task read it. Suitable use of event variables and WAIT statements will avoid these situations, but the best solution is to keep all references to a SEQUENTIAL file to within a single task.

Only one file should sequentially create a data set on a direct-access device or a

tape volume. If two or more files are writing on the same sequential data set on a direct-access device, each file will write on the data set independently of the others, and records written by one file will be overwritten by records from another file. Similarly, if two or more files attempt to write on a sequential data set on the same tape volume, the program will terminate abnormally. To avoid this, a file should be opened in a common ancestor task of those using the file name.

In sequential-access operations a particular data set should be referred to by only one file at a time in a program. If in PCP or MFT two or more files opened for sequential output refer to two data sets having the same SYSOUT class then the records written by one file will be overwritten by the records from the other file, and the job may be abnormally terminated. If the MVT system is being used, the records will be written separately, and will not be mixed or overwritten.

If two or more files opened for sequential access refer to two data sets related by, for example, having the same magnetic tape device or the same data set name, then the records will be overwritten whatever the system used.

Since error messages are written on the SYSPRINT file, it should be opened in the major task. Otherwise SYSPRINT will be opened in each task, and the error messages may be overwritten or lost.

The use of SYSPRINT for large STREAM files in multitasking is not recommended. The implementation uses system facilities to synchronize operations (PUT statements) and error messages) on the file; the effect of this is to make PUT statements on the SYSPRINT file longer to execute than PUT statements on other PRINT files.

EXCLUSIVE Files: If a record of a REGIONAL (2) or (3) EXCLUSIVE file is referred to:

1. More than once in a task, or
2. By more than one task

then the same region number must be specified in each KEY option.

Synchronization: I/O synchronization means avoiding operations on a given data set by two or more tasks at the same time. If user I/O synchronization is inadequate, then either a system completion code of 001 is given or the results are unpredictable.

If a task terminates abnormally while I/O operation on a file is in progress in an attached task, and later the same file is accessed in another task, the results may be unpredictable.

System completion codes 301 and 001 are returned under the following conditions:

301 is returned under one of two circumstances:

1. Synchronization error in PL/I program
2. An attaching task terminates abnormally while an attached task is still active

The first situation occurs if an attaching task attempts an I/O operation on a file on which an attached task is already performing an I/O operation. Use of EVENT and WAIT can eliminate this problem.

The second situation occurs when an attached task is performing an I/O operation by means of QSAM and the attaching task terminates abnormally (due to, for example, a source program error). The attaching task issues a WAIT to the ECB (event control block) associated with the file, in order to allow the I/O operation to complete before the file is closed. But the attached task has already issued a WAIT for the same ECB; as only one WAIT can be issued for an ECB, the operating system terminates the program abnormally and the completion code is returned.

001 Irrecoverable I/O error caused by unsynchronized access to data set from more than one task, or from more than one file.

Strings

Unpredictable results may occur when two tasks are performing simultaneous operations on the same bit string or character string. Unpredictable results may also occur when an operation involving an unaligned bit string is taking place in one task at the same time as an operation involving data, which is not necessarily bit data, whose storage is contiguous with that of the bit string, is taking place in another task. The occurrence of this problem is likely to be extremely infrequent. However, if it does occur, the WAIT statement and COMPLETION pseudo-variable should be used when multiprocessing to avoid such results. The following examples indicate cases which may produce unpredictable results because the attaching task and subtask are executing simultaneously.

Example 1:

```
MAIN: PROC OPTIONS(MAIN,TASK);
DCL C CHAR(3) VARYING INIT('1');
CALL A EVENT(E);
C=C||'2';
WAIT(E);
     A: PROC;
     C=C||'3';
     END;
PUT DATA(C);   /*THE VALUE OF C MAY BE
                 '12' OR '13' OR '123'
                 OR '132'*/
END MAIN;
```

Example 2:

```
MAIN: PROC OPTIONS(MAIN,TASK);
DCL 1 A, 2 B CHAR(1) INITIAL('X'),
        2 C BIT(3) UNALIGNED
                   INITIAL('111'),
        2 D DEC FIXED(5,0) INITIAL(0);
CALL SUB EVENT(E);
C='101'B;
WAIT(E);
     SUB: PROC;
     B='Y';
     D=6;
     END;
PUT DATA(A);   /*THE VALUES OF B, C,
                 AND D MAY BE
                 INCORRECT*/
END MAIN;
```

Task Termination

Normal Termination: If a task terminates normally with active subtasks, then:

1. An indefinite wait situation might be created. For example, a task K (with subtasks I and M) might itself be a subtask of a task A. If I and M contain events that are waited for in A then, if K terminates normally while I and M are still active, the result is an indefinite wait in A.

2. A warning message (IHE577I) will be put out on SYSPRINT for each immediate active subtask of the task which is terminating normally.

The user must be aware that this situation -- normal task termination with active subtasks -- can happen unexpectedly. For instance, a task that was not expected to lose control may do by some implicit I/O, as in the following example:

```
A:PROC;
    .
    .
    .
  Y = 0:
    .
    .
    .
  CALL B PRIORITY (10);
    .
    .
    .
  B:PROC;
    .
    .
    .
  X = 5/Y; /*ZERODIVIDE*/
    .
    .
    .
  END B;
    .
    .
    .
END A;
```

When ZERODIVIDE occurs in B, a considerable amount of I/O may be necessary to load dynamically the error handling module to deal with the interrupt. This can allow A to gain control of the CPU; if A terminates normally before B can regain control, B is terminated abnormally.

It is advisable to wait for subtasks to terminate before terminating any block in the attaching task. Terminating such a block, however, can be useful as it provides a means of terminating a subtask that is being waited for.

If a subtask is terminated when it has active subtasks, the completion value of these subtasks will be unchanged i.e., 0, and their status values will be set abnormal.

Abnormal Termination: In addition to the information on abnormal termination provided in "Testing Programs", the following is relevent to multitasking:

1. If a STOP statement is executed in a subtask, the FINISH condition is raised in the subtask, not in the major task.

2. If the EXIT statement is executed in a subtask, the subtask and all subtasks attached by it are terminated, and the status of the event variable in the subtask is set abnormal (if it is not already so).

3. If the ERROR condition is raised in a major task, and there is no ERROR ON unit or normal return from an ERROR ON unit, then the FINISH condition is raised and the program is terminated.

4. If the ERROR condition is raised in a subtask, and there is no ERROR ON unit or normal return from an ERROR ON

unit, an error message is printed, but the FINISH condition is not raised. The subtask (and any subtasks attached to it) is terminated and the status of its event variable is set to abnormal.

If the operating system detects an error, such as 'No core available', in a subtask, then the subtask is terminated, the status of its event variable is set abnormal, and a message describing the error is printed on SYSPRINT (if it is OPEN, otherwise on the console). If the subtask has subtasks of its own, the status of these subtasks will be unchanged. The ERROR condition is not raised (because the task is already terminated before the PL/I library receives control), but, if a SYSABEND or SYSUDUMP card exists, a dump will be produced.

MULTIPROCESSING

Multiprocessing permits two or more tasks to be executed simultaneously.

On a machine with a single CPU, the highest priority task which is not in the wait state or not performing any I/O operations is executed by the CPU. No other tasks can be executed at the same time. On a multiprocessing machine (i.e., a machine with more than one CPU), the CPUs can execute tasks simultaneously. To obtain the best performance from a multiprocessing machine, the tasks should be kept as independent of each other as possible so that they can execute simultaneously without having to wait for operations to be completed in another task.

The following example is assumed to be executed on a two CPU machine with no other jobs executing at the same time so that both CPUs are available for execution. In the example, the major task attaches task T1, which has a higher priority than the major task, at the first CALL statement. After the CALL statement has been executed, the major task executes simultaneously with T1 even though one major task has a lower priority. At the second CALL statement, task T2 is attached, again with a higher priority than the major task. In this case, after the CALL statement has been executed, T1 and T2, being the two highest priority tasks, execute simultaneously and the major task does not proceed unless T1 or T2 performs some I/O operations or goes into the wait state. Assuming this does not happen, when T1 completes, a CPU is available for continuing the execution of the major task. The major task then executes simultaneously with T2 until the WAIT statement when (assuming that T2 has not completed by this time) it waits for the completion of T2 and terminates itself. If the major task did not wait for T2 to complete, then T2 would be terminated abnormally when the major task terminated.

Example:

```
Z: PROC OPTIONS(MAIN,TASK);
 .
 .
 .
PRIORITY = -5;
 .
 .
 .
CALL X TASK(T1) EVENT(E1) PRIORITY(2);
 .
 .
 .
CALL Y TASK(T2) EVENT(E2) PRIORITY(2);
 .
 .
 .
X: PROC;
 .
 .
 .
END X;
Y: PROC;
 .
 .
 .
END Y;
 .
 .
 .
WAIT(E2);
END Z;
```

Synchronization

Synchronization of I/O is most important since the likelihood of simultaneous operations on a given data set by two or more tasks is increased in multiprocessing. (See "I/O Handling" in this section.) Also, it is possible that an attaching task may complete execution before its subtask. Attaching a subtask with a higher priority than the attaching task does not ensure that the subtask completes first, since the attaching task may execute at the same time as the subtask although it has a lower priority. Hence, it is always advisable to wait for a subtask to terminate before terminating the attaching task.

Pseudo-Register Vector (PRV)

The PRV is a task-oriented communication area; one PRV is established for each task or subtask. For details of the PRV format and use, see 'Appendix' C: Object Program Organization and Conventions'.

PL/I SORT

This feature provides an interface between a PL/I program and the System/360 operating system Sort/Merge program. The user calls the PL/I library module IHESRT at the appropriate entry point; this in turn calls the Sort/Merge program by means of a LINK macro. Information defining the data to be sorted and the records in which this data exists is passed as arguments to the IHESRT module.

The entry point selected depends on the source of the records to be sorted and their disposition afterwards. Four circumstances are possible:

1. Records in a data set are retrieved and passed for sorting; the sorted records are placed in a data set.
2. Records constructed or updated in a PL/I program or procedure are passed for sorting; the sorted records are placed in a data set.
3. Records in a data set are retrieved and passed for sorting; the sorted records are passed to a PL/I program or procedure.
4. Records constructed or updated in a PL/I program or procedure are passed for sorting; the sorted records are passed to a PL/I program or procedure.

The retrieval of records from a data set, passing them to the sort program and placing the sorted records in a data set are all performed by the PL/I program. Records constructed or updated in a PL/I program or procedure can be passed, after sorting, to the same PL/I program or procedure or to a different one.

The records passed to the sort program are sorted a number of times until the required sequence is obtained. Each of these minor sorts is performed on the contents of selected fields within the record; up to twelve of these fields can be designated. This is adequate to ensure even long complicated records containing a small range of data types and values can be correctly and successfully sorted.

The sequence in which the records are placed is the System/360 collating sequence. This is described in IBM System/360 Operating System PL/I Reference Data, Keywords and Character Sets, Form X20-1744.

The user must be familiar with the usage of the Sort/Merge program; full details of this are given in IBM System/360 Operating System Sort/Merge, Form C48-6543. Brief information on some aspects, for example, record format, storage requirements, data set description, is provided here as a guide to the environment required, but this is not intended to supplant use of the Sort/Merge manual.

PL/I Sort Environment

Record format: Blocked and unblocked fixed- and variable-length records can be passed to the sort program. Record size can vary over a wide range:

Minimum: 18 bytes

Maximum: about 32,000 bytes. The size for a particular application depends on the size of the main storage and on the type of intermediate storage used.

Sort performance is improved if the input records are blocked. There are no restrictions on block size; however, where possible, large records should have small blocking factors.

Storage requirements: At least 15500 bytes of main storage are required .The amount of intermediate storage required (used for workspace and temporary storing of partially sorted records) depends on the size of the input data set. The amount required for a particular application can be calculated from formulas provided in the Sort/Merge manual. The number of devices required is:

Minimum: Three tape units or one direct-access device

Maximum: Thirty-two tape units or six direct-access devices.

The devices used must not be mixed; either all tape units or all direct-access devices of the same device-type must be used. Tape units can be 7- or 9-track, or a mixture of both.

If a 2314 disk storage device is used, at least six work files must be made available for the sort.

If direct-access devices are used, then sort performance is improved if:

1. Each data set is kept on a seperate device

2. The number of data sets used is a minimum

3. All data sets are the same length.

Data Sets: Input and output data sets must be accessed with the queued sequential access method (QSAM).

The load-module execution step requires some or all of the following DD statements in addition to the DD statements for the PL/I program.

```
//SORTIN    DD Input data set

//SORTOUT   DD Output data set

//SORTWK01 DD Data sets for work areas;
//SORTWK02 DD from 3 up to 32 of these
//SORTWK03 DD data sets can be used
//SORTWKnn DD

//SORTLIB   DD Sort program data set

//SORTCKPT DD Sort checkpoint data set

//SYSOUT    DD System output data set

//SYSLMOD   DD These are required because
//SYSLIN    DD the sort program calls the
//SYSUT1    DD linkage editor to select
                the appropriate routines
                for SORTLIB, to generate
                the sort program required
                by the PL/I program

//SYSUDUMP DD Storage dump data sets,
//PL1DUMP  DD useful while the PL/I
                program is being debugged
```

The parameters for all except two of these data sets depend on the particular application requiring the sort. The two exceptions are:

```
//SORTLIB DD DSNAME=SYS1.SORTLIB,DISP=OLD

//SYSOUT  DD SYSOUT=A
```

DISP=SHR should be specified for the SORTLIB data if SORTLIB is to be used concurrently by more than one job.

PL/I Sort checkpoints are written on a data set identified by the DD statement //SORTCKPT DD ... etc. A programmer-deferred checkpoint restart of a PL/I Sort should use the DD statement //SYSCHK DD ... etc., to identify the checkpoint data set to the restart program. Further information on the use of checkpoint restart is given in the section "Checkpoint/Restart," in this publication.

User parameters specified for the SYSUT1 DD statement should include:

```
//SYSUT1    DD UNIT=(SYSDA,SEP=(SORTLIB,
                SYSLMOD,SYSLIN))
```

If both the following occur:

```
//SYSOUT    DD SYSOUT=A

//SYSPRINT DD SYSOUT=A
```

then, if the SYSOUT device is a tape storage unit, the user must take care that the two data sets do not use the same output device. Under MVT it can be avoided by specifying a different device class for each data set.

SORTIN and SORTOUT are not always required; details are given in the description of the various entry points to IHESRT. SORTCKPT, SYSUDUMP and PL1DUMP are' optional, as is the number of SORTWKxx statements.

If the PL/I program uses an IBM cataloged procedure for its job control language, then SORTIN, SORTOUT and the SORTWKxx statements must be specified as GO.SORTIN, etc., because they are associated with the PL/I program, not the sort program. On the other hand, SYSLMOD, SYSLIN and SYSUT1 are associated with the sort program and therefore should be specified as given above and not as part of the cataloged procedure.

The user requires either SYS1.LINKLIB (which contains the sort/merge program) or a private job library in which sort/merge has been placed. SYS1.LINKLIB is loaded automatically with IEMAA, but a //JOBLIB DD statement will be required for the job library.

User Control of SORT ddnames

For multiple invocations of SORT within a single job step, the standard ddnames of SORT (SORTIN, SORTOUT, SORTWK, SORTMODS, and SORTCKPT) can be changed by replacing the first four characters of the ddnames. This is achieved by adding an extra argument to the end of the argument list in the CALL statement to IHESRT. The argument is a character string, which can be any length, but only the first four characters

are used to replace the letters SORT in the standard ddnames.

If the string is null, the standard ddnames remain unchanged. If the string is more than four characters long, only the first four characters are used. If the string is from one to four characters long, then the first one to four characters in the standard ddnames are replaced.

The first character in the string must be alphabetic otherwise either an OPEN error may occur in SORT or an error will occur during scheduling of the job step if an illegal ddname (e.g., 2ORT) is found.

The PL/I (F) Compiler does not permit a variable number of arguments to the same entry name in separate CALL statements. Therefore a compilation which invokes multiple sorting operations must contain the CALL statements with the argument to specify the ddnames set to null ('') when using standard ddnames, and set to the required character string for the modified ddname.

Example:

```
TEST: PROCEDURE OPTIONS(MAIN);
      DECLARE IHESRTA ENTRY(CHAR, CHAR,
          FIXED BIN, FIXED BIN, CHAR);
      DECLARE STRING CHARACTER(2)
          INITIAL('PA');
      /*INVOKE SORT USING STANDARD
        DDNAMES*/
      CALL IHESRTA(ARG1, ARG2, ARG3,
          ARG4, '');
       .
       .
       .
      /*INVOKE SORT USING MODIFIED
        DDNAMES*/
      CALL IHESRTA(ARG1, ARG2, ARG3,
          ARG4, STRING);
       .
       .
       .
      END TEST;
```

In the first invocation of IHESRTA the following DD statements are required:

```
//SORTIN DD ...
//SORTOUT DD ...
//SORTWK1 DD ...
//SORTWK2 DD ...
```

In the second invocation of IHESTRTA the following DD statements are required:-

```
//PARTIN DD ...
//PARTOUT DD ...
//PARTWK1 DD ...
//PARTWK2 DD ...
```

Sorting Records from One Data Set to Another

Suppose that a data set containing 4000 logical records of the type shown in Figure 24.1 is to be sorted according to the values in four fields in each record:

1. PKDEC, a fixed-point decimal field; the contents are to be arranged in ascending order
2. CHTER, a character field, in ascending order
3. FLTNUM, a floating-point decimal field, in descending order
4. BINNUM, a fixed-point binary field, in ascending order

When the records are finally sorted, the contents of the field described above could be printed as:

PKDEC	CHTER	FLTNUM	BINNUM
01	BLACK	1.034	010B
01	BLACK	1.034	011B
01	BLACK	1.033	000B
01	BLACK	1.033	001B
01	BLACK	1.033	101B
01	ORANGE	2.087	010B
01	ORANGE	2.087	011B
01	ORANGE	0.072	101B
01	ORANGE	0.072	110B
02	BLACK	1.029	001B
		etc.	

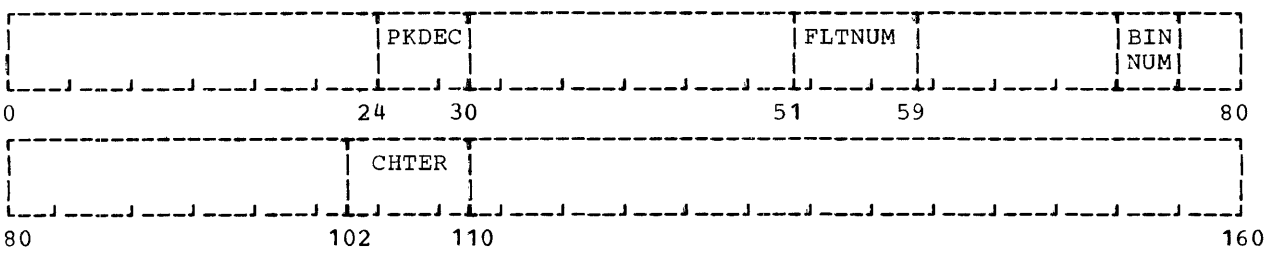


Figure 24.1. Figure-Length 160-byte Records, showing Fields on which Sort is to be Made

The information defining the data to be sorted and the records in which it exists is described in two sort program statements -- the SORT statement and the RECORD statement.

The SORT statement describes a control word that defines the fields on which the sort is to be made. The control word consists of up to twelve control fields; each control field describes a field in the record to be sorted. The statement format is:

SORT FIELDS=(b_1,l_1,f_1,s_1,...b_n,l_n,f_n,s_n),
SIZE=m, SKIPREC=z, CKPT

where

b = first byte of field to be sorted.

Binary data can start on any bit within a byte and is specified:

7.2 Bit 2 in byte 7

10.3 Bit 3 in byte 10

All other data starts on a byte boundary, which is specified as an integer.

l = length (in bytes) of the control field. Since binary data can start and end on any bit, its length is specified in the byte bit notation given above:

2.1 the length of the binary data field is 17 bits

All other lengths are specified as integers.

f = data type. The code for the various data types is:

BI	Binary
CH	Character
FI	Fixed-point
FL	Floating-point
PD	Packed decimal
ZD	Zoned decimal

If all the control fields have the same data type, then f_1,...f_n can be omitted from the control fields and FORMAT=x(where x is the data type) inserted after the right parenthesis and before the SIZE parameter.

These sort data types correspond to the following PL/I data types:

Sort	PL/I
BI	BINARY
	BIT
CH	CHARACTER
FI	FIXED BINARY
	FIXED DECIMAL
FL	FLOAT BINARY
	FLOAT DECIMAL
PD	FIXED DECIMAL
ZD	PICTURE

s = the order in which the contents of the field are to be sorted. The codes are:

A ascending order
D descending order

m = the number of records in the input data set. If this number is not known precisely, an estimated total can be specified thus:

SIZE=Em

z = the number of records to be skipped before the sort begins. This allows a sort to begin at any point in a data set, omitting any records for which a sort is not required.

CKPT A checkpoint is taken at several points in the sort program.

The specification of these SORT fields is subject to the following restrictions and conventions:

1. The total lengths specified in all the fields in one SORT statement must not be greater than 256 bytes. If binary data specifies part of a byte, the whole of that byte must be included in the length count. For example, a binary field starting at 11.3 and ending at 27.2 is 17 bytes long.

2. All the fields specified by the control fields must be in the first 4092 bytes of the record.

3. The maximum length of a decimal field is 16 bytes; all other fields can be up to 256 bytes long.

The SORT statement for the records in the example is:

```
SORT FIELDS=(24,6,PD,A,102,8,CH,A,51,8,FL,
          D,74.7,0.3,BI,A),SIZE=4000
```

The RECORD statement describes the format and length of the records to be sorted. The format is:

RECORD TYPE=r, LENGTH=(l_1,l_2,l_3,l_4,l_5,)

where

r = record format. The code is:

F fixed-length
V variable-length

l_1= length of each record in the input data set, as follows:

F records record length
V records maximum length

The length must be the same as the LRECL value in the DCB parameter for the SORTIN data set; if it is not, then the LRECL value is taken.

l_2= length of each record to be handled by the sort program, as follows:

F records record length
V records maximum length

If this value is not given in the RECORD statement, then it is assumed to be the same as l_1.

l_3= length of each record in the output data set, as follows:

F records record length
V records maximum length

If this value is not given in the RECORD statement, it is assumed to be l_2. The value must be the same as LRECL value in the DCB parameter in the SORTOUT data set: if it is not, then the LRECL value is used.

l_4= minimum length of variable-length records in the input data set.

If this value is not given, then it is assumed to be the greater of:

a. The minimum necessary to contain the control fields specified in the SORT statement, or

b. the minimum physical-record length required by the operating system.

l_5= the most frequent record length occuring in a data set containing variable-length records. It is called the modal length. If this value is not given, then it is assumed to be the average of the minimum and maximum lengths of the records in the input data set.

The specifications of these RECORD fields is subject to the following restrictions and conventions:

1. The lengths specified for variable-length records must include the four byte count field at the beginning of each record.

2. When a direct-access device is used for intermediate storage, the record length must not exceed one track.

3. The record format must be the same as that specified in the RECFM subparameter in the DCB parameter for the SORTIN and SORTOUT data sets. If it is not, then the SORTIN RECFM specification is used.

4. Values in the LENGTH parameter that are equal to those assumed by the program can be dropped from the operand. Values dropped from the right-hand end of the operands are simply omitted; values dropped at the beginning or middle of the operand must be indicated by commas:

LENGTH=(l_1,l_2)
LENGTH=(l_1,,,l_4)

The RECORD statement for the records in the example is:

```
RECORD TYPE=F, LENGTH=(160)
```

For this application (sorting records from one data set to another) the PL/I library module IHESRT is entered at the entry point IHESRTA as follows:

```
CALL IHESRTA(argument1,argument2,argument3,
             argument4):
```

where

$argument_1$ = a character-string expression representing the SORT statement.

$argument_2$ = a character-string expression representing the RECORD statement.

$argument_3$ = an arithmetic expression that on evaluation gives a fixed binary integer. This integer gives the amount of main storage available to the sort program. If this value is less tham 12,000 bytes, then 12,000 is assumed.

$argument_4$ = a fixed-binary variable of precision (15,0) that contains the value of the return code returned by the sort program:

0 Sort successful
16 Sort unsuccessful

If the sort is unsuccessful, diagnostic messages are printed on the console listing or the listing of the system output file (SYSOUT or SYSPRINT).

To use this module in a PL/I program, IHESRTA is declared as an entry name with the appropriate parameters. IHESRTA is then called with arguments described above. Once called, it continues to pass records for sorting and placing the sorted records in a data set until either the specified number of records has been sorted or the sort is unsuccessful. In the latter case, the sort stops at this point.

The value of the character-string expression for the SORT and RECORD statements has the form:

'bstatementb'

The blanks at the beginning and end of the expression are always required. An embedded blank must occur between SORT and FIELDS, and between RECORD and TYPE; no other embedded blanks are permitted.

When character-string constants are used that are too long for one record of the PL/I source program, they are continued in the following record. The user must take care that embedded blanks are not inadvertantly inserted at the beginning and end of such records; the value of the SORMGIN parameter must be taken into consideration.

A PL/I program to sort the records described above would require both the SORTIN and SORTOUT DD statements, as well as any others that are necessary. The source code could be:

```
ALPHA: PROC OPTIONS (MAIN);

   DCL IHESRTA ENTRY(CHAR(70),CHAR(28),
         FIXED BINARY(31,0), FIXED
         BINARY(15,0)),
      SFIELD CHAR(70),
      RFIELD CHAR(28),
      RETCDE FIXED BINARY(15,0);

   SFIELD=' SORT FIELDS=(24,6,PD,A,102,8,
            CH,A,51,8,FL,D,74.7,0.3,BI,A),
            SIZE=4000 ';

   RFIELD=' RECORD TYPE=F,LENGTH=(160) ';

   CALL IHESRTA(SFIELD,RFIELD,45056,
      RETCDE);

END ALPHA;
```

Sorting Records from a PL/I Program or Procedure onto a Data Set

Records constructed or updated in a PL/I program or procedure can be passed to the sort program by means of a RETURN (expression) statement, and (when sorted) placed in a data set, by using the entry point IHESRTB. This is invoked as follows:

CALL IHESRTB (argument$_1$,argument$_2$, argument$_3$,argument$_4$,argument$_5$)

where

arguments$_1$ - $_4$ = as for IHESRTA

argument$_5$= the entry name of the PL/I procedure supplying the records to the sort program.

The PL/I records are passed to an entry point in the sort program called a user exit. A user exit is a point in the executable code of the sort program at which control can be received from or passed to a user program that modifies the sort program or uses only part of it. The user exit for passing records from a user program to the sort program is E15.

The procedure invoked by argument passes to the sort program a character-string representation of the record to be sorted. If the record is not in character-string form, it must be defined on a character string and then passed. This may lead to difficulties in the PL/I program. The language rules specify that the attributes of the defined and the base items must match exactly except for their lengths and that string overlay defining on an aggregate parameter is not permitted. The (F) implementation permits the use of differing attributes and of this type of overlay defining, and produces error-level diagnostics when these situations occur. Successful link-editing and execution are possible provided the condition codes on the appropriate EXEC statements are adjusted to allow the step concerned to be executed.

In addition to the return code supplied by the sort program to the PL/I program, the PL/I program supplies a return code to the sort program that indicates whether there are any more records to be passed for sorting. This return code is set by one of the following statements:

```
CALL IHESARC(n);  (single-task programs)
CALL IHETSAC(n);  (multitasking programs)
```

where n has the values:

8 No more records will be passed

12 Insert the record passed into the file to be sorted

If the CALL IHETSAC(n) statement is to be used, the *TASK option must be specified* in the MAIN procedure statement.

The PL/I program supplying the records requires all the DD statements given above except the SORTIN DD statement. As there is no input data set, the SIZE parameter is not required. The code for a program supplying the records described above could be:

```
 BETA:  PROC OPTIONS (MAIN);
   DCL IHESRTB ENTRY (CHAR(60),CHAR(28),
         FIXED BINARY(31,0),
         FIXED BINARY(15,0), ENTRY),
     IHESARC ENTRY (FIXED BINARY),
     ENT15 RETURNS(CHAR(160)),
     SFIELD CHAR(60),
     RFIELD CHAR(28),
     RETCDE FIXED BINARY (15,0);
SFIELD=' SORT FIELDS=(24,6,PD,A,102,8,CH,A,
               51,8,FL,D,74.7,0.3,BI,A) ';
RFIELD=' RECORD TYPE=F,LENGTH=(160) ';
CALL  IHESRTB (SFIELD,RFIELD,45056,RETCDE,
               ENT15);
ENT15: PROC CHAR(160);
   DCL INPUT FILE RECORD DIRECT ENV
       (REGIONAL(3))KEYED,
     INREC CHAR(160) DEF INRECX,
     1 INRECX UNALIGNED,
        2 PADA CHAR(23),
        2 PKDEC FIXED DECIMAL (11),
        2 PADB CHAR(21),
        2 FLTNUM FLOAT(16),
        2 PADC CHAR(13),
        2 BINNUM FIXED BINARY(16),
        2 PADD CHAR(26),
        2 CHTER CHAR(8),
        2 PADE CHAR(51),
     N STATIC FIXED BINARY(15,0)
        INIT(4000);
IF N=0 THEN DO;
   READ FILE(INPUT) INTO(INRECX)
        KEY(N);
   CALL IHESARC(12);
   N=N-1;
   RETURN(INREC);
   END;
   ELSE CALL IHESARC(8);
   RETURN;
END ENT15;
END BETA;
```

Sorting Records from a Data Set into a PL/I Program or Procedure

Records from a data set can be sorted and than passed as arguments to a PL/I program or procedure, by using the entry point IHESRTC. This is invoked by:

```
CALL IHESRTC(argument1,argument2,argument3,
     argument4,argument6)
```

where

argument$_{1-4}$ = as for IHESRTA

argument$_6$ = the entry name of the PL/I procedure to which the sorted records are to be passed. The user exit for this is E35.

The records passed by the sort program must be in a character-string form. If this form is not the one required by the PL/I program, then the PL/I record must be defined on a character string. The same difficulties can be expected here as for IHESRTB.

The return code passed by the PL/I program to the sort program, using the CALL IHESARC(n) or CALL IHETSAC(n) statement has the values:

4 the record passed has been accepted, pass the next record

8 stop passing records, even if there are still more to come.

If no return code is passed to the sort program, then this program continues to pass records until all have been passed.

All the DD statements described earlier may be used except the SORTOUT DD statement.

If the records described above were retrieved from a data set, sorted and the passed to a PL/I program or procedure for printing on SYSPRINT, the code for the source program could be:

```
GAMMA: PROC OPTIONS(MAIN);

    DCL IHESRTC ENTRY (CHAR(70),  CHAR(28),
          FIXED BINARY(31,0),
          FIXED BINARY(15,0),ENTRY),
       IHESARC ENTRY (FIXED BINARY),
       SFIELD CHAR(70),
       RFIELD CHAR(28),
       RETCDE FIXED BINARY(15,0);

    SFIELD=' SORT FIELDS=(24,6,PD,A,102,8,
          CH,A,51,8,FL,D,74.7,0.3,BI,A),
          SIZE=4000 ';
    RFIELD=' RECORD TYPE=F,LENGTH=(160) ';
    CALL  IHESRTC  (SFIELD,RFIELD,   45056,
               RETCDE,ENT35);
  ENT35: PROC (OUTREC);
    DCL OUTREC CHAR(160),
      1 OUTRECX DEF OUTREC,
        2 PADA CHAR(23),
        2 PKDEC FIXED DECIMAL(11),
        2 PADB CHAR(21),
        2 FLTNUM FLOAT(16),
        2 PADC CHAR(13),
        2 BINNUM,
          3 NOBITA BIT(23),
          3 BINNO BIT(3),
          3 NOBITB BIT(6),
        2 PADD CHAR(26),
        2 CHTER CHAR(8),
```

```
        2 PADE CHAR(51),
     N STATIC FIXED BINARY(15,0)
        INIT(4000);
   IF N1=0 THEN DO;
     CALL IHESARC(4);
     PUT FILE(SYSPRINT) SKIP EDIT(PKDEC,
         CHTER,FLTNUM,BINNO)(F(11),X(2),
         A(8),X(2),F(16,3),X(2),B(3));
     N=N-1;
     RETURN;
     END;
   ELSE CALL IHESARC(8);
   RETURN;
END ENT35;
END GAMMA;
```

Supplying, Sorting, and Passing Back Records to a PL/I Procedure

Records constructed or updated by a PL/I program or procedure can be passed to the sort program; the sorted records can then be passed back to the same PL/I program or procedure or to a different one.

This requires the entry point IHESRTD, as follows:

```
CALL IHESRTD (argument1,argument2,
     argument3,argument4,argument5,
     argument6)
```

where

arguments$_{1-4}$ = as for IHESRTA

argument$_5$ = as for IHESRTB

argument$_6$ = as for IHESRTC

Neither the SORTIN nor the SORTOUT DD statements are required for this entry point, nor is the SIZE parameter. A PL/I program to sort the records described earlier could be:

```
DELTA: PROC OPTIONS(MAIN);
      DCL IHESRTD ENTRY(CHAR(60),CHAR(28),
           FIXED BINARY(31,0),
           FIXED BINARY(15,0),ENTRY,ENTRY),
         IHESARC ENTRY (FIXED BINARY),
         ENT15 RETURNS(CHAR(160)),
         SFIELD CHAR(60),
         RFIELD CHAR(28),
         RETCDE FIXED BINARY (15,0);
      SFIELD=' SORT FIELDS=(24,6,PD,A,
              102,8,CH,A,51,8,FL,D,74.7,
              0.3,BI,A) ';
      RFIELD=' RECORD TYPE=F,LENGTH=
              (160) ';

      CALL IHESRTD(SFIELD,RFIELD,45056,
        RETCDE,ENT15,ENT35);
      IF RETCDE=16 THEN DO;
          PUT FILE(SYSPRINT) SKIP LIST
          ('SORT WAS UNSUCCESSFUL');
          GO TO ENDA;
          END;



ENT15: PROC CHAR(160);
      DCL INREC CHAR(160) DEF INRECX,
       1 INRECX UNALIGNED,
              2 PADA CHAR(23),
              2 PKDEX FIXED DECIMAL(11),
              2 PADB CHAR(21),
              2 FLTNUM FLOAT(16),
              2 PADC CHAR(13),
              2 BINNUM FIXED BINARY(16),
              2 PADD CHAR(26),
              2 CHTER CHAR(8),
              2 PADE CHAR(51);
                   .
                   .
                   .
       (Code for generating records)
                   .
                   .
                   .
  IF I<=100 THEN DO;
      CALL IHESARC(12);
      I=I+1;
      RETURN(INREC);
      END;
  ELSE CALL IHESARC(8);
  RETURN;
END ENT15;



ENT35: PROC(OUTREC);
      DCL OUTREC CHAR(160),
        1 OUTRECX UNALIGNED DEF OUTREC,
            2 PADA CHAR(23),
            2 PKDEC FIXED DECIMAL(11),
            2 PADB CHAR(21),
            2 FLTNUM FLOAT(16),
            2 PADC CHAR(13),
            2 BINNUM,
              3 NOBITA BIT(23),
              3 BINNO BIT(3),
              3 NOBITB BIT(6),
            2 PADD CHAR(26),
            2 CHTER CHAR(8),
            2 PADE CHAR(51),
      N STATIC FIXED BINARY(15,0) INIT(1);
  IF N<=100 THEN DO;
      CALL IHESARC(4);
      PUT FILE(SYSPRINT) SKIP EDIT(PKDEC,
         CHTER,FLTNUM,BINNO)(F(11),X(2),
         A(8),X(2),F(16,3),X(2),B(3));
      N=N+1;
      RETURN;
      END;
  ELSE CALL IHESARC(8);
  RETURN;
END ENT35;
ENDA: END DELTA;
```

Use of PL/I SORT in a Multitasking Environment

If the PL/I SORT is invoked from different tasks within a main PL/I procedure, i.e., two or more sorting operations are to be performed asynchronously by separate subtasks, the following restriction applies when messages output by the sort programs are to be listed on the line printer: the SYSOUT DD statements which specify the data set for the messages must not use SYSOUT= in the operand field; this operand field must specify an actual device, rather than the system output stream, for example:

```
//SYSOUT DD UNIT=1403
```

This restriction does not apply if the messages are to be printed on the console.

DATA INTERCHANGE

The facilities that allow non-PL/I data sets to be used with PL/I programs are now extended to include FORTRAN data. Previously, the COBOL attribute (see 'The ENVIRONMENT Attribute' in Appendix B) provided the only facility for data interchange with a high-level language. Now, the ALIGNED and UNALIGNED attributes provide a similar facility for FORTRAN, that is, they allow PL/I programs to use FORTRAN data sets.

A FORTRAN unformatted data set consists of records which are simply the concatenation of individual internal data items without regard to alignment stringency. In general, the data construction in these records is equivalent to that of a PL/I UNALIGNED structure containing base elements which are the same as the data items on the record. For example

```
INTEGER * 4 A
LOGICAL * 1 B
REAL * 8 C

WRITE (5) A,B,C
```

The record written has the same format as the data in a PL/I structure of the form:

```
DECLARE    1 R UNALIGNED,
             2 S FIXED BINARY,
             2 T BIT(8),
             2 U FLOAT(16);
```

There are, however, two exceptions:

1. FORTRAN records may contain halfword binary data, which is not supported by the PL/I (F) compiler. Such data can still be read, however, (providing it is unsigned) by reading it into a character string of length 2, and then using UNSPEC to assign it to a PL/I FIXED BINARY variable.

2. FORTRAN IOCS can split its logical records into several physical records, a technique known as spanning. The fact that a record is spanned is transparent to the FORTRAN programmer. Spanned records are not supported by the PL/I (F) compiler. Unspanned FORTRAN records can be read by PL/I using UNALIGNED structures.

For use of the ALIGNED/UNALIGNED attributes, see 'Use of Storage' in the chapter 'Programming Techniques.'

The information in this section is divided into two main categories: the first part provides a list of the errors and pitfalls most likely to be encountered by users when first programming in PL/I; the second part lists various methods of improving the performance of a program, together with some housekeeping points and additional hints.

1. COMMON ERRORS AND PITFALLS

This is a list of the errors and pitfalls most likely to be encountered when writing a PL/I source program. Some of the items concern misunderstood or overlooked language rules, while others result from failure to observe the implementation conventions and restrictions of the PL/I (F) compiler, and are indicated by (I) appearing after the item.

a. Operating System and Job Control

(1) The option list in the PARM parameter of an EXEC statement should be enclosed in quotation marks, and is limited to a maximum length of 40 characters (including commas but excluding quotation marks). (I)

(2) When the source deck is in the input stream to the sequential scheduler, check that the last card in the source deck is a delimiter (/* in columns 1 and 2). (I)

(3) A STATIC variable in an overlay segment could be overwritten during an overlay operation unless it has the EXTERNAL attribute. (I)

(4) Special care should be taken when using ENVIRONMENT (INDEXED). For example, no secondary quantity can be allocated by the SPACE parameter. (I)

b. Source Program and General Syntax

(1) Keypunch transcription errors may occur unless particular care is taken when writing the following characters:

1 (numeral), I (letter), | (or),
/ (slash), ' (quotation mark);

¬ (not), 7 (seven),
> (greater than);

L (letter), < (less than).

O (letter), 0 (zero);

S (letter), 5 (five);

Z (letter), 2 (two);

_ (break character),
- (minus sign);

(2) Ensure that the source program is completely contained within the margins specified by the SORMGIN option. (I)

(3) Inadvertent omission of certain symbols may give rise to errors that are difficult to trace. Common errors are: unbalanced quotation marks; unmatched parentheses; unmatched comment delimiters (e.g., /* punched instead of */ when closing a comment); and missing semicolons.

(4) Reserved keyword operators in the 48-character set (e.g., GT, CAT) must in all cases be preceded and followed by a blank or comment.

(5) Care should be taken to ensure that END statements correctly match the appropriate DO, BEGIN, and PROCEDURE statements.

(6) In some situations, parentheses are required when their necessity is not immediately obvious. In particular, the expression following WHILE and RETURN must be enclosed in parentheses.

c. Program Control

(1) The procedure to be given initial control at execution time must have the OPTIONS(MAIN) attribute. If more than one procedure has the MAIN option, the first one gets control. (I)

(2) When a procedure of a program is invoked while it is still active, it is said to be used recursively. Attempting the recursive use of a procedure that has not been given the RECURSIVE attribute may result in a program interrupt after exit from the procedure. This will occur if reference is made to AUTOMATIC data of an earlier invocation of the procedure.

d. Declarations and Attributes

(1) DECLARE statements for AUTOMATIC variables are in effect executed at entry to a block; sequences of the following type are therefore likely to lead to unpredictable storage requests:

```
A: PROC;
   N=4;
   DCL B(N) FIXED;
   .
   .
   .
   END;
```

(2) Missing commas in DECLARE statements are a common source of error. For example, a comma must follow the entry for each element in a structure declaration.

(3) External identifiers should neither contain more than seven characters, nor start with the letters IHE. (I)

(4) In a PICTURE declaration, the V character indicates the scale factor, but does not in itself produce a decimal point on output. The point picture character produces a point on output, but is purely an editing character and does not indicate the scale factor. In a decimal constant, however, the point does indicate the scale factor. For example:

```
DCL A PIC'99.9',
    B PIC'99V9',
    C PIC'99.V9';
A,B,C=45.6;
PUT LIST (A,B,C);
```

This will cause the following values to be put out for A, B, and C, respectively:

```
04.5    456     45.6
```

If these values were now read back into the variables by a GET LIST statement, A, B, and C would be set to the following respective values:

```
004     56.0    45.6
```

If the PUT statement were then repeated, the result would be:

```
00.4    560     45.6
```

(5) Separate external declarations of the same identifier must not specify conflicting attributes, either explicitly or by default. If this occurs the compiler will not be able to detect the conflict. PL/I also requires that if an INITIAL value is specified in one declaration of a STATIC EXTERNAL variable, the same INITIAL value should appear in every declaration of that variable.

(6) An identifier cannot be used for more than one purpose within its scope. Thus, the use of X in the following sequence of statements would be in error:

```
PUT FILE (X) LIST (A,B,C); X=Y+Z;
X: M=N;
```

(7) It is advisable to declare all entry points, associated parameter lists, and any return values, to avoid inadvertent clashes of attributes.

If the attributes of the data items in an argument list do not match those declared for the ENTRY, a dummy argument is created with the correct attributes, and the data item is converted into the dummy. For example:

```
DCL X ENTRY (FIXED, CHAR(4)),
    Y FIXED, Z FIXED(1,0);
Y=45;
Z=0;
CALL X(Y,Z);

X:PROC(A,B);
  DCL A FIXED,
      B CHAR(4);
  END;
```

In the above example, a dummy is created for the second argument, Z, and is passed to X as 'bbb0'.

If the attributes declared for X in the entry name declaration were incompatible with the attributes of the arguments in the CALL statement, the compiler would issue a diagnostic message, and at execution time no conversion would take place. However, if the attributes declared for X in the entry name declaration conflicted with the attributes of the corresponding parameters in the PROCEDURE statement, the compiler would not detect the disagreement, and at execution time

the consequences of such an error would, in general, be unpredictable. For example, if X were declared

```
DCL X ENTRY (FLOAT, CHAR(4));
```

then 45 would be passed as FLOAT, but would be interpreted by X as FIXED, possibly with disastrous results.

Similarly, attributes declared for RETURN values must agree in the invoking and invoked procedures; however, the actual expression returned may be of any data type and will be converted to that declared. For example:

```
    DCL X RETURNS (CHAR(4));
    DCL A CHAR(4);

X:  PROC CHAR(4);
    RETURN (I*J*K);
    END X;

    A=X;
```

The precision of decimal integer constants should be taken into account when such constants are passed. For example:

```
CALL ALPHA(6);

ALPHA: PROCEDURE(X);
       DCL X FIXED DECIMAL;
       END;
```

The above example is incorrect because X will be given a default precision, while the constant, 6, will be passed with precision (1,0).

(8) When a data item requires conversion to a dummy, and the called procedure alters the value of the parameter, note that the dummy is altered, not the original argument. For example:

```
    DCL X ENTRY (FIXED, FIXED),
        A FIXED,
        B FLOAT;
    CALL X(A,B);

X:PROC(Y,Z);
  DCL (Y,Z) FIXED;
  Y=Z**100; /*A IS ALTERED IN
              CALLING PROC*/
  Z=Y**3;   /*B IS UNALTERED IN
              CALLING PROC*/
  END X;
```

(9) When the attributes for a given identifier are incompletely declared, the rest of the required attributes are supplied by default. The following default assumptions should be carefully noted.

FLOAT DECIMAL REAL is assumed for implicitly declared arithmetic variables, unless the initial letter is in the range I through N, when FIXED BINARY REAL is assumed.

If a variable is explicitly declared and any of the base, scale, or mode attributes is specified, the others are assumed to be from the set FLOAT/DECIMAL/REAL. For example:

```
DCL I;          /*I IS FIXED BINARY
                  (15,0) REAL
                  AUTOMATIC*/

DCL J REAL;     /*J IS FLOAT DECIMAL
                  (6) REAL
                  AUTOMATIC*/

DCL K STATIC;   /*K IS FIXED BINARY
                  (15,0) REAL
                  STATIC*/

DCL L FIXED;    /*L IS FIXED DECIMAL
                  (5,0) REAL
                  AUTOMATIC*/
```

(10) The precision of complex expressions is not obvious. For example, the precision of 1 + 1I is (2,0), that is, the precision follows the rules for expression evaluation.

(11) When a procedure contains more than one entry point, with different parameter lists on each entry, make sure that no references are made to parameters other than those associated with the point at which control entered the procedure. For example:

```
A: PROCEDURE(P,Q);
   P=Q+8; RETURN;
B: ENTRY(R,S);
   R=P+S; /*THE REFERENCE TO P
            IS AN ERROR*/
   END;
```

(12) BASED storage is allocated in terms of doublewords; therefore, even for the smallest item, at least eight bytes are required. (I)

(13) The variable used in the REFER option must be referred to unambiguously. For example:

```
DCL 1 A,
      2 Y FIXED BIN,
      2 Z FLOAT,
    1 B,
      2 Y FIXED BIN
      2 T(1:N REFER(B.Y));
```

In any references to this declaration, Y must be fully qualified to prevent a possible ambiguity.

(14) A pointer qualifier (explicit or implicit) may not be based or subscripted. (I)

(15) Conflicting contextual declarations must be avoided. P is often used as the name of a pointer; it must not, therefore, assume by default the characteristics of another data type. For example:

```
B BASED (P),
.
.
.
P AUTO,
.
.
.;
```

P is first contextually declared to be a pointer and then, by default, to be FLOAT DECIMAL.

(16) BASED variables cannot be used as arguments or parameters. (I)

(17) Offsets must be declared with a level 1 unsubscripted BASED area.

e. Assignments and Initialization

(1) When a variable is accessed, it is assumed to have a value which has been previously assigned to it and which is consistent with the attributes of the variable. If this assumption is incorrect, either the program will proceed with incorrect data or a program interrupt will occur. Such a situation can result from failure to initialize the variable, or it can occur as a result of the variable having been set in one of the following ways:

(a) by the use of the UNSPEC pseudo-variable

(b) by RECORD-oriented input

(c) by overlay defining a picture on a character string, with subsequent assignment to the character string and then access to the picture

(d) by passing as an argument a variable assigned in a different procedure, without matching the attributes of the parameter.

Failure to initialize a variable will result in the variable having an unpredictable value at execution time. Do not assume this value to be zero.

Failure to initialize a subscript can be detected by checking for subscripts out of range, when debugging the program.

(2) Any attempt to write out a variable or array that has not been initialized may well cause a data interrupt to occur. For example:

```
DCL A(10) FIXED;
A(1)=10;
PUT LIST (A);
```

To avoid the data interrupt, the array should be initialized before the assignment statement, thus:

```
A=0;
```

Note that this problem can also occur as a result of CHECK system action for an uninitialized array. If the CHECK condition were enabled for the array in the above example, and system action were taken, the results, and the way in which the program terminates, would be unpredictable. The same problem arises when PUT DATA is used.

(3) Note the distinction between = (assignment) and = (comparison). The statement

```
A=B=C;
```

means "compare B with C and assign the result (either '1'B or '0'B) to A, performing type conversion if necessary."

(4) Assignments that involve conversion should be avoided if possible (see section 1.f.2., below).

(5) In the case of initialization of or assignment to a fixed length string: if the assigned value is shorter than the string, it is extended on the right with blanks (for a character string) or zeros (for bit strings). For example:

```
DCL A CHAR(6),
    B CHAR(3) INIT('CR');
    A=B;
```

After the execution of the above statements, B would contain CRb, and A would contain CRbbbb.

(6) It is not possible to assign a cross section of an array of structures in a single statement; the whole of an array of structures, or a single element may be referenced, but not a cross section. (I)

(7) When SIZE is disabled, the result of an assignment which would have raised SIZE is unpredictable:

FIXED BINARY: The result of an assignment here -- which includes, for instance, source language assignments and the conversions implied by parameter matching -- may be to raise FIXEDOVERFLOW.

FIXED DECIMAL: Truncation to the nearest byte may occur, without raising an interrupt. If the target precision is even, an extra digit may be inserted in the high-order byte.

f. Arithmetic and Logical Operations

(1) The rules for expression evaluation should be carefully noted, with particular reference to priority of operations. The following examples show the kind of mistake that can occur:

X>Y|Z is not equivalent to X>Y|X>Z but is equivalent to (X>Y)|Z

X>Y>Z is not equivalent to X>Y&X>Z but is equivalent to (X>Y)>Z

The clause IF A=B||C is equivalent to IF A=(B||C),not to IF (A=B)||C

All operation sequences of equal priority are evaluated left to right, except for **, prefix +, prefix -, and ¬, which are evaluated right to left. Thus, the statement

```
A=B**-C**D;
```

is equivalent to

```
A=B**(-(C**D));
```

The normal use of parentheses is to modify the rules of priority; however, it may be convenient to use redundant parentheses as a safeguard or to clarify the operation.

(2) Conversion is governed by comprehensive rules which must be thoroughly understood if unnecessary trouble is to be avoided. Some examples of the effect of conversion follow.

(a) DECIMAL FIXED to BINARY FIXED can cause unexpected results if fractions are involved:

```
DCL I FIXED BIN(31,5) INIT(1);
    I = I+.1;
```

The value of I is now 1.0625. This is because .1 is converted to FIXED BINARY(5,4), so that the nearest binary approximation is 0.0001B (no rounding occurs). The decimal equivalent of this is .0625. A better result would have been achieved by specifying .1000 in place of .1. (See also item (f) below.)

(b) If arithmetic is performed on character string data, the intermediate results are held in the maximum precision:

```
DCL A CHAR(6) INIT('123.45');
DCL B FIXED(5,2);
B=A;   /*B HAS VALUE 123.45*/
B=A+A; /*B HAS VALUE 246.00*/
```

(c) The rules for arithmetic to bit string conversion affect assignment to a bit string from a decimal constant:

```
DCL A BIT(1),
    D BIT(5);
A=1; /*A HAS VALUE '0'B*/
D=1; /*D HAS VALUE '00010'B*/
D='1'B; /*D HAS VALUE
          '10000'B*/
IF A=1 THEN GO TO Y;
       ELSE GO TO X;
```

The branch will be to X, because the assignment to A resulted in the following sequence of actions:

(1) The decimal constant, 1, is assumed to be FIXED DECIMAL (1,0) and is assigned to temporary storage with the attributes FIXED BINARY(4,0), taking the value '0001'B;

(2) This value is now treated as a bit string of length (4), so that it becomes '0001'B;

(3) The resultant bit string is assigned to A. Since A has a declared length of 1, and the value to be assigned has acquired a length of 4, truncation occurs at the right, and A has a final value of '0'B.

To perform the comparison operation in the IF statement, '0'B and 1 are converted to FIXED BINARY and compared arithmetically. They are unequal, giving a result of "false" for the relationship A=1.

In the first assignment to D, a sequence of actions similar to

that described for A takes place, except that the value is extended at the right with a zero, because D has a declared length that is 1 greater than that of the value to be assigned.

(d) Assignment of arithmetic values to character strings involves conversion according to the rules for LIST-directed output.

Example 1

```
DCL A CHAR(4),
    B CHAR(7);
A='0'; /*A HAS VALUE '0bbb'*/
A=0;   /*A HAS VALUE 'bbb0'*/
B=1234567; /*B HAS VALUE
             'bbb1234'*/
```

Note: The three blanks are necessary to allow for the possibility of a minus sign and/or a decimal or binary point, with provision for a single leading zero before the point.

Example 2

```
DCL CTLNO CHAR(8) INIT('0');
DO I=1 TO 100;
    CTLNO=CTLNO+1;
    .
    .
    .
END;
```

In this example, a conversion error occurs because of the following sequence of actions:

(1) The initial value of CTLNO, that is, '0bbbbbbb', is converted to FIXED DECIMAL(5,0) for the addition, giving a temporary value of 00000.

(2) The decimal constant, 1, assumed to be FIXED DECIMAL(1,0), is added; in accordance with the rules for addition, the precision of the result is (6,0), giving a value of 000001.

(3) This value is now converted to a character string of length 9, value 'bbbbbbbb1', in preparation for the assignment back to CTLNO.

(4) Because CTLNO has a length of 8, the assignment causes truncation at the right; thus, CTLNO has a final value that consists entirely of blanks. This value cannot be successfully converted to arithmetic type for the second iteration of the loop.

(e) FIXED division can result in unexpected overflows or truncation. For example, the expression

```
25+1/3
```

would yield a value of 5.33...3. To obtain a result of 25.33...3, it would be necessary to write

```
25+01/3
```

The explanation is that constants have the precision and scale factor with which they are written, while FIXED division results in a value of maximum implementation-defined precision. The results of the two evaluations are reached as follows:

Item	Precn/Scale Factor	Result
1	(1,0)	1
3	(1,0)	3
1/3	(15,14)	0.33333333333333
25	(2,0)	25
25+1/3	(15,14)	5.33333333333333 (truncation on left; FIXEDOVERFLOW would be raised unless disabled)
01	(2,0)	01
3	(1,0)	3
01/3	(15,13)	00.3333333333333
25	(2,0)	25
25+01/3	(15,13)	25.3333333333333

(f) Checking of a picture is performed only on assignment into the picture variable:

```
DCL A PIC'999999',
    B CHAR(6) DEF A,
    C CHAR(6);
B='ABCDEF';
C=A; /*WILL NOT RAISE CONV
       CONDITION*/
A=C; /*WILL RAISE CONV*/
```

Note also (A, B, C as declared above):

```
A=123456; /*A HAS VALUE
            123456*/
          /*B HAS VALUE
            '123456'*/
```

```
C=123456; /*C HAS VALUE
                'bbb123'*/
C=A; /*C HAS VALUE '123456'*/
```

(g) A FIXED DECIMAL scalar with a declared even precision (P,Q) may have an effective precision of (P+1,Q), as the high-order byte may not be non-zero. The SIZE condition can be used to eliminate this effect:

```
DCL (A,B,C) FIXED DECIMAL (6,0);
    ON SIZE;
      .
      .
      .
(SIZE): A = B + C;
```

This ensures that the high-order byte of A is zero after the assignment.

g. DO groups

(1) The scope of a condition prefix applied to a DO statement is limited to execution of the statement itself; it does not apply to execution of the entire group.

(2) An iterative DO group is not executed if the terminating condition is satisfied at initialization:

```
I=6;
DO J=I TO 4;
   X=X+J;
END;
```

X is not altered by this group, since BY 1 is implied. Iterations can step backwards, and if BY -1 had been specified, three iterations would have taken place.

(3) Expressions in a DO statement are assigned to temporaries with the same characteristics as the expression, not the variable. For example:

```
DCL A DECIMAL FIXED(5,0);
    A=10;
    DO I=1 TO A/2;
    .
    .
    .
    END;
```

This loop will not be executed, because A/2 has decimal precision (15,10), which, on conversion to binary (for comparison with I), becomes binary (31,34).

Five iterations would result if the DO statement were replaced by

```
ITEMP=A/2;
DO I=1 TO ITEMP;
```

(4) DO groups cannot be used as ON-units.

(5) Upper and lower bounds of iterative DO groups are computed once only, even if the variables involved are reassigned within the group. This applies also to the BY expression.

Any new values assigned to the variables involved would take effect only if the DO group was started again.

(6) In a DO group with both a control variable and a WHILE clause, the evaluation and testing of the WHILE expression is carried out only after determination (from the value of the control variable) that iteration may be performed. For example, the following group would be executed at most once:

```
DO I=1 WHILE(X>Y);
.
.
.
END;
```

(7) I is frequently used as the control variable in a DO group, for example:

```
DO I=1 TO 10;
```

Within the scope of this implicit declaration, I might be contextually declared as a pointer, for example:

```
DCL X BASED(I);
```

The two statements are in conflict and will produce a diagnostic message. When I is a pointer variable, it can only be used in a DO group in one of the following ways:

1.
```
DCL (P, IA, IB, IC) POINTER;
    .
    .
    .
DO P=IA,IB,IC;
```

2.
```
DCL (P, IA) POINTER;
    .
    .
    .
DO WHILE(P=IA);
```

h. Data Aggregates

(1) Array arithmetic should be thought of as a convenient way of specifying an iterative computation. For example:

```
DCL A(10,20);
.
.
.
A=A/A(1,1);
```

has the same effect as

```
DCL A(10,20);
.
.
.
DO I=1 TO 10;
DO J=1 TO 20;
A(I,J)=A(I,J)/A(1,1);
END; END;
```

Note that the effect is to change the value of A(1,1) only, since the first iteration would produce a value of 1 for A(1,1). If the programmer wished to divide each element of A by the original value of A(1,1), he could write

```
B=A(1,1);
A=A/B;
```

or alternatively,

```
DCL A(10,20),
    B(10,20);
.
.
.
B=A/A(1,1);
```

(2) Note the effect of array multiplication:

```
DCL (A,B,C) (10,10);
.
.
.
A=B*C;
```

This does not effect matrix multiplication; it is equivalent to:

```
DCL (A,B,C) (10,10);
.
.
.
DO I=1 TO 10;
DO J=1 TO 10;
A(I,J)=B(I,J)*C(I,J);
END; END;
```

i. Strings

(1) Assignments made to a varying string by means of the SUBSTR pseudo-variable do not set the length of the string. A varying string initially has an undefined length, so that if all assignments to the string are made using the SUBSTR pseudo-variable, the string still has an undefined length and cannot be successfully assigned to another variable or written out.

(2) The user must ensure that the lengths of intermediate results of string expressions do not exceed 32767 bytes. This applies particularly to variable string lengths, as there is no object-time length checking. (I)

j. Functions and Pseudo-Variables

(1) When UNSPEC is used as a pseudo-variable, the expression on the right is converted to a bit string. Consequently, the expression must not be invalid for such conversion; for example, if the expression is a character string containing characters other than 0 or 1, a conversion error will result.

k. ON-Conditions and ON-Units

(1) Note the correct positioning of the ON statement. If the specified action is to apply when the named condition is raised by a given statement, the ON statement must be executed before that statement. The statements:

```
GET FILE (ACCTS) LIST (A,B,C);
ON TRANSMIT (ACCTS) GO TO TRERR;
```

would result in the ERROR condition being raised in the event of a transmission error during the first GET operation, and the required branch would not be taken (assuming that no previous ON statement applies). Furthermore, the ON statement would be executed after each execution of the GET statement.

(2) An ON-unit cannot be entered by means of a GOTO statement. To execute an ON-unit deliberately, the SIGNAL statement can be used.

(3) CONVERSION ON-units entered as a result of an invalid conversion (as opposed to SIGNAL) should either change the invalid character (by means of the ONSOURCE or ONCHAR pseudo-variable), or else terminate with a GOTO statement. Otherwise, the system will print a message and raise the ERROR condition.

(4) At normal exit from an AREA ON-unit, the standard system action is to try again to make the allocation. As a result the ON unit will be entered again, and an indefinite loop will be created. To avoid this, the amount allocated should be modified in the ON unit, for example, by using the EMPTY built-in function or by changing a pointer variable.

1. Input/Output

(1) The UNDEFINEDFILE condition may be raised if a STREAM file is reopened with attributes or options that conflict with attributes, options, or parameters previously specified for it. For example, if a file originally opened with a LINESIZE of 100 is subsequently reopened with a LINESIZE of 131, the UNDEFINEDFILE condition will be raised if the DCB subparameter BLKSIZE is not specified on the DD card, or if it is specified as less than 132. Difficulties of this nature can be avoided by the use of different file names, or by using the same file name with different TITLE option specifications. (I)

(2) The UNDEFINEDFILE condition is raised not only by conflicting language attributes (such as DIRECT with PRINT), but also by the following:

(a) Block size smaller than record size. This condition is not raised if spanned (VS- or VBS-format) records are used.

(b) LINESIZE exceeding the permitted maximum.

(c) Blocked records specified for REGIONAL organization.

(d) U- or V-format records specified for INDEXED, REGIONAL(1), or REGIONAL(2) organizations.

(e) KEYLEN not specified for creation of INDEXED, REGIONAL(2), or REGIONAL(3) data sets.

(f) Attempting to open an INDEXED data set for DIRECT OUTPUT.

(g) Attempting to open a CONSECUTIVE data set with DIRECT or KEYED attributes.

(h) Specifying an RKP option, for an INDEXED data set, with a value resulting in KEYLEN+RKP exceeding LRECL.

(i) Specifying a V-format logical record length of less than 18 bytes for STREAM data sets.

(j) Specifying, for F-format blocked records, a block size which is not an integral multiple of the recordsize.

(k) Specifying, for V-format records, a logical record length that is not at least four bytes smaller than the specified block size.

(l) Attempting to open a paper-tape reader for OUTPUT or UPDATE.

(m) Attempting to open a file with the UNBUFFERED attribute for blocked records.

(n) Attempting to use blocked records in the system input stream (SYSIN DD DATA or SYSIN DD *) with an UNBUFFERED file. The default record format for the system input stream is FB-format. Since this stream is not checked on input, the presence of FB-format records will not be detected until an attempt is made to open the file, when UNDEFINEDFILE will be raised.

Note: If the UNDEFINEDFILE condition is raised because either the key length or the block size is not specified, a subsequent attempt to open the file will not raise this condition again.

(I)

(3) If a file is to be used for both input and output, it must not be declared with either the INPUT or the OUTPUT attribute. The required option can be specified on the OPEN statement. There must be no conflict between file attributes specified in the declaration and those specified by the OPEN statement.

(4) Input/output lists must be surrounded by a pair of parentheses; so must iteration lists. Therefore, two pairs of outer parentheses are required in

```
GET LIST ((A(I) DO I=1 TO N));
```

(5) The last eight bytes of a source key to access a regional data set must be the character string representation of a fixed decimal integer. When generating the key, the rules for arithmetic to character string conversion should be considered. For example, the following group would be in error:

```
DCL KEYS CHAR(8);
DO I=1 TO 10;
   KEYS=I;
   WRITE FILE(F) FROM (R)
     KEYFROM (KEYS);
END;
```

The default for I is FIXED BINARY(15,0), which requires not 8 but 9 characters to contain the character string representation of the arithmetic values.

(6) Note that the file must have the KEYED attribute if the KEY, KEYFROM, or KEYTO options are to be used in any input/output statement referring to that file.

(7) The standard file names SYSIN and SYSPRINT are implicit only in GET and PUT statements. Any other reference, such as those in ON statements or RECORD-oriented input/output statements, must be explicit.

(8) PAGESIZE and LINESIZE are not file attributes, that is, they cannot be included in a DECLARE statement for the file; they are options on the OPEN statement.

(9) When an EDIT-directed data list is exhausted, no further format items will be processed, even if the next format item does not require a matching data item. For example:

```
DCL A FIXED(5),
    B FIXED(5,2);
GET EDIT (A,B) (F(5),F(5,2),X(70));
```

The X(70) format item will not be processed. To read a following card with data in the first ten columns only, the SKIP option can be used:

```
GET EDIT (A,B) (F(5), F(5,2)) SKIP;
```

(10) The number of data items represented by an array or structure name appearing in a data list is equal to the number of scalar elements in the array or structure; thus if more than one format item appears in the format list, successive elements will be matched with successive format items. For example:

```
DCL 1 A,
      2 B CHAR(5),
      2 C FIXED(5,2);
.
.
.
PUT EDIT (A) (A(5),F(5,2));
```

B will be matched with the A(5) item, and C will be matched with the F(5,2) item.

(11) Arrays are transmitted in row major order (e.g., A(1,1), A(1,2), A(1,3), ... A(2,1), ... etc.)

(12) Strings used as input data for GET DATA and GET LIST must be enclosed in quotation marks.

(13) The 48-character representation of a semicolon (,.) is not recognized as a semicolon if it appears in a DATA-directed input stream; the 11-8-6 punch must be used. (I)

(14) If a new record is added by DIRECT access to an INDEXED data set whose overflow areas are already full, a record will be irretrievably lost. The position of the new record, in relation to the existing records on the track, will determine whether it is the new record or an existing one which is lost. If the new record would follow the last existing record on the track, the new record will be lost. Otherwise, the last existing record on the track will be lost. In either case, the KEY condition will be raised.

(15) The user must be aware of two limitations of PUT DATA; (i.e., no data list). Firstly, its use with an ON statement is restricted because the data known to PUT DATA would be the data known at the point of the ON unit. Secondly, and more serious, the data will be put out as normal data-directed output, which means that any unallocated or unassigned data may raise a CONVERSION or other condition.

If the ON-unit

```
ON ERROR PUT DATA;
```

is used in an outer block, it must be remembered that variables in inner blocks are not known and therefore

will not be dumped. It would be a good practice, therefore, to repeat the ON-unit in all inner blocks during debugging.

If an error does occur during execution of the PUT DATA statement, and this statement is within an ON ERROR unit, the program will recursively enter the ERROR unit until no more storage remains for the DSA. Since this could be wasteful of machine time and printout, the ERROR unit should be turned off once it is activated. Instead of:

```
ON ERROR PUT DATA;
```

better code would be:

```
ON ERROR BEGIN;
        ON ERROR SYSTEM;
        PUT DATA;
        END;
```

When PUT DATA is used without a data-list every variable known at that point in the program is transmitted in data-directed output format to the specified file. Users of this facility, however, should note that:

a) Uninitialized FIXED DECIMAL data may raise the CONVERSION condition or a data interrupt.

b) Unallocated CONTROLLED data will cause arbitrary values to be printed and, in the case of FIXED DECIMAL, may raise the CONVERSION condition or a data interrupt.

(16) Use of locate mode I/O. A pointer set in READ SET or LOCATE SET may not be valid beyond the next operation on the file, or beyond a CLOSE statement. In OUTPUT files, WRITE and LOCATE statements can be freely mixed.

For UPDATE files, the REWRITE statement with no options must be used if it is required to rewrite an updated record. The result of this REWRITE is always to rewrite the contents of the last buffer onto the data set.

For example:

```
3   READ FILE (F) SET (P);
              .
              .
              .
5   P->R = S;
              .
              .
              .
7   REWRITE FILE (F);
              .
              .
              .
11  READ FILE (F) INTO (X);
              .
              .
              .
15  REWRITE FILE (F);
              .
              .
              .
19  REWRITE FILE (F) FROM (X);
```

Notes:

Statement 7 will rewrite a record updated in the buffer.
Statement 15 will only rewrite exactly what was read, i.e., it will not change the data set at all.
Statement 19 will raise ERROR, since there is no preceding READ statement.

There are two cases where it is not possible to check for the KEY condition on a LOCATE statement until transmission of a record is attempted. (This will generally occur on execution of the next PL/I output statement for this file.)

These are:

1. When there is insufficient room in the specified region to output the record on a REGIONAL(3) V- or U-format file. Neither the record raising the condition nor the current record are transmitted.

2. When the embedded key differs from the KEYFROM in an ISAM file.

If this LOCATE statement is to transmit the last record before the file is closed, in case 1, the record is not transmitted, and in the second case, the embedded key is overwritten with the KEYFROM string, and the record is transmitted.

Thus the condition may be raised by a CLOSE statement or by an END statement that causes implicit closing. Until the error is corrected, the record cannot be transmitted nor can any further operation be carried out on the file.

If a LOCATE statement was used in the addition of KEYED records to any type of REGIONAL data set then, if RECORD condition is raised, the key value presented at subsequent operations must not be less than the current one.

(18) Allocation and freeing of BASED Variables: If a reference is made, at object time, to a BASED variable that has not been allocated storage, an unpredictable interrupt (protection, addressing or specification) may occur.

(19) Areas, pointers, offsets and structures containing any of these cannot be used with STREAM I/O. PUT DATA cannot be used with BASED variables.

When a BASED variable is freed, the associated pointer no longer contains useful information. This pointer can only be used again if:

1. It is re-allocated with the same or another BASED variable, or,

2. A value is assigned to it from an offset or another pointer

A BASED variable allocated in an area must be freed in that area. For example:

```
DCL A AREA, B BASED (X);
ALLOCATE B IN (A);
   .
   .
   .
FREE B;              /* ILLEGAL */
FREE B IN (A);       /* LEGAL   */
```

2. PROGRAMMING FOR INCREASED EFFICIENCY

In PL/I there are often several different ways of producing a given effect. One of these ways will usually be more efficient from a particular point of view than another, depending largely on the method of implementation of the language features concerned. However, it should be realized at the outset that a primary cause of program inefficiency occurs at the problem definition stage, before any actual programming is done: PL/I cannot be used to full advantage unless the problem is defined in terms of PL/I.

The purpose of this section is to help the programmer make the best use of the PL/I (F) Compiler. The first two parts are presented from two different viewpoints:

a. Improving the speed of compilation

b. Improving the speed of execution.

The remainder of the section is of common interest, and deals with methods of decreasing the dictionary size; use of storage; use of compile-time facilities; use of input/output facilities; and additional hints.

a. Improving Speed of Compilation

The following measures are suggested for use where compilation time is an important factor.

(1) Allocate as much storage to the compiler as possible, using the SIZE=999999 option on the EXEC statement. This reduces the chances of bringing the spill mechanism into operation.

(2) Keep the number of BEGIN blocks and procedures to a minimum. Do not use BEGIN-END to effect statement grouping; this is more simply obtained by use of DO-groups.

(3) Try to avoid using the ATR, XREF, LIST, DECK, and CHAR48 compiler options.

(4) Avoid features which give rise to large dictionary entries and large amounts of text. For example,

```
DCL A PIC '(4000)X';
```

occupies one complete dictionary block.

(5) Use of the following features causes optional compiler phases to be loaded: ALLOCATE, LIKE, USES/SETS, CHECK, iSUB defining, built-in functions with aggregate arguments, GENERIC entry names, DELAY/DISPLAY. If any of these features can be completely avoided without extensively increasing the source code, there will be a corresponding increase in compilation speed.

(6) On re-runs, further slight increases in efficiency can be obtained by

(a) removing all unreferenced labels and data;

(b) correcting all source errors, and, where possible, minimizing the number of diagnostic messages produced, including such messages as "FILE/STRING option missing in GET/PUT statement";

(c) subdividing the program if the auxiliary storage has been used;

(d) specifying the NOSOURCE option;

(e) specifying the FLAGS option.

See also 2.c. Decreasing Size of Dictionary.

b. Improving Speed of Execution

The following measures are suggested for use where execution time is an important factor. Note that while some of these measures may slow down the compilation, this is offset by the fact that others will accelerate it. In the main, there should be no serious increase in compilation time.

(1) Make use of the OPT=01 option. Avoid the STMT option.

(2) Avoid unnecessary program segmentation and block structure; all procedures, ON-units and BEGIN blocks need prologues and epilogues, the initialization and housekeeping for which carry a considerable overhead. (Prologues and epilogues are described in Appendix C of this publication.) Whenever possible, use GOTO or IF statements to control program logic, rather than the CALL statement.

(3) Branching in IF statements can be improved by using DO and END statements to bracket a THEN clause, rather than using a GOTO statement in the THEN clause. For example:

```
IF A=B THEN DO;
   C=D;
   E=F;
            END;
L: etc.
```

is more efficient than

```
IF A¬=B THEN GO TO L;
   C=D;
   E=F;
L: etc.
```

(4) When GO TO is used in an IF statement, more efficient object code is produced by the GO TO if it refers to a label within the same block rather than to a label outside the block.

(5) Keep IF clauses simple; separate any multiple conditions into a series of simple IF statements. For example:

```
IF A=B
  THEN IF C=D
    THEN IF E=F
      THEN GO TO M;
```

is more efficient than

```
IF (A=B)&(C=D)&(E=F)
  THEN GO TO M;
```

(6) Avoid extensive use of adjustable arrays and/or CONTROLLED storage.

(7) Use constants wherever possible instead of expressions.

(8) Exercise care in specifying precision. For example,

```
DCL A FIXED DEC(8,4),
    B FIXED DEC(10,2),
    C FIXED DEC(10,1);
    .
    .
    .
C=A+B;
```

This requires almost twice as much code as it would if B had been declared (10,4), because the evaluation of A+B requires a scale factor of 4.

(9) Use the PICTURE attribute only when necessary. For example, use FIXED DECIMAL(5,2) instead of PIC'999V99'. If a picture field is used in more than one arithmetic operation, convert it once and then use the new form in each operation. This holds for any conversion required more than once.

If it is necessary to use data with the PICTURE attribute in arithmetic expressions, use pictures that will be handled in-line, as this considerably reduces execution time. Pictures with all 9s, a V and a non-drifting sign are particularly useful. For example:

```
'999'
'$99v99'
's99'
'V999'
```

(10) Internal switches and counters, and data involved in substantial computation or used for subscripts, should be declared BINARY; data required for output should be kept in DECIMAL form.

(11) Keep data conversions to a minimum. Some possible methods follow:

(a) Use additional variables. For example, if a problem specifies that a character variable has to be regularly incremented by 1,

```
DCL CTLNO CHAR(18);
.
.
.
CTLNO = CTLNO+1;
```

requires two conversions, while

```
DCL CTLNO CHAR(8),
   DCTLNO DEC FIXED;
.
.
.
DCTLNO=DCTLNO+1;
CTLNO=DCTLNO;
```

requires only one conversion.

(b) Take special care to make structures match when it is intended to move data from one structure to another.

(c) Avoid mixed mode arithmetic, especially the use of character strings in arithmetic calculations.

(12) Declare arrays in the procedure in which they are used, instead of passing them as arguments. Declare subscript variables in the block in which they are used, as FIXED BINARY.

(13) In multiple assignments to subscripted variables, restrict the assignment to three variables.

(14) If a subscripted item is referred tc more than once with the same subscript, assign the element to a scalar variable:

```
R=(A(I)+1/A(I))+A(I)**A(I);
```

should be replaced by

```
ASUB=A(I);
R=(ASUB+1/ASUB)+ASUB**ASUB;
```

(15) Bit strings should, if possible, be specified as multiples of eight bits. Bit strings used as logical switches should be specified according to the number of switches required. In the examples below, (a) is preferable to (b), and (b) to (c):

1. Single Switches

(a)
```
DCL SW BIT(1) INIT ('1'B);
.
.
.
IF SW THEN DO;
.
.
.
```

(b)
```
DCL SW BIT(8) INIT('1'B);
.
.
.
IF SW THEN DO;
.
.
```

(c)
```
.
DCL SW BIT(8) INIT('1'B);
.
.
.
IF SW = '10000000'B THEN DO;
.
.
.
```

2. Multiple Switches

(a)
```
DCL B BIT(8);
.
.
.
B = '11100000'B;
.
.
.
IF B = '11100000'B THEN DO;
.
.
.
```

(b)
```
DCL B BIT(3);
.
.
.
B = '111'B;
.
.
.
IF B = '111'B THEN DO;
.
.
.
```

(c)
```
DCL (SW1,SW2,SW3) BIT(1);
.
.
.
SW1, SW2, SW3, = '1'B;
.
.
.
IF SW1&SW2&SW3 THEN DO;
.
.
.
```

If bit string data is to be held in structures, such structures should be declared ALIGNED.

(16) Note that concatenation operations are time-consuming.

(17) Varying length strings are not as efficient as fixed length strings.

(18) Fixed length strings are not efficient if their length is not known at compile time, as in the following example:

```
DCL A CHAR(N);
```

(19) Avoid using the SIZE, SUBSCRIPTRANGE, STRINGRANGE and CHECK ON-conditions, except during debugging. Debugging aids (see "Testing Programs" in this publication) should be removed from the program before running it as a production job.

(20) Do not refer to the DATE built-in function more than once in a run; it is expensive. Instead, refer to the function once and save the value in a variable for subsequent use; e.g. instead of

```
PAGEA= TITLEA||DATE;
PAGEB= TITLEB||DATE;
```

it is more efficient to write

```
DTE=DATE;
PAGEA=TITLEA||DTE;
PAGEB=TITLEB||DTE;
```

(21) Allocate sufficient buffers to prevent the program becoming I/O bound.

(22) Use blocked output records.

(23) Open a number of files in a single OPEN statement.

(24) In STREAM input/output, use long data lists instead of splitting up input/output statements.

(25) Use EDIT-directed input/output in preference to LIST- or DATA-directed.

(26) Consider the use of overlay defining to simplify transmission to or from a character string structure. For example:

```
DCL 1 IN,
      2 TYPE CHAR(2),
      2 REC,
        3 A CHAR(5),
        3 B CHAR(7),
        3 C CHAR(66);
GET EDIT(IN) (A(2),A(5),A(7),A(66));
```

In the above example, each format-item/data-field pair is matched separately, code being generated for each matching operation. It would be more efficient to define a character string on the structure and apply the GET statement to the string:

```
DCL STRNG CHAR(80) DEF IN;
.
.
.
GET EDIT (STRNG) (A(80));
```

(Input/output is also discussed in item 2.e., below.)

(27) Many operations which, in earlier versions of the PL/I (F) compiler, were handled by calls to PL/I library routines, are now handled in-line. The consequent saving in execution time for these operations can be of the order of ten to one or more; the overall effect on program execution will obviously depend on the number of times these operations are used in a program. It will repay the user, therefore, to recognise which operations are performed in-line and which require a library call, and to arrange his program to use the former wherever possible. The majority of these in-line operations are concerned with data conversion and string handling.

Data Conversion: The data conversions performed in-line are shown in Figure 25. A conversion outside the range or condition given, or marked 'Not done' is performed by a library call.

Not all the picture characters available may be used in a picture involved in an arithmetic conversion. The only ones permitted are:

1. V and 9
2. Drifting or non-drifting characters $, S, +, -
3. Zero suppression characters Z, *
4. Punctuation characters ,, ., /, B

For in-line conversions, pictures with this subset of characters are divided into three types:

1. Pictures of all 9s with (optionally) a V and a leading or trailing sign.
   Example: '99V999', '99', 'S99V9', '99V+', '$999'
2. Pictures with zero suppresion characters and (optionally) punctuation characters and a sign character. Also, type 1 pictures with punctuation characters.
   Example: 'ZZZ', '**/**9', 'ZZ9V.99', '+ZZ.ZZZ', '$///99', '9.9'
3. Pictures with drifting strings and (optionally) punctuation characters and a sign character.
   Example: '$$$$', '-,--9', 'S/SS/S9', '+++9V.9','$$$9-'

Conversion		Comments and Conditions	Minimum Optimization Code	
Source	Target		SIZE Disabled	SIZE Enabled
FIXED BINARY	FIXED BINARY	-	0	0
	FIXED DECIMAL	If either scale factor = 0 and the other scale factor ≤ 0, then the opt. code may be 0	1	1
	FLOAT	If source scale factor = 0, then the opt. code may be 0(whether SIZE is enabled or not)	1	1
	Bit string	String must be fixed-length, ALIGNED, and with length <256	0	Not done
	Character string or Picture	Source scale factor must be ≥ 0 String must be fixed-length with length <256 Picture types 1, 2 or 3	1	Not done
FIXED DECIMAL	FIXED BINARY	If source and target scales have the same sign and are non-zero, then the opt. code (SIZE disabled) must be 1	0	1
	FIXED DECIMAL	-	0	0
	FLOAT	Source precision must be <10	1	1
	Bit String	Source scale factor must be zero String must be fixed-length, ALIGNED, and with length <256	0	Not done
	Character string	Source scale factor must be ≥ 0 String must be fixed-length and length <256	1	1
	Picture	Picture types 1, 2 and 3 For picture types 1 and 2 with no sign, the Opt. code may be 0	1	Not done
FLOAT	FIXED BINARY		1	Not done
	FIXED DECIMAL	Target precision must be ≤9	1	Not done
	FLOAT	Source and target may be single or double length	0	0
	Bit string	String must be fixed-length, ALIGNED, and with length <256	1	Not done
Bit string	FIXED BINARY	Source string must be fixed-length, ALIGNED, and with length <256	0	Not done
	FIXED DECIMAL and FLOAT	Source must be fixed-length, ALIGNED, and with length <32	1	Not done
	Bit string	Source and target must be ALIGNED with length <2040	0	0

Figure 25. Implicit Data Conversions Performed In-Line

Conversion		Comments and Conditions	Minimum Optimization Code	
Source	Target		SIZE Disabled	SIZE Enabled
Character string: Fixed-length	Character string: Fixed-length	Target length must be ≤256	0	0
	VARYING	Source and target lengths must be ≤256	0	0
VARYING	Character string (VARYING)	Source and target lengths must be ≤256	0	0
Picture	Character string	String must be fixed-length with length <256	0	0
	Picture	Pictures must be identical	0	0
Picture type 1	FIXED BINARY	Source precision must be <10	1	Not done
	FIXED DECIMAL	If picture has a sign, then the opt. code must be 1	0	Not done
	FLOAT	Source precision must be <10	1	Not done
	Picture	Picture types 1, 2 or 3	1	Not done
Label	Label	-	0	0
	Pointer/Offset	Pointer/Offset	0	0

Figure 25. Implicit Data Conversions Performed In-Line (continued)

Sometimes a picture conversion is not performed in-line even though the picture is one of the above types. This may be because:

1. The optimization code value (OPT=nn) is too low,

2. SIZE is enabled,

3. There is no overlap between the digit positions in the source and target.
   Example: DECIMAL (6,8) or DECIMAL (5, -3) to PIC '999V99' will not be performed

4. The picture may have certain characteristics that make it difficult to handle in-line.
   Example: 'ZZ.99'

   a. Punctuation between a drifting Z or a drifting * and the first 9 is not preceded by a V

   b. Drifting or zero expression characters to the right of the decimal point
      Example: 'ZZV.ZZ', '++V++'

String Handling: The string functions and operations performed in-line are shown in Figures 26 and 27. It should be noted that even the string functions indicated as always being performed in-line may sometimes call a library routine. For example, if the expression in the BIT or CHAR functions requires an implicit conversion not handled in-line, the appropriate library routines will be called.

(27) If a file is declared DIRECT INDEXED, then the ENVIRONMENT options INDEXAREA and NOWRITE should be applied if possible.

c. Decreasing Size of Dictionary

If a compilation overflows the dictionary, the immediate solution is to subdivide the program, recompiling internal procedures as separate external procedures. If this is impracticable or undesirable, the dictionary requirements of the compilation must be reduced. The following cause large

String Operation	Comments and Conditions	
	Source	Target
Assign	Non-adjustable, ALIGNED, fixed-length bit string <2048 bits long	Non-adjustable, ALIGNED, fixed-length bit string <2048 bits long
	Non-adjustable, ALIGNED, bit string <2048 bits long	Non-adjustable, ALIGNED, VARYING bit string <2048 bits long
(OPT=0)	Non-adjustable, UNALIGNED, fixed-length bit string that is a scalar element of an AUTOMATIC, BASED or STATIC structure with no adjustable bounds or extents	Non-adjustable, UNALIGNED, fixed-length bit string that is a scalar element of an AUTOMATIC, BASED or STATIC structure with no adjustable bounds or extents. The string must be 1 bit long.
Note: The assignment VARYING string to fixed-length string is not handled in-line	Fixed-length or VARYING character string <256 characters long	Fixed-length or VARYING character string <256 characters long
'And', 'Not', 'Or'	Non-adjustable, ALIGNED, fixed-length or VARYING bit strings, with length: fixed-length - <2048 bits VARYING - ≤32 bits	
Compare	Non-adjustable fixed-length character strings <256 characters long Non-adjustable, ALIGNED, fixed-length or VARYING bit strings, with length: fixed-length - <2048 bits VARYING - ≤32 bits	
Concatenate	Non-adjustable fixed-length or VARYING character strings <256 characters long Non-adjustable, ALIGNED, fixed-length or VARYING bit strings ≤32 bits long	
STRING function	Scalars and non-adjustable contiguous array or structure variables	

Notes:
1. Operations with VARYING strings require OPT=1.
2. If the expression in an IF statement is a bit string satisfying the conditions for the source string when OPT=0, then, if the string is <10 bits long, in-line code is generated to test the value of the string.

Figure 26. Conditions under which the String Operations are Handled In-Line

dictionary requirements, and should be avoided if possible:

(1) Excessive use of labels.

(2) Use of structures rather than separate scalar variables.

(3) The LIKE attribute.

(4) CONTROLLED identifiers. If they are necessary, minimize the number of ALLOCATE statements in the program.

(5) Adjustable bounds and string lengths.

(6) The DEFINED attribute.

(7) Secondary entry points.

(8) Long BCD identifiers.

(9) Expressions as function arguments.

(10) iSUB defining.

(11) Passing subscripted structures or cross-sections of arrays as arguments to procedures.

(12) Explicitly qualified BASED variables. If it is necessary to use explicit pointer qualification, minimize the number of different pointer variables

used as explicit qualifiers of the same BASED variable.

String Function	Comments and Conditions
BIT	Always
BOOL	Non-adjustable, ALIGNED bit strings, where the third argument is one of the logical operators 'and', 'not', 'or' or exclusive 'or'
CHAR	Always
INDEX	Second argument must be a non-adjustable character string <256 characters long
LENGTH	Always
SUBSTR	STRINGRANGE must be disabled
UNSPEC	Always

Figure 27. Conditions under which the String Functions are Handled In-Line

d. Use of Storage

(1) Wherever possible, data for computation should be binary, rather than decimal.

(2) The following character string assignment causes a constant of 250 blanks to be generated at compile-time:

```
DCL A CHAR(250)
A='b';
```

To avoid generating such a large constant, the coding could be as follows:

```
DCL A CHAR(250)
DCL B CHAR(1);
B='b';
A=B;
```

B is padded, and no constant is generated.

If A had a length of 256 or more, then the assignment would be handled by a library call and a constant with only one blank would be generated at compile-time. In this case, the first coding example would be satisfactory.

(3) If a file declared as INDEXED is to be used for DIRECT UPDATE but will not have records added to it, the use of the ENVIRONMENT option NOWRITE will save data management about 5000 bytes of storage.

(4) The Alignment Attributes: These allow the user to provide alignment for string and arithmetic data as follows:

```
ALIGNED: Arithmetic:
           FIXED DECIMAL: byte
           FLOAT(DOUBLE): doubleword
           Other: word
         String: byte

UNALIGNED: Arithmetic and character
             string: byte
           Bit string: bit
```

Thus the UNALIGNED attribute can be used to obtain denser packing of data in main storage, with the minimum of padding.

Area, pointer, offset, label, task and event data are always aligned on word or doubleword boundaries (see Figure 49). They can never be UNALIGNED.

In data aggregates, the explicitly declared alignment for the aggregate applies to each element in the aggregate. In structures, however, this alignment can be overridden by an alignment specified for a particular base element.

For example:

```
DCL 1 STR UNALIGNED,
        2 A,
        2 B ALIGNED,
        2 C;
```

Here A and C will be UNALIGNED and B will be ALIGNED.

Default attributes now depend on the data type of the element concerned, both for data items and for data aggregates. These defaults are:

```
UNALIGNED  All string data and
           PICTURE items

ALIGNED    All arithmetic data i.e.,
                 BINARY
                 DECIMAL
                 FIXED
                 FLOAT
                 COMPLEX
```

For example:

```
DCL A BIT(4),
    I,
    (B CHAR(10), X) UNALIGNED,
    (C BIT(12), Y FIXED) ALIGNED;
```

Here A is UNALIGNED by default, I is ALIGNED by default, and B, C, X and Y, are as explicitly declared.

```
DCL (A1(80) CHAR(6), A2(80) BINARY)
    ALIGNED,
   B1 (3,3) BIT(2),
   C1 (3,3) CHAR(4),
   D1 (100) DECIMAL;
```

Here A1 and A2 are as explicitly declared, B1 and C1 are UNALIGNED by default, and D1 is ALIGNED by default.

```
DCL 1 A,
      2 B,
      2 C BIT(4) UNALIGNED,
      2 D ALIGNED,
        3 E BIT(2),
        3 F,
      2 G CHAR(10);
```

Here A i s a major structure
B is FLOAT DECIMAL ALIGNED by default
C is explicitly UNALIGNED
D is a minor structure
E is BIT ALIGNED (inherited from D)
F is FLOAT DECIMAL ALIGNED by default(here ALIGNED is inherited from D, but it is also the default for F if D had not been declared ALIGNED)
G is CHAR UNALIGNED

The user must take care that the alignment attributes are correct when matching variables for:

1. Use of the DEFINED attribute
2. Arguments and associated parameters
3. Accessing different generations of a based variable.

e. Use of Compile-Time Facilities

A facility is provided which allows the programmer to manipulate and modify his source text during compilation. This facility can be used to effect conditional compilation of sections of PL/I code, to expand sections of PL/I code in line (e.g., DO loops), to insert text into a program from an external library (see below), and to replace or eliminate PL/I identifiers (e.g., variable names, keywords, etc.). In order to use this facility, the MACRO option must be specified on the EXEC control card and a SYSUT3 DD statement must be included in the job control cards for the compilation. The requirements for this data set are given in the section "Job Processing." If the SOURCE2 option is specified on the EXEC card, a listing of the input to the compile-time processor is produced.

The implementation restrictions for this facility are listed in Appendix B of this publication.

Using the Compile-time INCLUDE Statement: A string of text from an external library can be inserted into a program by using the compile-time statement INCLUDE. This text must be a member of a partitioned data set and may consist of either PL/I source statements or compile-time statements (including another INCLUDE) or a mixture of the two. A DD statement for the data set to be included must be present in the job control cards for the compilation.

The identifier pairs in the INCLUDE statement specify the ddname which occurs in the name field of the appropriate DD card, and a data set member name, in that order. The first identifier, the ddname, is optional. If it is not specified, the default data set SYSLIB is assumed. In this case, a SYSLIB DD card is required. The second identifier, the data set member name, is not optional. If it is missing, the statement is considered to be incorrect. Therefore, if only one identifier is specified, it is assumed to be the member name.

Restrictions on data sets for the INCLUDE statement are specified in Appendix B of this publication.

A library of text can be created by using the operating system update utility routine called IEBUPDTE. This facility is described in the publication IBM System/360 Operating System, Utilities.

The following examples illustrate the creation of a partitioned data set and the use of the INCLUDE statement to insert text into a source program.

Example 1: Create Library

```
//JOB1     JOB  123,JOHNSMITH,
                MSGLEVEL=1

//STEP1    EXEC PGM=IEBUPDTE,
                PARM=NEW

//SYSPRINT DD   SYSOUT=A

//SYSUT2   DD   DSNAME=MACROLIB,
                DISP=(NEW,KEEP),
                UNIT=2311,
                VOLUME=SER=NNNNNN,
                SPACE=(80,(25,25,1))
```

```
//SYSIN     DD  *

./         ADD   NAME=DCL,LEVEL=00,SOURCE=0

./      NUMBER   NEW1=DC000000,INCR=00000010

                 % DCL(I,J)FIXED;
                 % I=10;

./         ADD   NAME=DCLP,LEVEL=00,SOURCE=0

./      NUMBER   NEW1=D0500000,INCR=00000010

                 VAR(1)=15;

                 % DO J=2 TO 5;

                 VAR(J)=10*(VAR(J-1)+J);

                 % END;

./       ENDUP

/*
```

Comment:

In the foregoing example, a library of text called MACROLIB is created, with two members, named DCL and DOLP.

Example 2: INCLUDE Library

```
//JOB2          JOB  123,JOHNSMITH,
                   MSGLEVEL=1

//STEP1        EXEC  PROC=PL1LFC,
                   PARM.PL1L='MACRO,
                   SOURCE2'

//PL1L.PRIVLIB DD   DSNAME=MACROLIB,
                   DISP=(OLD,DELETE),
                   UNIT=2311,
                   VOLUME=SER=NNNNNN

//PL1L.SYSIN    DD  *

    PROC1:PROCEDURE OPTIONS (MAIN);

          DCL VAR(5) DEC FIXED;

          % INCLUDE PRIVLIB (DCL);

          % IF I1=10 % THEN % DO; VAR=0;

                              % END;

                        % ELSE % INCLUDE
                          PRIVLIB(DOLP);

          PUT DATA(VAR);

          END PROC1;

  /*
```

Comments:

The % INCLUDE PRIVLIB (DCL) statement causes the following text to be inserted:

```
% DCL(I,J)FIXED;
% I=10;
```

The % INCLUDE PRIVLIB (DOLP) statement causes the following text to be inserted:

```
VAR(1)=15;
% DO J=2 TO 5;
VAR(J)=10*(VAR(J-1)+J);
% END;
```

In this example, the member called DCL is read in, scanned, and the compile-time statements are executed, thus assigning a value of 10 to the compile-time variable I. If the value of I had not been 10, the source program produced by the compile-time processor would have been:

```
PROC1: PROCEDURE OPTIONS (MAIN);
       DCL VAR(5) DEC FIXED;
       VAR=0;
       PUT DATA(VAR);
       END PROC1;
```

However, since the value of I is 10, the source program generated in the example is:

```
PROC1: PROCEDURE OPTIONS (MAIN);
       DCL VAR(5) DEC FIXED;
       VAR(1)=15;
       VAR(2)=10*(VAR(2-1)+2);
       VAR(3)=10*(VAR(3-1)+3);
       VAR(4)=10*(VAR(4-1)+4);
       VAR(5)=10*(VAR(5-1)+5);
       PUT DATA(VAR);
       END PROC1;
```

f. Use of Input/Output Facilities

The characteristics of a data set (record format, length, blocking factor, etc.) will significantly alter the time overhead of programs performing a large amount of input/output. Use of the ENVIRONMENT attribute should be kept to a minimum in order to allow the DD statement to supply the optional data set characteristics for a given use of a program. In general, the use of two or more buffers is advantageous to achieve overlap between I/O transmission and computing time. Further, the blocking of records will generally save both time and space on a data set.

It is advantageous to open a number of files in a single OPEN statement, since separate opening activities (either explicit or implicit) will require the reloading of the OPEN modules from the system resi-

dence. It should also be considered that when a file is open, buffers or workspace and data management interface modules are occupying storage. Accordingly, if storage space is a prime consideration, the judicious use of the OPEN and CLOSE statements will help to control the available storage.

Of the three STREAM data transmission techniques available, DATA-directed will generally be the most costly, both in time and space (symbol table entries exist at object time for each data variable transmitted). LIST-directed I/O is available for free format I/O, but the greatest degree of control is available by using EDIT-directed facilities, which is generally the most efficient technique, both in object time space and execution.

RECORD I/O offers facilities for handling data aggregates as single data entities, rather than element by element. Certain advantages, accordingly, are available in terms of efficient access to data set records and efficient use of data set space, since the data is unconverted and unedited.

When using variable-length (V-format) records, the use of spanned (VS- or VBS-format) records should be considered. Spanned records permit the programmer greater flexibility in selecting the optimum block size for the most efficient input/output-processor overlap and external storage utilization.

When using the INDEXED data set organization, accessing records in an overflow area can slow processing considerably. It is recommended that INDEXED data sets are regularly recreated so that logically-deleted but physically-present records are purged from the data set and records are collected from the overflow areas into the prime data areas. Another good reason for making this a regular practice is that when the overflow areas are full, records can be irretrievably lost.

Consideration should be given especially to the choice between the BUFFERED and UNBUFFERED file attributes, available for SEQUENTIAL access. A BUFFERED file permits the object program to perform "anticipatory buffering", that is, overlap of device transmission and computing time. An UNBUFFERED file, though sometimes saving space required for buffers, does not permit such overlap, unless the EVENT option is used. (It should be noted, in any case, that an UNBUFFERED file will require hidden buffers if the record format is V, or if the data set is INDEXED, or REGIONAL(2) or (3).)

RECORD Condition: Depending upon the record format employed, the RECORD condition may or may not be raised when the length of a record variable differs from the length of records within the data set. Figure 28 indicates when the condition is raised, and the results of transmissions before raising the condition (a dash indicates a valid situation, the condition not being raised). Note that if the record variable in the READ statement is the null string, no data is transmitted into the variable but a record is transmitted from the data set. In this case, the RECORD condition is raised.

If the RECORD condition is raised on a LOCATE statement, the pointer variable is not set and the contents of the previously located buffer will not be transmitted.

ONKEY Built-in Function: The value returned by this function is the key of the record that caused an I/O condition to be raised. This value depends on the condition raised and the key option specified in the I/O statement:

1. ERROR Condition raised because the I/O statement is invalid:

Option	Value returned
KEY|KEYFROM	KEY|KEYFROM String
KEYTO	Null String

2. I/O condition other than ENDFILE raised, or ERROR condition raised after data transmission is complete:

Option	Value returned
KEY|KEYFROM	Recorded key (or intended recorded key produced from the source string), with length=data-set key length; an exception is REGIONAL(1), where the value is the 8-byte character representation of the region number.
KEYTO	Recorded key, with length: Fixed: Length of KEYTO string VARYING: As for KEY|KEYFROM

3. ENDFILE condition raised:

Option	Value returned
KEYTO	Null string

4. Any condition, or no condition:

Option	Value returned
None	Null String[1]

Note: Under the following conditions:

Record Format	Mode			
	INPUT[1]		OUTPUT[2]	
	R1 < R2	R1 > R2	R1 < R2	R1 > R2
F	Truncate R2	Excess R1 length undefined#	Excess R2 length undefined	Truncate R1
U V VS VBS	Truncate R2*	-	-	Truncate R1

R1: Length of record variable
R2: Actual or maximum length of data set record
[1] If record variable is a VARYING string, then its current length is set equal to MIN(R_2, maximum length of record variable).
[2] R1=current length if the record variable is a VARYING string.
* If U-format and the file is UNBUFFERED (and requires no hidden buffers), truncation occurs, but the RECORD condition cannot be raised since this condition cannot be detected.
# A short record read from the data set will raise the TRANSMIT condition; the RECORD condition is raised only if the length of the record variable does not equal LRECL.

●Figure 28. Situations under which RECORD Condition is raised in RECORD-Oriented I/O

1. File is SEQUENTIAL INDEXED

2. RKP ≠ 0

3. The statement was a REWRITE

if the KEY condition is raised because the embedded key in the record variable or the buffer (if there is no FROM option) is not equal to the embedded key provided on the preceding READ, then ONKEY returns the key of the record provided by the READ. The length of this key is the length of the data set.

Input/Output Error Recovery: When I/O transmission errors are encountered, exhaustive recovery procedures are performed automatically by data management, so that, upon entry to a TRANSMIT ON-unit, it is unnecessary and impossible to perform further transmission error correction procedures. Synchronization of transmission errors and entry to relevant ON-units can only be guaranteed for input errors. An output operation error will be detected in a succeeding output operation, the particular one being dependent upon the blocking factor and the number of allocated buffers.

g. Additional Hints

(1) Operating System and Job Control

External procedures, but not internal procedures, are treated as separate control sections.

(2) Declarations and Attributes

(a) Do not rely too heavily on default attributes. Explicit declarations help to clarify the source program logic, and in some cases (for example, precision) reduce the chance of error.

(b) Variables declared FIXED BINARY or FLOAT BINARY are automatically aligned on the proper word boundary, regardless of whether they are single or part of an aggregate. FIXED DECIMAL variables are stored in packed decimal format and the System/360 decimal instructions are used in operations involving them. FLOAT DECIMAL variables are stored in floating-point format; operations involving them are carried out using the floating-point instruction set.

(3) Assignments and Initialization

(a) High order zeros will be inserted if required on assignment to or initialization of an arithmetic variable:

```
DCL A FIXED DECIMAL (5,2) INIT (12);
/*A HAS VALUE 012.00*/

DCL B FIXED BINARY (15,0);
B=12;
/*B HAS VALUE 000000000001100B*/
```

(b) Arrays may be initialized by

assignment from a scalar expression:

```
DCL A(10);
A=0;
```

The scalar value will be assigned to each element of the array. Similarly, when a scalar expression is assigned to a structure, its value will be assigned to each element of the structure:

```
DCL 1B,
    2 C BIT(1),
    2 D CHAR(1),
    2 E CHAR(4);
B=0;
```

As a result of this assignment, the values of the various elements will be:

```
C  '0'B
D  'b'
E  'bbb0'
```

(4) DO Groups

Iterations can step backwards, and the expression in the WHILE clause can refer to the control variable, e.g.,

```
DO I=N+2*L BY -X WHILE (I>0);
END;
```

The control variable can be modified within the loop.

It is possible to transfer from within a DO loop to a label on the END statement for the group. This has the effect of incrementing the control variable without intermediate processing; control will not fall through. It is also possible to transfer out of an iterative DO group before the terminating value of the control variable is reached.

(5) Functions

The arguments in a function reference can be modified by the function.

(6) ON-conditions and ON-units

Note the scope of condition prefixes:

```
 (SIZE):A:PROC;
          .
          .
          .
(NOSIZE):IF M>N THEN DO;
           J=E+F;
           END;
          .
          .
          .
         END A;
```

In the above example, SIZE is disabled only during the evaluation of the expression M>N; SIZE is enabled for the assignment J=E+F.

Programs can be checked using the debugging facilities provided by PL/I. When the user is satisfied that his program is working correctly, debugging statements should be removed from the source deck, which is then recompiled to produce an optimum object program ready for execution.

DEBUGGING FACILITIES

Certain language features are provided in PL/I to assist the programmer to debug his program. The facilities include:

Control over interruptions and error handling

The ability to obtain a trace of active procedures

Symbolic output

Communication with the program during execution

The use of specific language features provides the debugging facilities; in addition, the programmer may use his own techniques, such as inserting PUT statements at selected points.

Control of Interruption and Error Handling

Some conditions may be enabled or disabled by means of the prefix option feature, full details of which can be found in the publication IBM System/360 Operating System, PL/I Reference Manual.

In association with this, the programmer may specify his own exit (to be taken when a particular condition occurs) or may cause an interruption by means of the SIGNAL statement. In particular, attention should be paid to the CHECK condition, as this enables the programmer to maintain a close watch on any variables he wishes to nominate.

Should the programmer wish to exercise control of a more general nature, he may make use of the ERROR condition and, in his "ON-unit" further analyze the program by means of the ONCODE and ONLOC functions.

Standard system action for an ERROR condition raised in a non-tasking environment causes the FINISH condition to be raised.

ON-Codes

The ONCODE built-in function may be used by the programmer in any ON-unit to determine the nature of the error or condition which caused entry into that ON-unit.

An ON-unit, which has been established by the execution of an ON statement, is entered when the associated ON-condition is raised during execution of PL/I compiled code or of a PL/I library module. Thus, for example, a FIXEDOVERFLOW ON-unit would be entered whenever any of the conditions occur for which the language demands the raising of the FIXEDOVERFLOW condition.

Two ON-conditions, ERROR and FINISH, require special explanation. The ERROR condition is raised:

1. Upon execution of a SIGNAL ERROR statement
2. As a result of system action for those ON-conditions for which the language specifies system action to be "comment and raise the ERROR condition"
3. As a result of an error (for which there is no ON-condition) occurring during program execution

The FINISH condition is raised:

1. Upon execution of a SIGNAL FINISH, STOP, or EXIT statement
2. Upon normal completion of the MAIN procedure of a PL/I program
3. Upon completion of the action associated with the raising of the ERROR condition, except when a GO TO statement in the ON ERROR unit has resulted in transfer of control out of that unit

As a general rule, the value of the ON-code returned by the ONCODE function is that of the specific condition which caused entry into the ON-unit. Thus, in an ON CONVERSION unit, the programmer can expect an ON code corresponding to one of the

conversion conditions which cause the CONVERSION condition to be raised in PL/I. However, this is not necessarily true when executing an ON ERROR or an ON FINISH unit; the values are as follows:

1. When entered as a result of a SIGNAL ERROR or a SIGNAL FINISH, STOP or EXIT statement, or as a result of normal termination, the ON code values will be those of ERROR or FINISH respectively.

2. When entered for any other reason, the ON code value will be that associated with the error or condition which originally caused the ERROR condition to be raised.

Several separate but related occurrences may cause a particular PL/I ON-condition to be raised. For example, the TRANSMIT condition may be raised:

1. By execution of a SIGNAL TRANSMIT statement

2. By occurrence of an input TRANSMIT error

3. By occurrence of an output TRANSMIT error

Although it is often useful to know precisely what caused an ON-condition to be raised, at times it will be sufficient simply to know which ON-condition was raised. This will apply particularly if the ONCODE function is used in an ERROR ON-unit after system action has occurred for an ON-condition. The ON codes have therefore been grouped, each group containing the codes associated with a particular ON-condition.

From time to time it may become necessary or desirable to add new ON-codes into a group. Perhaps a group containing one ON-code only may be expanded. This fact must be remembered when the ONCODE function is used to determine if a particular PL/I ON-condition has been raised. It is important to test to see whether the ON-code is within the range specified, even if there is only one ON-code in the range; otherwise, when a new set of library modules is used, it may become necessary to recompile the program.

When a group contains only one ON-code value, it is impossible to test specifically for the signaled condition. With more than one ON-code in the group, the first in the group represents the signaled condition, provided the program was compiled using a compiler which was released at the same time as the library in which the expansion of the group first appeared.

The ON-codes and their associated conditions and errors are shown below. The groups and their ON-code ranges are shown in Figures 29 and 30. Language ON-conditions are shown in capitals, others in lower case letters.)

Code	Condition/Error
0	ONCODE function used out of context
3	Source program error
4	FINISH
9	ERROR
10	NAME
20	RECORD (signaled)
21	RECORD (record variable smaller than record size)
22	RECORD (record variable larger than record size)
23	RECORD (attempt to write zero length record)
24	RECORD (zero length record read)
40	TRANSMIT (signaled)
41	TRANSMIT (output)
42	TRANSMIT (input)
50	KEY (signaled)
51	KEY (keyed record not found)
52	KEY (attempt to add duplicate key)
53	KEY (key sequence error)
54	KEY (key conversion error)
55	KEY (key specification error)
56	KEY (keyed relative record/track outside data set limit)
57	KEY (no space available to add keyed record)
70	ENDFILE
80	UNDEFINEDFILE (signaled)
81	UNDEFINEDFILE (attribute conflict)
82	UNDEFINEDFILE (access method not supported)
83	UNDEFINEDFILE (blocksize not specified)
84	UNDEFINEDFILE (file cannot be opened, no DD card)
85	UNDEFINEDFILE (error initializing REGIONAL data set)
90	ENDPAGE
300	OVERFLOW
310	FIXEDOVERFLOW
320	ZERODIVIDE
330	UNDERFLOW
340	SIZE (normal)
341	SIZE (I/O)
350	STRINGRANGE
360	AREA raised in ALLOCATE statement
361	AREA raised in assignment statement
362	AREA signaled
500	CONDITION
510	CHECK (LABEL)
511	CHECK (VARIABLE)
520	SUBSCRIPTRANGE
600	CONVERSION (internal) (signaled)
601	CONVERSION (I/O)
602	CONVERSION (transmit)
603	CONVERSION (error in F-format input)
604	CONVERSION (error in F-format input) (I/O)

605 CONVERSION (error in F-format input) (transmit)
606 CONVERSION (error in E-format input)
607 CONVERSION (error in E-format input) (I/O)
608 CONVERSION (error in E-format input) (transmit)
609 CONVERSION (error in B-format input)
610 CONVERSION (error in B-format input) (I/O)
611 CONVERSION (error in B-format input) (transmit)
612 CONVERSION (character string to arithmetic)
613 CONVERSION (character string to arithmetic) (I/O)
614 CONVERSION (character string to arithmetic) (transmit)
615 CONVERSION (character string to bit string)
616 CONVERSION (character string to bit string) (I/O)
617 CONVERSION (character string to bit string) (transmit)
618 CONVERSION (character to picture)
619 CONVERSION (character to picture) (I/O)
620 CONVERSION (character to picture) (transmit)
621 CONVERSION (P-format input - decimal)
622 CONVERSION (P-format input - decimal) (I/O)
623 CONVERSION (P-format input - decimal) (transmit)
624 CONVERSION (P-format input - character)
625 CONVERSION (P-format input - character) (I/O)
626 CONVERSION (P-format input - character) (transmit)
627 CONVERSION (P-format input - sterling)
628 CONVERSION (P-format input - sterling) (I/O)
629 CONVERSION (P-format input - sterling) (transmit)
1000 Attempt to read output file
1001 Attempt to write input file
1002 GET/PUT string length error
1003 Unacceptable output transmission error
1004 Print option on non-print file
1005 Message length for DISPLAY statements zero
1006 Illegal array datum data-directed input
1007 REWRITE not immediately preceded by READ
1008 GET STRING -- unrecognizable data name
1009 Unsupported file operation
1010 File type not supported
1011 Inexplicable I/O error
1012 Outstanding READ for update exists
1013 No completed READ exists - incorrect NCP value
1014 Too many incomplete I/O operations
1015 EVENT variable already in use
1016 Implicit-OPEN failure - cannot proceed
1017 Attempt to REWRITE out of sequence
1018 ERROR condition raised if end-of-file is encountered before the delimiter while scanning list-directed or data-directed input, or if the field width in the format list of edit-directed input would take the scan beyond the end-of-file.
1019 Attempt to close file not opened in current task
1500 Short SQRT error
1501 Long SQRT error
1504 Short LOG error
1505 Long LOG error
1506 Short SIN error
1507 Long SIN error
1508 Short TAN error
1509 Long TAN error
1510 Short ARCTAN error
1511 Long ARCTAN error
1514 Short ARCTANH error
1515 Long ARCTANH error
1550 Invalid exponent in short float integer exponentiation
1551 Invalid exponent in long float integer exponentiation
1552 Invalid exponent in short float general exponentiation
1553 Invalid exponent in long float general exponentiation
1554 Invalid exponent in complex short float integer exponentiation
1555 Invalid exponent in complex long float integer exponentiation
1556 Invalid exponent in complex short float general exponentiation
1557 Invalid exponent in complex long float general exponentiation
1558 Invalid argument in short float complex ARCTAN or ARCTANH
1559 Invalid argument in long float complex ARCTAN or ARCTANH
2000 Unacceptable DELAY statement
2001 Unacceptable use of the TIME built-in function
3000 E-format conversion error
3001 F-format conversion error
3002 A-format conversion error
3003 B-format conversion error
3004 A-format input error
3005 B-format input error
3006 Picture character string error
3798 ONSOURCE or ONCHAR out of context
3799 Improper return from CONVERSION ON-unit
3800 Structure length ≥ 16**6 bytes
3801 Virtual origin of array ≥ 16**6 or ≤ -16**6
3900 Attempt to wait on inactive and incomplete event
3901 Task variable already active
3902 Event already being waited on
3903 Wait on more than 255 incomplete events

Range	Group
3-5	As for 1000-9999
10-199	I/O ON-conditions
300-399	Computational ON-conditions
500-549	Program check-out conditions
600-899	Conversion conditions
1000-9999	Error conditions (also 3-5)

Figure 29. Main ON-Code Groupings

Range	Group	Range	Group
0	ONCODE	510-519	CHECK
3	Source program error	520-529	SUBSCRIPTRANGE
4	FINISH		
9	ERROR	530-599	(Unallocated)
10-19	NAME	600-899	CONVERSION
20-39	RECORD	900-999	(Unallocated)
40-49	TRANSMIT	1000-1199	I/O errors
50-69	KEY	1200-1499	(Unallocated)
70-79	ENDFILE	1500-1699	Data processing errors
80-89	UNDEFINEDFILE	1700-1999	(Unallocated)
90-99	ENDPAGE	2000-2099	Unacceptable statement errors
100-299	(Unallocated)	2100-2999	(Unallocated)
300-309	OVERFLOW	3000-3499	Conversion errors
310-319	FIXEDOVERFLOW	3500-3799	(Unallocated)
320-329	ZERODIVIDE	3800-3899	Structure and array errors
330-339	UNDERFLOW	3900-3999	Tasking errors
340-349	SIZE	4000-8090	(Unallocated)
350-359	STRINGRANGE	8091-8199	Program interrupt errors
360-369	AREA	8200-8999	(Unallocated)
370-499	(Unallocated)	9000-9999	System errors
500-509	CONDITION		

Figure 30. Detailed ON-Code Groupings

3904 Active event variable as argument to COMPLETION pseudo-variable
3905 Invalid task variable as argument to PRIORITY pseudo-variable
3906 Event variable active in assignment statement
3907 Event variable already active
3908 Attempt to wait on an I/O event in wrong task
8091 Invalid operation
8092 Privileged operation
8093 EXECUTE statement executed
8094 Protection violation
8095 Addressing interruption
8096 Specification interruption
8097 Data interruption
9000 Too many active ON-units and entry parameter procedures
9002 Invalid free storage (main procedure)

Trace of Active Procedures

A trace of active procedures may be obtained by use of the SNAP option in an ON statement. However, this technique has the limitation that it records only procedures which are active at the time when the condition occurs, because of the use of dynamic storage; when the storage is released it is immediately available for some other use, and so cannot be used to maintain a full trace. If a full flow trace is required, then this should be programmed, either by means of the SIGNAL statement in association with an ON statement and ON-unit, or by specifying all procedure names in a CHECK list with the appropriate action in an ON-unit.

The format of the SNAP output is either of the following:

1. CONDITION xxxx OCCURRED AT OFFSET ± hhhhh FROM ENTRY POINT E1

2. CONDITION xxxx OCCURRED AT OFFSET ± hhhhh FROM ENTRY POINT OF xxxx ON-UNIT

followed by:

```
CALLED FROM PROCEDURE WITH ENTRY POINT
E2
CALLED FROM PROCEDURE WITH ENTRY POINT
E3
etc., etc.
```

If the statement number compiler option is specified, the SNAP output message will also contain IN STATEMENT nnnnn immediately following the word OCCURRED in the first line, or after the word CALLED in subsequent lines. The notation nnnnn gives the number of the statement in which the condition occurred.

The characters that replace xxxx are an abbreviated form of the name of the ON-condition which has occurred (the abbreviations are given in Figure 31). hhhhh is a hexadecimal offset; E1, E2, etc., are entry point names indicating the actual entry points used to enter the procedure in which the condition occurred, or from which the next named entry point was called.

If a condition occurs in an ON-unit, then the entry point name in the second line will be that of the procedure from which the ON-unit was entered, not necessarily the procedure in which the ON-unit is situated.

Condition	Abbreviation
OVERFLOW	OFL
SIZE	SIZE
FIXEDOVERFLOW	FOFL
SUBSCRIPTRANGE	SUBRG
CHECK	CHCK
CONDITION	COND
FINISH	FIN
ERROR	ERR
ZERODIVIDE	ZDIV
UNDERFLOW	UFL
STRINGRANGE	STRG
NAME	NAME
RECORD	REC
TRANSMIT	TMIT
KEY	KEY
ENDFILE	ENDF
UNDEFINEDFILE	UNDF
CONVERSION	CONV
ENDPAGE	ENDP

Figure 31. Abbreviations for ON-Conditions

If SNAP SYSTEM has been specified in a programmer's ON statement, the system action message described in the section called "Object Time Diagnostic Messages" will be printed, followed by the trace of active procedures:

```
CALLED FROM PROCEDURE WITH ENTRY POINT
E2
etc.
```

The one exception is the case of SNAP SYSTEM for the CHECK conditions. In these cases a standard SNAP message will be written, followed by the standard system action print-out for the CHECK condition.

Symbolic Output

The data-directed input/output features may be used instead of, or in addition to, the CHECK condition handling. The programmer may use data-directed output to obtain status information in terms of the symbols used in the source program. Refer to the publication IBM System/360 Operating System, PL/I: Refrence Manual, for a full description of this feature.

Communication with the Program

The DISPLAY statement provides a means of communicating with the program while it is being executed.

The two forms in which an operator message can appear on the typewriter are as follows:

Without the REPLY option, which gives the unaltered character string specified by the programmer.

With the REPLY option, which gives the character string specified by the programmer preceded by a two-digit code generated by the Operating System. The operator must use this code as a prefix to his reply message. The EVENT option may be used here.

User Requested Dump

An additional debugging feature is the ability to obtain a storage dump at any point in the program.

A dump is obtained by the statement:

```
CALL dump identifier [(argument)];
```

The dump identifier is one of the following

IHEDUMP or IHEDUMT - dump all core and terminate

IHEDUMJ or IHEDUMC - dump all core and continue processing

The dump will include information such as register values, load list, contents of PIE, and a storage dump. To help the user interpret the dumps, a directory is printed out at the start of each dump. This directory includes:

1. The contents of the SYSPRINT file buffers, if the file is open.

2. The name of the files currently open, with the addresses of the relevant control blocks.

3. The name of the current file.

4. The addresses of the save areas and of other areas of special interest.

The argument in the CALL Statement is optional; if it is used the dump identifier must be declared as ENTRY(FIXED BINARY). The argument is an expression that is evaluated at execution-time; the result is a fixed binary integer that appears in the heading of the dump. This integer must be in the range 0 to 127, a number outside this range is replaced by 127.

If no DD card is supplied, or if an unrecoverable error is detected during the output, e.g. incorrect chaining of save areas, then a standard operating system ABEND dump will occur and will terminate the job step.

For more information on storage dumps and the information they provide, see IBM System/360 Operating System: Messages, Completion Codes and Storage Dumps, Form C28-6631.

User Completion Codes for Abnormal Termination

PL/I programs can terminate abnormally in six different ways:

1. EXIT (abnormal termination of a task)

2. STOP (abnormal termination of the program)

3. If the ERROR condition is raised and there is neither an ERROR ON-unit nor a FINISH ON-unit with a GO TO statement.

4. CALL IHEDUMP or CALL IHEDUMT.

5. If an interrupt occurs during execution of the error handler routine. This results in execution of the ABEND macro.

6. An ABEND in OS/360 in the major task.

In cases (5) and (6) above, a full storage dump will be printed, provided that a SYSABEND DD card has been used. If no such card exists, and MFT or PCP is being used, an indicative dump will be printed.

If, in cases (1) to (5), a program terminates abnormally in a non-tasking program, a completion code is printed out. In the first four cases listed above, the completion code has a value which is the sum of a basic code plus the current value of the return code. (The return code can be set by the programmer or may have a default value of zero - see the chapter "Managing Programs".) The basic code values are:

EXIT and STOP	1000
ERROR	2000
CALL IHEDUMP	3000
Interrupt in error handler	4000

When a program terminates abnormally with a completion code of 4000, this means that disastrous error has occurred, such as a control block being overwritten.

When a program terminates abnormally with a completion code of 3333 this means that a disastrous error has occurred in the dump output modules.

When the operating system terminates a program, the user completion code is zero, and the system completion code is the operating system completion code.

There are two instances in which, since execution of the program has never commenced, no user completion code is issued. Instead, a special return code value is generated by the PL/I library and then passed to the return code register of the operating system in the usual way. These two instances, and the return codes generated, are:

Pseudo-register vector too long	4004
No MAIN procedure	4008

All the 4000-series codes mentioned above are accompanied by a message at the operator's console. (For further explanation of the underlying causes of these messages, refer to the explanations given for each message in "Object-Time Error Messages" in Appendix G of this publication.)

Return Codes

Return codes are set by use of the statement CALL IHESARC.

COMPILER INPUT

Source records must be in one of the two following formats:

Fixed-length records of logical record length up to 100 characters. Blocked records are acceptable. The maximum block size must be a multiple of the logical record length and must be one capable of being accommodated by the SIZE option.

However, blocked records are not accepted from paper tape. Fixed-length records on paper tape have a fixed number of characters after translation; the number of characters before translation is not fixed.

Undefined format records of length up to 100 characters.

The format, logical record length, and block size may be specified in the DCB operand in the DD statement, they may be taken from the data set label, or they may be implied from the occurrence of the source file in the input stream. If not specified, undefined format is assumed with a block size of 100.

In order to use the undefined format, the identity of SYSIN must be switched to the appropriate input device for the duration of the compilation job-step by means of a DD statement, either explicit or cataloged, for that job-step. The DD statement must specify U-type record format, and if the input is of paper tape origin, the appropriate translate mode must also be specified, depending on the nature of the tape code. The user must ensure that an end-of-record indication is given by the correct code on the paper tape, with the paper tape reader keys set correctly to interpret the code.

In both cases the source listing option (SOURCE) causes the complete input record to be printed.

The SORMGIN option specifies the area within the record which contains PL/I source text. The rest of the record may be used for identification purposes. If SORMGIN is not specified, the standard default values of 2 and 72 are assumed (see the section called "Compiler Processing" for a description of the SORMGIN option).

REPRESENTATION OF DATA

The representation of data in System/360 storage is described in the following paragraphs, together with the various permitted precisions and lengths of such data.

Coded Arithmetic Data

By virtue of declared attributes, the following eight types of coded arithmetic data forms may exist:

REAL FIXED DECIMAL precision (P,Q) data is represented in the packed-decimal format. The P digits occupy FLOOR ((P + 2)/2) bytes, aligned on any byte. When P is even, the effective precision is (P + 1,Q) for arithmetic operations other than division. An unwanted high-order digit may exist, therefore, and may remain undetected and be included in further operations. It can only be eliminated if SIZE is enabled. The maximum precision avaialble is 15 digits. Arithmetic operations are performed on P-digit integers, according to the scale factor Q. The scale factor is not directly associated in storage with the data, but is specified in a data element descriptor (DED) passed to the library for operations involving the data. The default precision is 5,0.

REAL FIXED BINARY precision (P,Q) data is represented in the fixed-point binary format. The P digits occupy four bytes, word aligned (half-word or double-word precision is not supported). The maximum precision is 31 digits. Arithmetic operations are performed as for real fixed decimal data; again, the scale factor is supplied to the library via a DED. The default precision is 15,0.

REAL FLOAT DECIMAL precision (P) data is represented in the hexadecimal floating-point format. Before and after arithmetic operations, the data is normalized in storage. The maximum available precision is 16 digits. If the specified precision is less than or equal to 6, the data occupies four bytes, word aligned (short floating-point form). If the specified precision is 7 or more, the data occupies eight bytes, double-word aligned (long floating-point form). The default precision is 6.

Declared attributes	Default attributes	Maximum precision	Default precision
Initial letter:			
A --> H, O --> Z	REAL,FLOAT,DECIMAL	-	6 decimal digits
I --> N	REAL,FIXED,BINARY	-	15 bits
BINARY	REAL, FLOAT	53 bits	21 bits
DECIMAL	REAL, FLOAT	16 decimal digits	6 decimal digits
FIXED	REAL, DECIMAL	15 decimal digits	5 decimal digits
FLOAT	REAL, DECIMAL	16 decimal digits	6 decimal digits
REAL[1]	FLOAT, DECIMAL	16 decimal digits	6 decimal digits
FIXED BINARY	REAL	31 bits	15 bits
FIXED DECIMAL	REAL	15 decimal digits	5 decimal digits
FLOAT BINARY	REAL	53 bits	21 bits
FLOAT DECIMAL	REAL	16 decimal digits	6 decimal digits
REAL[1] FIXED	DECIMAL	15 decimal digits	5 decimal digits
REAL[1] FLOAT	DECIMAL	16 decimal digits	6 decimal digits
REAL[1] BINARY	FLOAT	53 bits	21 bits
REAL[1] DECIMAL	FLOAT	16 decimal digits	6 decimal digits

Note 1: If COMPLEX is declared instead of REAL, the attributes are the same as for REAL and are applied to each of the two arguments.

Figure 32. Attributes and Precisions for Coded Arithmetic Data

REAL FLOAT BINARY precision (P) data is represented in the hexadecimal floating-point format. Before and after arithmetic operations the data is normalized in storage. The maximum available precision is 53 digits. If the specified precision is less than or equal to 21 the data occupies four bytes, word aligned (short floating-point form). If the specified precision is 22 or more, the data occupies eight bytes, double-word aligned (long floating-point form). The default precision is 21.

COMPLEX FIXED DECIMAL precision (P,Q) data is represented as for REAL FIXED DECIMAL. The real and imaginary parts occupy immediately adjacent fields, the real part first.

COMPLEX FIXED BINARY precision (P,Q) data is represented as for REAL FIXED BINARY. The real and imaginary parts occupy immediately adjacent full words, the real part first.

COMPLEX FLOAT DECIMAL precision (P) data is represented as for REAL FLOAT DECIMAL. The real and imaginary parts occupy immediately adjacent full or double words, depending upon the precision; the real part occupies the first field.

COMPLEX FLOAT BINARY precision (P) data is represented as for REAL FLOAT BINARY. The real and imaginary parts occupy immediately adjacent full or double words, depending upon the precision; the real part occupies the first field.

String Data

There are four types of string data:

1. Fixed-length CHARACTER
2. Fixed-length BIT
3. Variable-length CHARACTER
4. Variable-length BIT

Variable-length data has associated control areas known as "dope vectors" which describe the strings. A dope vector contains a record of the maximum length and the current length of the string, together with a pointer to the beginning of the string. Dope vectors need not be adjacent to the data they describe, but will normally occupy storage of the same storage class.

A string dope vector is created for all variable-length strings. In addition a dope vector is created for any fixed-length string which is an argument to either a procedure or a library routine. A variable-length string is addressed through its dope vector. A fixed-length string may be addressed directly or, if it has one, through its dope vector. Refer to Appendix

C of this publication for a detailed description of string dope vectors.

CHARACTER data is stored contiguously from the left end of a field long enough to accommodate the maximum length declared for the string. The leftmost byte has no special alignment: it is this byte which is addressed by a CHARACTER string's dope vector.

BIT data may be either UNALIGNED or ALIGNED. Both are stored eight bits per byte. Unalignment refers to the relative location of adjacent strings, not to the density of a single string. Aligned data fields occupy an integral number of bytes. Unaligned data fields occupy only as many bits as their maximum lengths require. BIT data is stored contiguously from the leftmost bit of its field, and is addressed at this leftmost bit.

Pictured Data

Data declared with a PICTURE attribute is stored in two fields. One field contains the picture information, the other the data element. The data is addressed independently of the picture information. Pictured data requires two addresses.

Data of either type, arithmetic or string, may be specified by a PICTURE attribute. A pictured arithmetic data item is termed a numeric field. There are five types of numeric fields, which are described as follows:

REAL PICTURE (fixed decimal picture) data is represented by bytes on a byte boundary. The precision of a fixed decimal numeric field may not exceed the limits for the corresponding coded arithmetic form, i.e., 15 digit positions.

REAL PICTURE (float decimal picture) data is represented by bytes on a byte boundary. The precision must not exceed the limits for the corresponding coded arithmetic form, i.e., 16 digit positions and an exponent of P digits where P is defined as for the exponent of an E format item.

REAL PICTURE (sterling picture) data is represented by bytes on a byte boundary. The precision is 3 + number of digits in the pound field + number of fractional digits in the pence field; this must not exceed the limits for the coded fixed decimal form, i.e., 15 digit positions.

COMPLEX PICTURE (fixed decimal picture) data - the real and imaginary parts are represented as two immediately adjacent byte fields, the real part first. Both parts are described by the single picture specification which is as described for a REAL fixed decimal numeric field.

COMPLEX PICTURE (floating decimal picture) data - the real and imaginary parts are represented as two immediately adjacent byte fields, the real part first. Both parts are described by the single picture specification which is as described for a REAL float decimal numeric field.

CHARACTER, but not BIT, string data may be specified by a PICTURE attribute. The data is represented by one byte per character on a byte boundary.

The picture field contains the declared picture specification without its surrounding quotation marks and with iteration of characters expanded.

Data Element Descriptor (DED)

The format of the data element descriptor (DED) (Figure 33) is as follows:

Flags: An eight-bit encoded form of declared information (see Figure 34).

The P byte is the declared or default precision of the data item. Maximum values are:

Binary Fixed:	31
Binary Float:	53
Decimal Fixed:	15
Decimal Float:	16

The Q byte is the declared or default scale factor of the data item in excess 128 notation (i.e., if the implied fractional point is between the last and next to last digit, Q will have the value 129). For numeric fields, Q is the resultant scale factor derived from the apparent precision as specified in the picture, i.e., the number of digit positions after a V picture item as modified by an F (scale factor) item.

The W byte specifies the number of bytes allocated for the numeric field.

The L byte specifies the number of bytes allocated for the picture associated with a numeric field; if the data item is string, L occupies two bytes; if arithmetic, one byte.

The picture specification field contains the picture declared for the data item; if the data item is string, the picture may

Data Type	Representation	DED Formats (in Bytes)					
		1	2	3	4	5	6.....
Arithmetic	Fixed-point Floating-point Packed-decimal	Flags	P	Q			
	Numeric Field	Flags	P	Q	W	L	Picture
String	Unpictured	Flags					
	Pictured	Flags	L		Picture		

Figure 33. Data Element Descriptor (DED)

	0	1	2	3	4	5	6	7	
0 =	String	0	Unaligned	Fixed	Pictured	Bit	0	0	= 0
		0	Aligned	Vary-ing	Un-pictured	Charac-ter	0	0	= 1
1 =	Arith-metic	0	Non-Sterling	Short	Numeric Field	Deci-mal	Fixed	Real	= 0
		0	Sterling	Long	Coded	Binary	Float	Complex	= 1

Figure 34. Eight-Bit Encoded Form of Declared Information in Flags

occupy 1 through 32,767 bytes; if arithmetic, 1 through 255 bytes. If the original picture specification contained iteration factors, it will have been fully expanded.

Pointer Data

A pointer variable is stored as four bytes aligned on a fullword boundary. The four bytes contain:

Byte 0: Zero

Bytes 1-3: A (Based variable):

For null values, the four bytes contain X'FF000000'.

Offset Data

An offset variable is stored in four bytes aligned on a fullword boundary. The data consists of byte offsets. The null value is X'FF000000'.

Label Data

Data of type LABEL takes on the values of statement labels. Label variables occupy a two-word field aligned to a fullword address. The field is used as follows:

```
Word 1:  Activation indicator
Word 2:  Bits 0-7    Gives nature of
                     activation indicator
         Bits 8-31   Address of statement
                     in object program
```

The data is addressed at the first fullword address.

Task Data

A task variable is stored in a 28-byte area aligned on a fullword boundary. The format is shown in Figure 35. On allocation, the compiler initializes the variable as follows:

Byte 0: Zero
Byte 8-11: A(Symbol table entry)

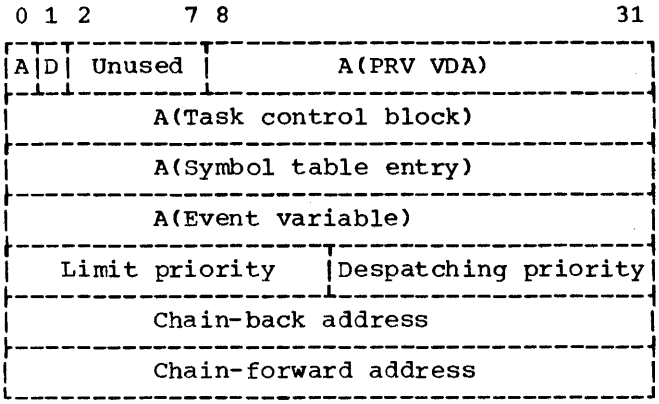


Figure 35. Format of the Task Variable

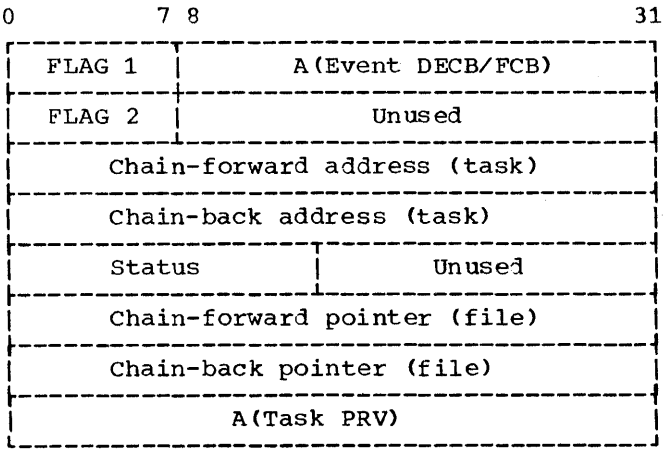
The flags set are:

A = 0 TASK variable inactive
= 1 TASK variable active
D = 0 CALL with TASK option
= 1 CALL without TASK option

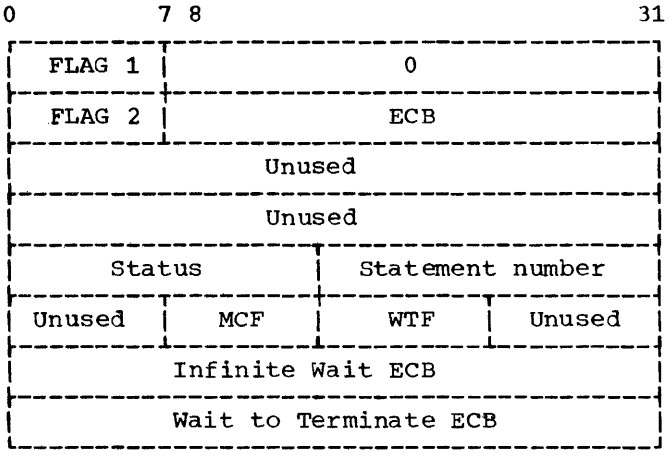
Event Data

An event variable is stored in a 32-byte area aligned on a fullword boundary. The formats are shown in Figures 36 and 36.1. On allocation, the compiler initializes the variable as follows:

Byte 0: Zero
Byte 4: Zero
Status: Zero
Flag : Zero



•Figure 36. Event Variable Used with I/O



•Figure 36.1. Event Variable Used with a Task

The flags set are:

Flag 1

I/O Event

1000 0000 Active EVENT variable
0100 0000 EVENT variable associated with I/O
0010 0000 No WAIT required
0001 0000 A(FCB) in first word of variable
0000 1000 EVENT variable to be checked
0000 0100 DISPLAY EVENT variable
0000 0010 IGNORE option with this event

Task EVENT

0000 0000 Multitasking non-I/O EVENT variable
1000 0000 Active EVENT variable
0010 0000 Normal PL/I termination
0001 0000 Abnormal PL/I termination
0000 0001 EVENT variable being waited on

Flag 2

I/O EVENT

1000 0000 EVENT being waited for
0100 0000 EVENT is complete

Area Data

An area variable is stored as 16 bytes plus the area length required; it is aligned on a doubleword boundary. The format is shown in Figure 37. The ADDR function applied to an area returns the address of the 16-byte area, not of the area itself. An area is described by an

area dope vector (see Appendix C, 'Object Program Organization and Conventions')

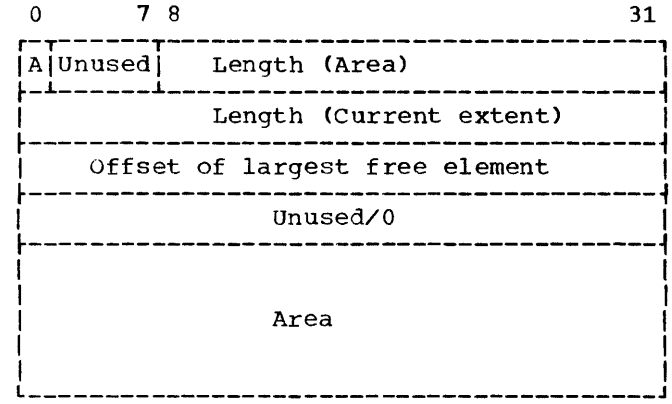


Figure 37. Format of the Area Variable

The flag set is:

A = 1 AREA variable contains free list.

If a free list exists, the fourth word is set to 0.

Arrays

Arrays of any data type described in the section on scalars are allowed. Arrays are stored in row-major order, in increasing storage addresses. Except for aligned bit strings, the elements of an array which is not in a structure are stored contiguously.

Each array may be described by a dope vector holding the dimension information declared for the array. The number of dimensions of the array is not included in its dope vector. The routine which refers to the dope vector must know the number of dimensions. External procedures know the number from a declaration of the parameter, and Library routines receive the number as an argument.

Arrays of scalars and fixed-length strings are described by a single level of dope vector.

Arrays of variable-length strings have a dope vector which points to an array of string dope vectors, which in turn give the current and maximum lengths of each element and point to the actual strings.

Structures

Structures require dope vectors whether they contain arrays or not.

A structure dope vector for a non-dimensioned structure is an ordered list of addresses of the elements of which the major structure is composed, including the elements of minor structures contained within the major structure. Minor structures do not have separate dope vectors. The elements of a minor structure are a contiguous subset of the elements of its major structure, and can therefore be addressed from the major structure's dope vector. If an element is an array, the address in the structure dope vector is replaced by the array dope vector. An array of variable-length strings in a structure would therefore be addressed as follows:

The structure dope vector has an entry which is an array dope vector

This array dope vector points to an array of string dope vectors

Each string dope vector then points to a string

If a structure has a dimension attribute, it specifies an array of structures. However, the dope vector does not point through an array dope vector to an array of structure dope vectors. Instead, the dimensionality is applied to the elements of the structure.

The Creation of Dope Vectors

The PL/I (F) compiler produces dope vectors in the following five circumstances:

1. When arguments are passed to a procedure (whether internal or external).
2. When lengths of strings or dimensions of arrays are variable, or greater than 256 bytes (or 2048 bits) in length.
3. When a variable is DEFINED on another.
4. When minor structures are passed as arguments (a subset of the major structure will not suffice in all cases).
5. When asterisk notation is used, creation of separate dope vectors is required for projections of arrays.

The following implementation conventions and restrictions apply to the operating system/360 PL/I (F) implementation.

INPUT/OUTPUT CONVENTIONS AND RESTRICTIONS

PL/I and Data Sets

A PL/I program does not nominate data sets directly. Instead, under the System/360 operating system, it associates a file with a job control DD statement that identifies a data set. The file is associated with the DD statement by means of:

1. The TITLE option: This specifies the name (up to eight characters long) of the DD statement

2. The file name: If there is no TITLE option, the file name (padded or truncated to eight characters) is used as the ddname.

Associating the file with the data set requires the merging of information describing each of them and ensuring that there is no conflict of information on any particular item. At execution time the compiler calls the PL/I library I/O modules to handle the source program information. The I/O modules generate a skeleton DCB and insert in it, first, the file attribute from the DECLARE and OPEN statements, and then, any default file attributes required.

The library modules then issue an OPEN macro, which fills in the remaining DCB fields with information from, first, the DD statement for the data set and, then, the data set label. Neither the DD statement nor the data set label can override information already provided by the source program; the data set label cannot override information provided by the DD statement.

If there are any DCB fields still not set, default information is provided by the PL/I library OPEN module.

SYSPRINT is normally used as a STREAM OUTPUT PRINT file; it can be used as a RECORD file if declared INTERNAL.

DISPLAY

The maximum lengths of character string acceptable are 100 characters for the message and 72 characters for the reply. The reply string's current length is set equal to its maximum length and padded with blanks if necessary.

PAGESIZE

The maximum size of a page is 32,767 lines; the minimum is 1 line. If the page size is not specified a value of 60 lines is assumed.

LINESIZE

The maximum and minimum line size depend on the record format.

Record Format	LINESIZE	
	Min.	Max.
V	Non-PRINT file: 10 PRINT file: 9	32,751
U,F	1	32,759

If a line size is not specified, the default values are:

PRINT file - 120 characters

Non-PRINT output file - no default value

The LINESIZE value determines the logical-record length in the data set (i.e. the value of LRECL):

F- and U-format records: LRECL = LINESIZE

V-format records: LRECL = LINESIZE + 4

For PRINT files, an extra byte (for the ASA control character) is added to each of the above LRECL values.

If BLKSIZE is specified, its value and the LRECL value must be compatible. If BLKSIZE is not specified, its value is calculated from the LINESIZE value.

LINESIZE, SKIP and COLUMN in Non-PRINT Files

LINESIZE (expression): Evaluation of the expression gives an integer that must be within the limits described above. For V and U format records, LINESIZE is the maximum size of a line. If a variable LINESIZE is required, the maximum value must be specified as the BLKSIZE in the DD statement or in the ENVIRONMENT attribute. Short lines are padded with blanks for F format records only.

SKIP(expression): For output files, SKIP action depends on the record format:

F-format On a short line, SKIP fills out the remainder of the line with blanks.

V-format SKIP puts out the current line as a short record. If the byte count of the line is less than 14 (18 with control bytes), the line will be blanked up to that size. Successive lines will be of the same minimum length, padded with blanks.

U-format SKIP will put out the current line as a short record.

COLUMN (expression): For input files, if the value of the expression is greater than the current record length, a value of 1 is assumed.

For both input and output files, if the value is less than the current position on the line, the file is positioned at COLUMN (value) on the next line.

For output files, all characters from the current position in the line to the next position are blanked out. For U- or V-format records, if another record is required, a short record is put out subject to the rules described under the SKIP option.

Block Size and Record Size

The maximum size of a block or a record must not exceed 32,760 bytes.

See the section "Structure Mapping" in Appendix C for details of data aggregate size requirements necessary in calculating the record and block size for data sets using RECORD I/O facilities.

Data-Directed Input/Output

The maximum length of a qualified name, including the separating periods, is 255 characters.

The semicolon, which is not in the 48-character set, must always be represented in the input stream by the proper 11-8-6 punch (i.e., the character sequence ,. is not recognized as a semicolon).

Edit-Directed Input/Output

When using the E-format output, E(w,d,s), s must be less than 17 digits. When using E(w,d), d must be less than 16 digits.

If the number of significant digits in E-format data is greater than 16, then:

E-format input: CONVERSION condition raised
E-format output: Data is truncated

Character Code

Input to the object program is assumed to be in EBCDIC mode.

48-Character Set

48-character set "reserved" words (e.g., GT, LE, CAT, etc.) words must be preceded and followed by a blank or a comment. If they are not, the interpretation by the compiler is undefined and may not, therefore, be what the user intended.

A record containing part or all of a 48-character set reserved word must be 3 characters or more in length.

The ENVIRONMENT Attribute

The ENVIRONMENT attribute may contain the subfields given below. Only one option from each group is permitted.

General format of the ENVIRONMENT attribute is:

```
ENVIRONMENT([CONSECUTIVE|INDEXED|
    REGIONAL(1)|REGIONAL(2)|REGIONAL(3)]
    [{V|VS|VBS|F|U} (Blocksize
    [,Recordsize ])] [LEAVE|REWIND]
    [BUFFERS(n)] [COBOL] [CTLASA|CTL360]
    [INDEXAREA[(m)]] [NOWRITE] [GENKEY])
```

Data Set Organization: There are five types of data set organization. These are detailed below.

CONSECUTIVE: A data set consisting of unkeyed records, which are accessed in a physically sequential order. This organization is assumed if none is specified.

INDEXED: A data set which consists of keyed records, located by means of several levels of index.

REGIONAL(1): A data set which consists of keyed records, without recorded keys, which are located by means of relative record positions within the data set.

REGIONAL(2): A data set which consists of keyed records, with recorded keys, which are located by means of relative record positions, and by a search for a recorded key to match the given key.

REGIONAL(3): Identical to REGIONAL(2), except that positioning is in terms of relative tracks.

Record Format: Logical records may be in one of three formats: fixed length (format F), variable length (format V), or undefined (format U). Information regarding record format may be supplemented by the DD statement DCB subparameters, BLKSIZE and LRECL. The block size and record size must be specified in bytes.

F(blocksize[,recordsize]) specifies fixed length records with the block size as stated. The record size may be specified optionally, which indicates that records are blocked (i.e., that each physical record contains more than one logical record). In this case, the block size must be a simple multiple of the record size. If the record size is not specified, the records will not be blocked. Blocked records are not supported for UNBUFFERED files.

V(max-blocksize[,max-recordsize]) specifies that records are of varying length. A number of complete logical records are grouped together to form a physical record in such a way that each physical record does not exceed the maximum block size stated. If the maximum record size is specified, no logical record may exceed this size. Four bytes of control information per record, plus four bytes per block, are contained in V-format records. These items must be considered when specifying the block size and record size subfields. Blocked records are not supported for UNBUFFERED files.

VS(max-blocksize[max-recordsize]) specifies that spanned records are to be processed. The maximum block size can be exceeded by any logical record, in which case the record is segmented and written on two or more consecutive blocks. One block is written for each complete record, or segment, if spanning takes place.

VBS(max-blocksize[max-recordsize]) specifies that spanned records are to be processed. The maximum block size can be exceeded by any logical record, in which case the record is segmented and written on two or more consecutive blocks. Each block is written to within four bytes of the maximum length specified. A block can contain up to two spanned record segments or any number of complete records, or a mixture of both.

U(max-blocksize) specifies blocks of varying length up to the maximum specified.

Notes:

1. All record formats permit the use of a "printer control character." This character appears as the initial character of each record (except for V format, when it appears after the four control bytes).

2. The specification of a single parameter (blocksize) in the record format in the ENVIRONMENT attribute is taken to imply unblocked records. For F- and V-format records record size is then derived from the block size parameter and therefore overrides information supplied on the DD statement.

Data Set Positioning: The LEAVE and REWIND options are used for positioning a magnetic-tape volume when a data set is closed or when a volume switch is required in a multi-volume data set. The REWIND option allows the DISP parameter in the data set DD statement to control the action taken. Figure 37.1 shows the result when these options are specified.

Buffer Allocation: BUFFERS(n) specifies the number of buffers to be allocated for the data set; this number must not exceed 255. For BUFFERED files, if the option is omitted or the number specified is zero, two buffers are automatically allocated upon opening the data set. The buffer

Option		Action at Closing of Data Set	Action at End of Volume
LEAVE		Volume wound on to end of data set (If the file is BACKWARDS, the volume is rewound to the beginning of the data set) Channel: busy	No repositioning Channel: available
REWIND	PASS	Volume is wound on to end of data set (If the file is BACKWARDS, the volume is rewound to the beginning of the data set) Channel: busy	No repositioning Channel: available
	KEEP, CATLG, UNCATLG	Volume is rewound and unloaded Channel: available	
	DELETE	Volume is rewound Channel: available	
Both LEAVE and REWIND		REWIND is ignored	
Neither LEAVE nor REWIND		Volume is repositioned to the beginning of the current data set on that volume. (If the file is BACKWARDS, the volume is wound to the end of the data set.) Channel: busy	

Figure 37.1. Effect of LEAVE and REWIND Options on Repositioning of Magnetic-Tape Volumes

count may be supplied by a DD statement DCB subparameter BUFNO, if not specified in the ENVIRONMENT attribute.

An additional buffer is obtained for STREAM output data sets using U-format records.

COBOL Option: This specifies that files with this attribute will contain structures mapped according to the COBOL algorithm. This type of file may be used only for READ INTO, WRITE FROM, and REWRITE FROM statements.

COBOL data types and the equivalent PL/I data types are shown in Figure 38.

Files with the COBOL option may not be passed as arguments.

READ INTO, WRITE FROM and REWRITE FROM statements specifying COBOL files may have the EVENT option only if the compiler is able to determine that the PL/I and COBOL structure mappings are identical (i.e., all elementary items have identical boundaries). If they are not identical, or if the compiler cannot detect that they are identical, then an intermediate variable is created to represent the level 1 item mapped using the COBOL algorithm. The PL/I variable is assigned to this variable before the WRITE FROM is executed, or assigned from it after the READ INTO is executed. Thus the I/O statement containing the EVENT is not the one that completes the I/O operation; in these cases, the EVENT is ignored.

If an ON condition arises during a READ INTO, then

1. the INTO variable may not be used in the on-unit,

2. if the completed INTO variable is required, there must be a normal return from the on-unit.

Printer/Punch Control Characters: Two options are available, for RECORD CONSECUTIVE OUTPUT files only: CTLASA and CTL360. They have the following meaning:-

CTLASA - Requires the implementation to set the ASA control bit in the DCB subparameter DCBRECFM.

CTL360 - Requires the implementation to set the machine code control bit in the DCB subparameter DCBRECFM.

COBOL data type	PL/I data type
DISPLAY	PICTURE with A and/or X picture characters CHARACTER
COMPUTATIONAL Decimal length (=No. of 9s in picture) is	
1 to 4	No equivalent
5 to 9	FIXED BINARY (integers only)
10 to 18	No equivalent
COMPUTATIONAL-1	FLOAT (n) BINARY (for n ≤ 21) FLOAT (n) DECIMAL (for n ≤ 6)
COMPUTATIONAL-2	FLOAT (n) BINARY (for n > 21) FLOAT (n) DECIMAL (for n > 6)
COMPUTATIONAL-3	FIXED DECIMAL (precision and scale as in COBOL picture)

Figure 38. Equivalence of COBOL and PL/I Data

These options are used in order that spacing, skipping, etc., may be achieved in RECORD I/O files. It is the user's responsibility to ensure that the first byte of each record contains a valid ASA or machine code control character. These options will be ignored for STREAM files.

Performance with DIRECT INDEXED (Input or Update) Files: Performance is improved if INDEXED DIRECT files (which use BISAM) are specified with the INDEXAREA and NOWRITE options.

INDEXAREA [(m)]. Used with INDEXED DIRECT files opened for INPUT or UPDATE (and ignored by other files). This option causes the highest level of index to be loaded into main storage. The permitted formats are:

INDEXAREA (m): the parameter m is a decimal constant ≤ 32767. If the index size ≤ m, the index is loaded into main storage; if the index size > m, the index is not loaded. This enables the programmer to place a limit on the amount of core he is prepared to allocate for an index area. If m > 32767, the parameter is ignored and the index (whatever its size) is loaded into main storage.

INDEXAREA: if no parameter is specified the index (whatever its size) is loaded into main storage.

NOWRITE. Used for INDEXED DIRECT files opened for UPDATE (and ignored by other files). This option indicates to data management that there are no records to be added to the file and therefore the write-add facility is not required. If an attempt is made to execute a WRITE statement on a file with this option, a diagnostic message will be printed and the ERROR condition raised.

GENKEY Option

The GENKEY option applies only to INDEXED data sets. It enables the programmer to classify keys used in the data set into generic classes and to use a SEQUENTIAL KEYED INPUT or SEQUENTIAL KEYED UPDATE file to access and read records according to the classification of their keys.

A generic key is a character string that identifies a class of keys. All keys which begin with such a string are members of that class of keys. For example, the recorded keys 'ABCD', 'ABCE', and 'ABDF' are all members of the classes identified by the generic keys 'A' and 'AB'. The first two keys are also members of the class 'ABC', and the three keys can be considered as unique members of the classes 'ABCD', ABCE', 'ABDF', respectively.

The GENKEY option allows the programmer to start sequential reading or updating of an INDEXED data set from the first non-dummy record that has a key in a particular class. The class is identified by the inclusion of its generic key in the KEY option of a READ statement. Subsequent records can be read by READ statements without the KEY option, or they can be updated by REWRITE statements. It is the responsibility of the programmer to check that the last record in a generic class has been processed, as no indication is given when it is reached.

In the following example, a key length of more than three bytes is assumed:

```
DCL IND FILE RECORD SEQUENTIAL
    UPDATE KEYED
    ENV(INDEXED GENKEY);
      .
      .
      .
READ FILE(IND) INTO(INFIELD)
    KEY('ABC');
      .
      .
```

```
          .
NEXT: READ FILE(IND) INTO(INFIELD);
          .
          .
          .
      GO TO NEXT;
```

In the above example, the first READ statement causes the first non-dummy record in the data set whose key begins with 'ABC' to be read into INFIELD. Each time that the second READ statement is executed, the next non-dummy record will be retrieved.

If the data set contains no non-dummy records with keys of the specified generic class, the KEY condition is raised, and, on return from the on-unit, the next READ statement will read the first record in the data set. However, if the data set contains only dummy records, the ENDFILE condition is raised.

Note how the use of the GENKEY option affects execution of a READ statement that supplies a source key shorter than the key length specified in the KEYLEN subparameter of the DD statement which defines the data set. If the GENKEY option is used, it causes the source key to be interpreted as a generic key, and the record to be read is the first non-dummy record in the data set whose key begins with this source key. If the GENKEY option is not used, a short source key is padded on the right with blanks to form a source key of the specified key length, and the record to be read is the record which has this padded key, if such a record exists.

The use of the GENKEY option does not affect the result of supplying a source key whose length is greater than or equal to the specified key length. The source key, truncated on the right if necessary, identifies a specific record whose key can be considered the only member of its generic class.

EVENT Option

The EVENT option is implemented for RECORD input/output statements used as follows:

Access	File Organization
SEQUENTIAL	CONSECUTIVE UNBUFFERED REGIONAL UNBUFFERED
DIRECT	CONSECUTIVE INDEXED or REGIONAL

Note: The EVENT option should not be used on a WRITE statement if V or U format records are being added to a REGIONAL(3) data set which is being accessed in a direct update mode.

WAIT Statement

If the user wishes to specify more than one event name in a WAIT statement, the multiple-wait option must have been specified at SYSGEN time.

If a WAIT statement is executed and the events required to satisfy the WAIT contain a mixture of I/O and non-I/O events all non-I/O events will be set complete before any of the I/O events.

TITLE Option

If the TITLE option specified exceeds eight characters, then the first eight are used.

BASED Variables

The implementation of offsets and pointers does not support bit addressing. This restriction has no practical effect on ALIGNED bit strings. With UNALIGNED bit strings belonging to arrays or structures, however, only offsets or pointers to major structures or minor structures with byte (or higher) alignment should be used. Other restrictions on the use of based variables are given in Appendix H.

Initializing LABEL Variables in Structures with the LIKE Attribute

Initialization of LABEL variables in these structures requires careful handling particularly as the implementation does not provide the result specified by the language. A structure A is declared, using the LIKE attribute, to be identical to a structure B. Structure B contains a LABEL variable that is initialized, using the INITIAL attribute, to the value of a LABEL constant. The initial value of the corresponding LABEL variable in A is the initial value of the LABEL constant known in the block containing the declaration of B, not A.

For example:

```
DCL 1 B,
      2 L LABEL INITIAL (L1);
    .
    .
    .
L1: . ;      /*B.L = L1*/
    .
    .
    .
BEGIN;
DCL A LIKE B;
    .
L1: . ;      /*A.L IS GIVEN THE VALUE OF
               L1 IN STRUCTURE B*/
    .
    .
    .
END;
```

COMPILE-TIME PROCESSING CONVENTIONS AND RESTRICTIONS

The MACRO Option

The MACRO option should be included among the complete set of options for the compiler invocation if the program contains compile-time statements.

Precision

The precision, N, for a compile-time variable declared FIXED is 5.

INCLUDE Conventions

Included text must be a member of a partitioned data set. If only a single identifier is specified, i.e., of either of the following forms

```
(identifier)
identifier,
```

then it is assumed to be the name of a member of the data set with the ddname SYSLIB. If $identifier_1$ ($identifier_2$) is written then $identifier_1$ is the ddname and $identifier_2$ is the member name. DD cards must be provided for those data sets used. Records in these data sets must have a fixed length of not more than 100 characters. The maximum blocking factor is 5. The source margin and character set options on the EXEC control card also apply to included text.

Compile-Time Procedures

There may be no more than 254 compile-time procedures per compilation. Further, each procedure is limited to a maximum of 15 parameters.

Compile-Time DECLARE

No more than three levels of factoring are permitted in a compile-time DECLARE statement.

Combined Level of Nesting and Depth of Replacement

At any point in the program the combined level of nesting and depth of replacement is restricted to 50 levels. However, since not all nested or replacement items require the same amount of space, a program may run with a greater actual nesting or replacement depth than 50 levels. Depth of replacement is self-defining, but nesting level requires some clarification. A nesting level is required for:

each pair of parentheses, either explicit or implied by hierarchy of operation

each IF, DO, or PROCEDURE statement

each member of a parenthesized list, such as factor lists in DECLARE statements or argument lists of procedures.

Limitations on Size of Compile-Time Processor Input

The user's program is maintained internally as blocks of text. Blocksize is assigned at the start of processing and is a function of machine size as specified by the SIZE option on the EXEC card. The total size of internal text is restricted to 90 times the size of a text block. The minimum system configuration results in a blocksize of 1K, so a total of 90K is allowed for internal text. This minimum figure is roughly equivalent to 1000 source input statements.

Limitations on Number of Compile-Time Variables

The maximum number of compile-time variables which can be used in a program depends on the total size of the dictionary, which is restricted to 65,000 bytes. Assuming an average dictionary entry size of 28 bytes, this restricts the processor to approximately 2,300 items. An entry is made in the dictionary for each macro variable, macro procedure name, INCLUDE identifier, macro label, and unique compile-time constant. In addition, two dictionary entries are created for each iterative DO, one for each THEN or ELSE clause, and one for each compile-time procedure. Error message references are also entered into the dictionary. The dictionary is cleared at the end of compile-time processing; it is therefore unnecessary to keep the above considerations in mind if estimating available dictionary space during actual program compilation.

Output Line Numbering

Where constants or comments span more than one line, the output line numbering refers to the first input line number of the string or comment.

OTHER COMPILER CONVENTIONS AND RESTRICTIONS

OPTIONS Attribute

The list in the OPTIONS attribute may include the options MAIN, TASK and REENTRANT. The MAIN option should be used for the external procedure which is required to be given initial control at object time. The REENTRANT option must be specified if the object program generated by the compiler is to be reenterable.

The TASK option must be used if the procedure is to be invoked for tasking or to be invoked with other procedures with the TASK option.

Parameter to the MAIN Procedure

A single parameter may be passed by the EXEC statement for the execution job step to the MAIN procedure. If this facility is used, the first parameter to the MAIN procedure should be declared as a VARYING character string; the maximum length is 100, and the current length is set equal to the parameter length at object time. The parameter can also be a fixed-length character string.

Number of Variables

The maximum number of variables in the source program depends on the total size of the dictionary, which (for NOEXTDIC) is restricted to approximately 65,000 bytes. This is equivalent to a restriction of roughly 1,200 variables for a scientific user and to 1,000 for a commercial user. In computing these figures a reasonable allowance has been made for constants, statement labels, and other items which may require dictionary entries.

If the EXTDIC option is specified, the maximum size of the dictionary is approximately 1.5 times 65,000 bytes for a block size of 1K, and approximately 3.5 times 65,000 bytes for other block sizes.

The figures for variables are necessarily approximate, since the size of a dictionary entry varies with the type of variable, length of identifier, whether it is a structure element, and so on.

Number of Executable Statements

The total size of the internal text, at any point in the compilation, is restricted to 90 times the size of a text block. The size of a text block is itself dependent on the amount of core storage available to the compiler, as specified by the SIZE option The minimum block size is 1,024 bytes (1K), giving a maximum size for the internal text of 92,260 (90K). This is equivalent to roughly 280 executable statements.

The maximum block size is 16,384 bytes, giving a maximum of 1,474,560 bytes for the size of internal text. This is equivalent to roughly 14,000 executable statements.

The figures given for numbers of statements are necessarily approximate, since the number of bytes per statement will vary between different types of source programs.

Size of an Individual Statement

All statements, other than a DECLARE statement, are limited to 3,500 source

characters, i.e., equivalent to 50 cards. The 'content' of any statement, other than a DECLARE statement, is limited by the size of a text block; this varies, as described in the preceding paragraphs, with the storage available, but will not be less than 1,024 bytes.

The content of a statement can be calculated by ignoring nonsignificant blanks and comments, expanding iteration factors in string constants and pictures, and then adding one byte for each occurrence of an identifier, and three bytes for each occurrence of a constant. To this, for binary constants add the iterations of any CHARACTER or BIT strings (note that at this point BIT strings are treated as characters, not bits), since the (F) compiler expands the strings as if the programmer had written them in full, and two decimal digits for decimal constants. At most, these restrictions will limit a statement to six cards, but the limit will normally be between 20 and 30 cards, even for a text block of 1,024 bytes.

These restrictions also apply to a DECLARE statement for the text between any two commas which are not contained within parentheses.

Factoring of Attributes

The number of left parentheses used for factoring attributes in DECLARE statements is limited to 73 in a compilation.

Limitations on Nesting

There must not be more than 50 levels of nesting at any point in the compilation. The degree of nesting at any point is the number of PROCEDURE, BEGIN, or DO statements without a corresponding END statement, plus the number of currently active IF compound statements, plus the number of currently unmatched left parentheses, plus the number of dimensions in each active array expression, plus the maximum number of dimensions in each active structure expression.

The number of nested iteration factors in a format list must not exceed 20. The maximum nesting of ENTRY attributes within an ENTRY or GENERIC attribute is 3.

The GENERIC Attribute

There is a limitation on the number of family members and arguments which may be associated with a GENERIC entry name. The value given by evaluating the following formula must not exceed 700:

$$3n + 8\sum_{1}^{n} a_i + 8MAX(a_1, a_2, \ldots, a_n) + 3d$$

where n = the number of family members
a_i = the number of arguments relating to the ith family member
d = the greatest function nesting depth at which an invocation of the GENERIC entry name appears

Number of Blocks in a Compilation

The number of PROCEDURE, BEGIN, and iterative DO groups, plus the number of ON statements, must not exceed 255.

Level Numbers

The maximum declared level number permitted in a structure is 255. The maximum true level number permitted in a structure is 63.

Number of Parameters

The maximum number of parameters permitted at any entry point is 64.

Number of Dimensions

The maximum number of dimensions permitted, including dimensions inherited from containing structures, is 32.

Array Bounds

Arrays are limited, for each dimension, to a lower bound of -32,768 and to an upper bound of 32,767.

Data-Directed List

The maximum number of elements permitted in a list for data-directed input is 320. Each base element of a structure counts as a separate list element.

Structure and Array Expressions

The level of nesting in structure and array expressions is limited by the following rule:

For each level of nesting of structure or array expressions, add 2 for the maximum number of dimensions in the structure or array, add 2 for the maximum level in a structure expression, add 3 for each subscript or argument list in the expression or assignment, and finally, add 15.

The total for the whole nest should not exceed 900.

Constants

The precision or length of constants may not be greater than the precision or length of the corresponding type of variable.

Sterling Constants

The maximum number of digits allowed in the pounds field of a sterling constant is 13.

The number of digits following the decimal point in the pence field must not exceed 13 minus the number of digits in the pounds field.

String Constants

The number of characters in a string constant, after expansion of iteration factors, may not exceed the size of a dictionary block minus 14. The size of a dictionary block will vary with the storage available to the compiler in the same way as does text block size, but will not be less than 1,024 bytes.

Floating-Point Constants and E Format Items

The exponents of floating-point numbers are restricted to a maximum of 2 digits for decimal or 3 digits for binary.

Constants Returned by Procedures

If a procedure has more than one entry point, and each entry point returns a value, code is generated to convert each value returned to each of the data types for the entry points. If any of these values is a constant, it is possible that this constant cannot be converted to all the data types specified. A severe error message will be put out, and execution will be unsuccessful.

This situation can be avoided by assigning the constant to a variable of the same data type, and then returning this variable. For example:

```
DCL A ENTRY RETURNS(CHAR(8)),
    B ENTRY RETURNS(FIXED DECIMAL(15)),
    C ENTRY RETURNS(BIT(64)),
    ATEMP CHAR(8);
    .
    .
    .
 A: ENTRY CHAR(8);
    ATEMP = 'A08';
    RETURN(ATEMP);
 B: ENTRY FIXED DECIMAL(15);
    RETURN(108);
 C: ENTRY BIT(64);
    RETURN('10101'B);
```

The use of ATEMP avoids the interrupt caused by the CHARACTER->FIXED DECIMAL and the CHARACTER->BIT conversions. However, execution may still be unsuccessful, and a warning message is put out to remind the user.

Compiler-Generated Names

The number of names generated by the compiler must not exceed 11,264 in a compilation. One name is generated for each PROCEDURE, BEGIN, or ON block, for each variable declared as CONTROLLED INTERNAL, and for each INTERNAL file.

Temporary Results in Expression Evaluation

The maximum number of temporary results which may exist during the evaluation of an expression or during an assignment statement is 200.

An estimate of the number of temporary results which may exist during the evaluation of an expression can be obtained from the following:

At each level of parenthesis, count one for each operator which is forced to be evaluated before an inner level of parentheses. For each such operator, count one for each operand which requires conversion before use, count one for each nested function, count one for each subscripted variable used as a target in an assignment statement, and finally, count one for each pseudo-variable and each argument of a pseudo-variable.

Multiple Assignments and Pseudo-Variables

Multiple assignments are limited by the following rule:

> Count 11 for each target of a multiple assignment, add 3 for each pseudo-variable, and then add 11 for each argument of a pseudo-variable. The total must not exceed 4,085.

Function Values

The maximum number of different data types or precisions returned by one function may not exceed 256.

Qualified Names

The number of characters in a qualified name, which is to be used either for data-directed input/output or in CHECK lists, must not exceed 256.

Note that if the DATA option without a list is used for data-directed input, this will include all structure elements in the compilation.

String Lengths

The length, in characters or bits, of a string variable or intermediate string result is limited to 32,767.

String Lengths in Intermediate Result Fields

When non-adjustable VARYING strings, or functions which return non-adjustable VARYING strings, are used in an expression, the lengths of the intermediate result fields are calculated from the maximum lengths of the operands. If these lengths are at or near the maximum permitted by the implementation (32767 bytes or bits), the length of the intermediate fields may be greater than the implementation maximum; if so, they will be truncated on the left. This situation can occur with concatenation, the UNSPEC function with a character-string argument, the REPEAT function, and the STRING function.

The use of adjustable VARYING strings can create a similar problem. When an operand of the concatenate operator or the argument of the UNSPEC function is an adjustable VARYING string, the length of the intermediate result field is not tested, and execution will fail. This situation can also occur with SUBSTR if the third argument is not a constant, because in this case the result is an adjustable VARYING string.

Similarly, when a VARYING string is passed as an argument to a fixed-length string parameter, the length of the temporary argument created is the maximum length. If the user wishes to pass the current length of the VARYING string (in, for example, Y=X(A)), a possible method is:

```
DCL ATEMP CHAR(*) CTL;
    ALLOCATE ATEMP CHAR(LENGTH(A));
    ATEMP=A;
    Y=X(ATEMP);
    FREE ATEMP;
```

AREA Sizes

The size of an area is limited to 32767 bytes. In this implementation, the AREA size is provided by the value associated with the AREA attribute or by the default value of 1000 bytes.

LABEL Attribute

The number of statement-label constants specified by the LABEL attribute is limited to 125 in any particular label list.

POSITION

The maximum value of the integer constant in the POSITION attribute is 32,767.

PICTURE

The maximum length of a PICTURE describing a numeric field, after expansion of iteration factors, is 255.

The maximum length of a PICTURE describing a character string, after expansion of iteration factors, is the size of a dictionary block, less 14. The size of a dictionary block will vary with the storage available to the compiler in the same way as does text block size, but will not be less than 1,024 bytes (or 768 bytes if the EXTDIC option is in use).

SETS List

The total of twice the number of identifiers in a SETS list, plus the number of parameter numbers in a SETS list, must not exceed 255.

Scale Factor

The scale factor of a variable, or of an intermediate result of type FIXED, must be in the range -128 and +127.

Precision

The maximum precision of a variable or of an intermediate result is:

```
53  for  FLOAT BINARY
16  for  FLOAT DECIMAL
31  for  FIXED BINARY
15  for  FIXED DECIMAL
```

Floating-Point Magnitude

The magnitude of a normalized floating-point variable or intermediate result must lie in the range from 2.4×10^{-78} to 7.2×10^{75}.

Built-In Functions

The default value for the second argument of the FIXED built-in function is 15 for binary data, and 5 for decimal data.

The default value for the second argument of the FLOAT built-in function is 21 for binary data, and 6 for decimal data.

The length of the bit string which is the value returned by the UNSPEC function is defined by the type of the argument.

Argument Type	Length of Bit String
FIXED BINARY (p,q)	32
FIXED DECIMAL (p,q)	8*FLOOR ((p+2)/2)
FLOAT BINARY (p)	32 if p ≤ 21 64 if p ≥ 22
FLOAT DECIMAL (p)	32 if p ≤ 6 64 if p ≥ 7
CHARACTER (n)	8*n
BIT (n)	n
POINTER	32
OFFSET	32
AREA	(Number of bytes allocated + 16)*8

The length of the string returned by the ONSOURCE and DATAFIELD built-in functions is subject to an implementation maximum of 255 characters.

MAX, MIN, MOD Built-In Functions

When the arguments to these functions have different attributes, all the arguments are converted, before the function is invoked, to the highest characteristics. Contrary to the language specification, both the precision and the scale factor of an argument will be adjusted. If all the arguments are FIXED, application of the highest-characteristics rule may, in conjunction with the maximum precision defined by the implementation, cause truncation and hence an inaccurate result. For example:

```
DCL X FIXED DECIMAL (12,1),
    Y FIXED DECIMAL (12,9);
  Z = MOD (X,Y);
```

Here Z (whatever its attributes) will be wrong. X and Y are stored in a temporary field which would have, according to the precisions of the operands, a precision larger than the implementation permits. Therefore the implementation-defined maximum is applied, resulting in a precision of (15,9). Y can be stored satisfactorily inside such a field but X is truncated, with the loss of its five high-order digits.

MOD Built-In Function

When the MOD function is used with FIXED arguments of different scale factors, the results may be truncated. If SIZE is enabled, an error message will be printed; if SIZE is disabled, no error message will be printed and the result is undefined.

COMPLETION Built-In Function and Pseudo-variable

The COMPLETION built-in function and pseudo-variable were previously defined in PL/I with the name EVENT. The (F) compiler implements the COMPLETION built-in function and pseudo-variable whether the keyword used is COMPLETION or EVENT.

STRING Built-In Function

The argument may be a scalar, array, or structure variable that consists of one of the following:

1. Bit strings
2. Character strings
3. Decimal numeric pictures
4. A mixture of (2) and (3)

It may not be an expression.

The argument can be ALIGNED or UNALIGNED; if it is ALIGNED, padding is not included in the result.

The concatenated string in the result has a maximum length of 32767 bytes.

Length of Identifiers

The following types of identifiers should contain not more than seven characters:

All EXTERNAL data identifiers
EXTERNAL PROCEDURE and ENTRY labels
EXTERNAL Files
CONDITION identifiers

If this restriction is exceeded, the first four characters are concatenated with the last three to form an abridged identifier.

In addition, such identifiers must not start with the letters IHE, lest they conflict with the names of Library modules.

Subscripted Identifiers

For subscripted identifiers, the maximum number of characters in the subscript is limited to 225 characters. This figure includes the first left parenthesis, the commas, and the final right parenthesis; it excludes redundant characters such as blanks and plus signs.

CHECK Lists

The maximum number of entries in a CHECK condition, whether in a prefix list or in an ON statement, is 510.

The maximum number of data items being checked at any point in the compilation varies between 2078-2n and 3968-2n, where n is the number of currently checked items which have the attribute EXTERNAL.

If a structure or part of a structure is in a CHECK condition, the number of items in this restriction must include all elements of the structure.

OBJECT-TIME CONVENTIONS AND RESTRICTIONS

Data-Directed Input

When the CHECK condition is enabled for data-directed input, assignment of each element of an array will cause the whole array to be written out.

Edit-Directed Output

E- and F-format items are rounded, not truncated.

CHECK Condition

If an identifier which is read in by a GET DATA statement is included in a CHECK list anywhere in the program, then the CHECK condition is raised, and will be treated as enabled unless the block containing the GET DATA statement has an explicit NOCHECK prefix. If the CHECK condition is raised, SYSTEM action will be taken unless the GET DATA statement lies within the dynamic scope of an ONCHECK statement for the identifier in question.

If a READ statement with the EVENT option has a KEYTO or an INTO variable for which the CHECK condition is enabled, the value of the variable will be printed immediately after the READ statement, not after the WAIT statement. Consequently the printed values of the variable will be the old, not the new values.

CONVERSION ON-Condition

If a return from an ON-unit for CONVERSION is made, then, unless it was entered on account of a SIGNAL statement, the data conversion will be reattempted. This implies the use of corrective measures on the field in error using either the ONSOURCE or the ONCHAR pseudo-variables. If corrective action is not taken in the ON-unit, and normal return is attempted, a message will be printed and the ERROR condition will be raised.

If the ONSOURCE or ONCHAR pseudo-variable is used outside an ON-unit, or in an ON-unit other than either a CONVERSION ON-unit or an ERROR or FINISH ON-unit entered because of system action for CONVERSION, then a message is printed and the ERROR condition is raised.

If the ONSOURCE built-in function is used out of context, a null string is returned. If the ONCHAR function is used out of context, a blank is returned.

ON-Units and Entry Parameter Procedures

There is an implementation limit to the number of ON-units and/or entry parameter procedures which can be active at any time. An entry parameter procedure is one that passes an entry name as parameter to a procedure it calls. The total permissible number of these ON-units and/or entry parameter procedures is 127.

Exponentiation

The expression X**(-N) for N>0 is evaluated by taking the reciprocal of X**N. This may cause the OVERFLOW condition to occur as the intermediate result is computed, which corresponds to UNDERFLOW in the original expression.

Collating Sequence

In the execution of PL/I programs, comparisons of character data will observe the collating sequence resulting from the representations of characters in bytes of System/360 storage, in extended binary coded decimal interchange code (EBCDIC). The BCD and EBCDIC punched card codes and graphics for the PL/I 60-character set are tabled in collating sequence order in Figure 4 of this publication.

ENTRY Names as Arguments and ON Statements in Recursive Contexts

In the first version of the (F) compiler, ENTRY parameters were invoked with the environment existing at the time of invocation. In subsequent versions, they will be invoked with the environment existing at the time when the ENTRY name was passed as an argument.

Example:

```
P1:  PROC RECURSIVE;
     B = 1;
     CALL P4(P3);
     RETURN;

P4:  ENTRY(PP);
     B = 2;
     CALL PP;
     P3: PROC;
         PUT DATA (B);
         END;
     END;
```

Note: For the first version, the above procedure gave B = 2;, for subsequent versions it gives B = 1;.

In the first version of the (F) compiler, ON-units in recursive contexts were entered with the environment existing when the condition occurred. In subsequent versions, the ON-unit will be entered with the environment which was in existence when the ON statement was executed.

Example:

```
P:  PROCEDURE RECURSIVE;
    DECLARE I STATIC INITIAL (0),
        M AUTOMATIC;
    I = I + 1;
    M = I;
    IF I = 1 THEN DO;
        ON OVERFLOW PUT DATA (M);
        END;
    IF I = 3 THEN SIGNAL OVERFLOW;
             ELSE CALL P;
    END;
```

Note: In the first version, the procedure gave M = 3; subsequent versions give M = 1.

These modifications of semantics can affect only those programs which contain both recursive procedures and either entry parameters or ON statements.

Concatenated Data Sets

Concatenation of data sets with "unlike attributes" (device type, record format, etc.) is not supported at object time.

LOCATE MODE

ON Conditions

UNDEFINEDFILE condition for implicit OPEN is raised in the normal way.

TRANSMIT condition is raised when it is detected, which may be some statements later. ONKEY does not necessarily give the key of the record that caused an I/O transmission error.

RECORD condition is raised in the usual way for LOCATE. It cannot occur for READ SET or REWRITE without the FROM option.

KEY condition is raised in the usual way except for a LOCATE statement on an INDEXED file with RKP≠0. If this happens then the sequence of operations is:

1. On the LOCATE statement the KEYFROM key is checked for sequence; the KEY condition is raised if a sequence error is found.

2. Processing continues until the next operation on the file.

3. At the next operation on the file, the embedded key in the buffer is checked against the KEYFROM string that was given in the LOCATE statement. If they differ, then:

   Explicit CLOSE: the KEY condition is raised

   Implicit close: the KEYFROM string replaces the embedded key in the buffer, an error message is written on the console, and the file is closed.

4. On normal return from the ON unit, control passes to the next statement The current statement is not executed.

Record Alignment

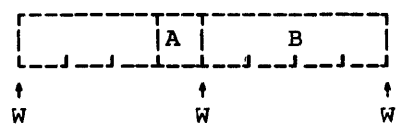
The user must pay attention to record alignment within the buffer when using locate mode I/O. The first data byte of the first record in a block is generally aligned in a buffer on a doubleword boundary (see Figure 39); the next logical record begins at the next available byte in the buffer. The user must ensure that the alignment of this byte matches the alignment requirements of the based variable with which the record is to be associated.

Most of the alignment problems described here occur in ALIGNED based or non-based variables. If these variables were UNALIGNED, the preservation of the record alignment in the buffer would be considerably easier.

If a VB format record is to be constructed with logical records defined by the structure:

```
DCL 1 S,
      2 A CHAR(1),
      2 B FIXED BINARY;
```

this structure is mapped as in Figure 39.



W = Word boundary

Figure 39. Format of Structure S

If the block was created using a sequence of WRITE FROM(S) statements, the format of the block would be as in Figure 40, and it can be seen that the alignment in the buffer differs from the alignment of S.

There is no problem if the file is then read using move mode READ statements, e.g., READ INTO(S), because information is moved from the buffer to correctly aligned storage.

If, however, a structure is defined as:

```
1 SBASED BASED(P) LIKE S;
```

and READ SET(P) statements are used, then, reference to SBASED.B would, for the first record in the block, be to data aligned at a doubleword plus one byte, and would probably result in a specification interrupt.

The same problem would have arisen had the file originally been created by using the statement:

```
LOCATE SBASED SET(P);
```

Again, for the first record in the block, P would be set to address a doubleword and references to SBASED.B would be invalid.

In both cases the problem is avoided if the structure is padded in such a way that B is always correctly aligned:

```
1 S,
  2 PAD CHAR(3),
  2 A CHAR(1),
  2 B FIXED BINARY;
```

The block format would now be as in Figure 41; B is always on a word boundary. Padding may be required at the beginning and end of a structure to preserve alignment.

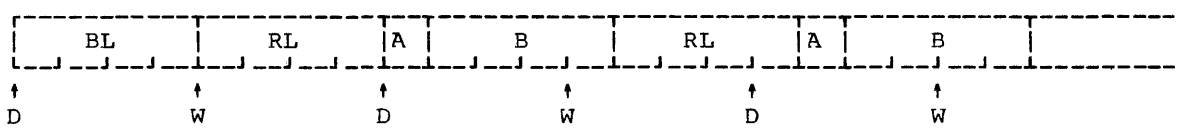
The alignment of different types of record within a buffer is shown in Figure 42. For all organizations and record types except blocked records in INDEXED files with RKP=0, the first data byte in a block is always on a doubleword boundary. The position of any successive records in the buffer depends on the record format.

For unblocked INDEXED, the LOCATE statement will use a hidden buffer if the data set key length is not a multiple of 8. The pointer variable will point at this hidden buffer.

A special problem arises when using locate mode in conjunction with based variable containing adjustable extents, i.e., containing a REFER attribute. Consider the based structure:

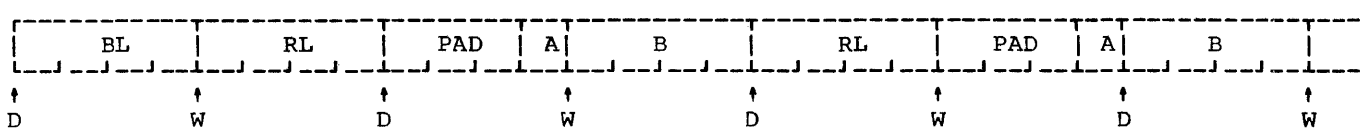
```
1 S BASED(P),
  2 N,
  2 C CHAR (L REFER (N));
```

If it is desired to create blocked V-format records of this type, using locate mode, then this record alignment must be such that N is word aligned. If L is not a multiple of 4 then, if the alignment of the



BL = Block length D = Doubleword boundary
RL = Record length W = Word boundary

Figure 40. Block Created from Structure S



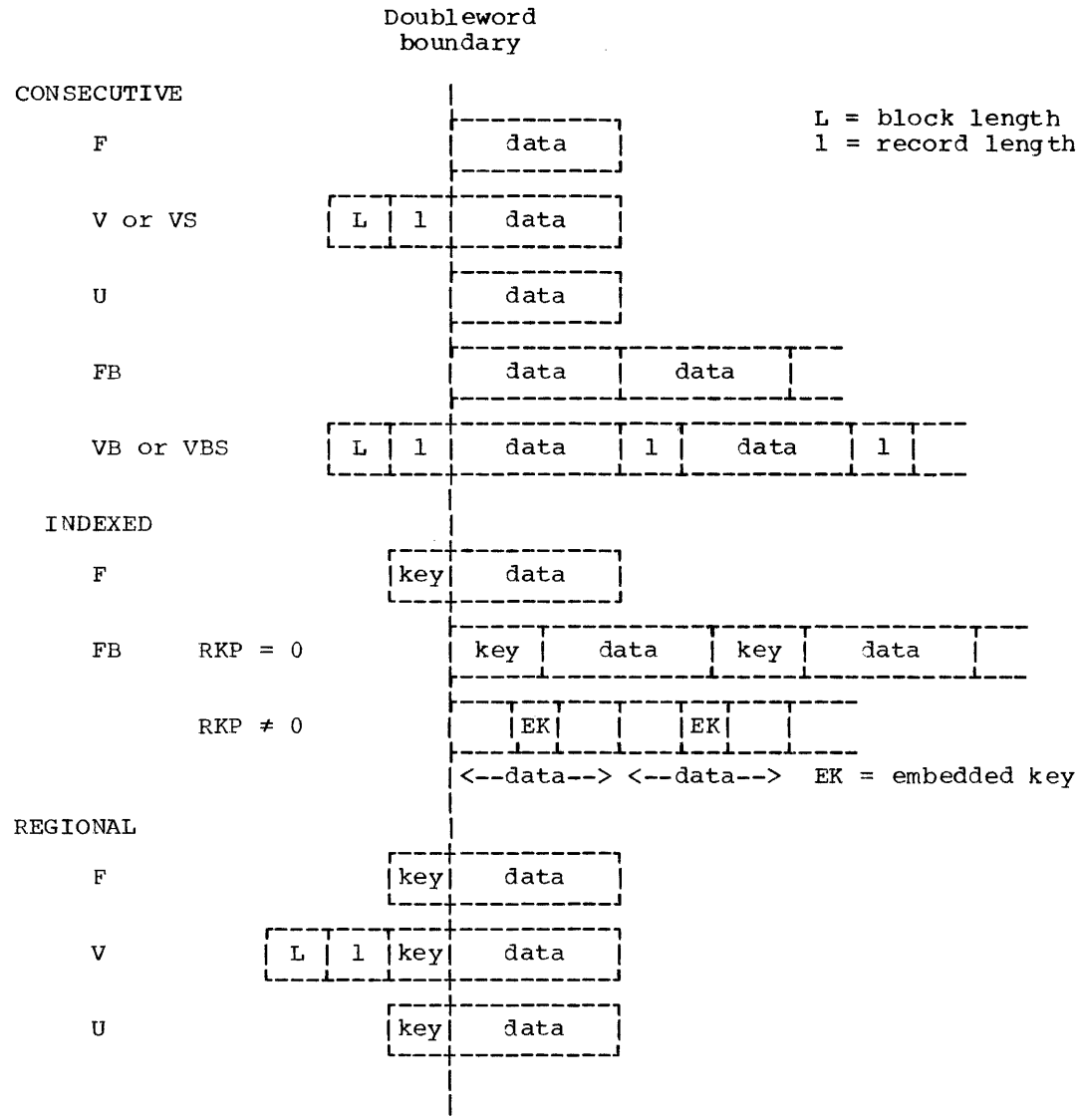
BL = Block length D = Doubleword boundary
RL = Record length W = Word boundary

Figure 41. Block Created by Structure S with Correct Alignment

current record is correct, that of the following record will be wrong. Correct alignment can be obtained by the following sequence:

```
LENGTH = L;
     /* SAVE DESIRED LENGTH L */
L = 4* CEIL(L/4);
     /* ROUND UP TO MULTIPLE OF 4 */
LOCATE S FILE (F);
N = LENGTH;
     /* SET REFER VARIABLE */
```

This technique can be adapted to other uses of the REFER attribute.



•Figure 42. Alignment of Data in a Buffer in Locate Mode I/O, for Different Formats and File Organizations

APPENDIX C: OBJECT PROGRAM ORGANIZATION AND CONVENTIONS

INTRODUCTION

The main features of PL/I which affect object code organization are, briefly, as follows:

1. The block structure, rules of scope, and storage class
2. Interruption activity
3. Asynchronous facilities
4. Data types and extents

The term 'block' is used to mean either a PROCEDURE or a BEGIN-END group which contains declarations of AUTOMATIC variables. Because such storage is dynamic, it is allocated only when the block is entered. It is stressed that the code for a block is present at all times and as that code is read-only, one copy only is necessary, even if the procedure is recursive. Therefore, when a block is activated, storage is allocated.

PSEUDO-REGISTER VECTOR (PRV)

The pseudo-register vector (PRV) is a task-oriented communications area, addressed through register PR(12): one PRV is established for each task or subtask. Object programs not employing multitasking will have only one PRV. Because of the possibility of multitasking in a dynamic environment, the PRV is contained in dynamic storage. The PRV contains a number of pseudo-registers which effectively operate as implicit arguments and give information about, for example, current program status. Since all references to specific pseudo-registers within the PRV are made by the addition of a fixed displacement to the base address (contained in register PR) of the PRV, read-only modules are able to address dynamically allocated storage obtained for any task (library workspace, for example).

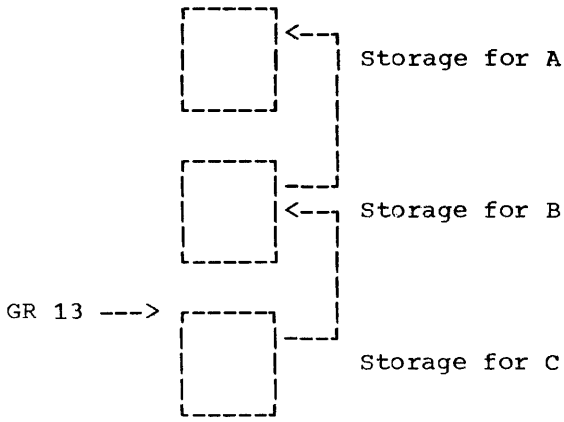
Addressing of the PRV within library modules is effected by using Q-type address constants which are fixed during link-editing. All pseudo-register address constants within the PL/I implementation are two bytes in length; the maximum size of a PRV is 4096 bytes.

RUN-TIME STACK

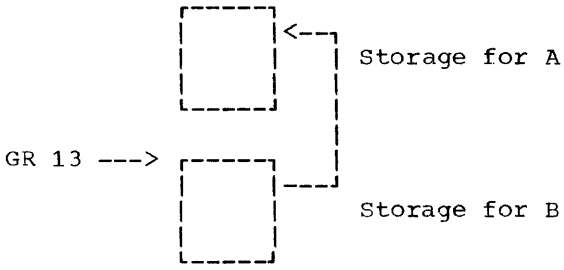
At any one time during the execution of a PL/I program, control resides in one particular block of code. If this block invokes another, then the old block is "pushed down" in the stack when the new one assumes control. This stacking of blocks of executable code is exactly matched by the allocations of AUTOMATIC storage, and so both storage and code can be controlled by the same stacking mechanism. This is achieved by means of the dynamic storage allocation facilities (provided by the library or control program) in association with general register 13. This register is maintained such that it always points at the storage area corresponding to the current block of code. As the areas are chained together it is a comparatively simple matter to "pop up" the stack when leaving a block, by resetting general register 13 to point at the previous area in the chain, and to release the current area. The following program and diagram illustrate this mechanism.

```
A:  PROCEDURE;
      .
      .
      .
   CALL B;
      .
      .
      .
B:  PROCEDURE;
      .
      .
      .
  C:  BEGIN;
      .
      .
      .
    END C;
    .
    .
    .
   END B;
    .
    .
    .
  END A;
```

When control is in block C, the stack looks like this:



When control reverts from block C to block B, the stack takes on the following appearance:



These areas of storage are known as dynamic storage areas (DSAs).

DYNAMIC STORAGE AREA (DSA)

Each Dynamic Storage Area is an entry in the run-time stack and, consequently, each DSA must possess a standard layout. Figure 43 shows the functional content of a DSA.

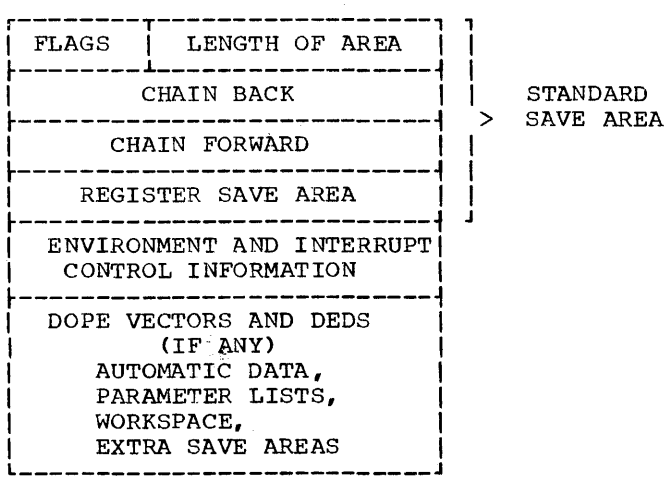


Figure 43. Functional Content of a Dynamic Storage Area

When a routine in System/360 assembler language is used in a tasking environment, any DSA obtained in it must have at least 108 bytes. (The minimum for a DSA in a non-tasking environment is 100 bytes).

VARIABLE DATA AREA (VDA)

In some instances, it may be necessary to obtain storage after the DSA has been allocated. For example, the following statements would cause this situation to arise:

```
A : PROC;
    DCL I INITIAL(10);
    DCL A(I) CHAR(6);
    .
    .
    .
    END A;
```

In the above example, the prologue of procedure A would first establish its DSA, then initialize the variable I, and then obtain more dynamic storage for the array A. This storage is known as a variable data area (VDA) and can conceptually be regarded as a secondary stack based on individual members of the primary stack of DSAs.

VDAs are required in a number of circumstances in addition to that outlined above. In particular, code may be compiled to obtain a VDA whenever there is a demand for temporary workspace (e.g. for strings of more than 256 bytes).

PROLOGUES AND EPILOGUES

It can be seen from a study of the language and from the foregoing description, that it is necessary for each block to have a "prologue" and an "epilogue." Each prologue must save registers in the area provided, obtain storage for its new DSA, update register 13 to point at the DSA, maintain the forward and backward chains, initialize the environment and interrupt control entries, initialize the dope vectors (if any), and decide which entry point is required. The epilogue has complementary tasks to perform before finally returning.

In view of the fact that these prologues/epilogues must be executed each time a block is entered and left, space economy is achieved by incorporating as much of this work into subroutines as possible.

The library routines contain certain sections of the prologue and epilogue which are common to all prologues and epilogues. The functions of the prologue subroutines are:

1. To preserve the environment of the invoking block
2. To obtain and initialize AUTOMATIC storage for the block
3. To provide chaining mechanisms to enable the program's progress to be traced.

The main functions of the epilogue subroutine are to release storage for the block, and to recover the environment of the invoking block before returning control to it.

INTERRUPT ACTIVITY AND CONTROL

All interrupt activity in PL/I is controlled at the language level by means of prefix options and ON statements. At object time, all interruptions are channeled through the library error handling module, IHEERR. Hardware interruptions are passed to IHEERR because a SPIE macro is issued as part of a PL/I program's initialization, in which IHEERR is nominated as the exit for the user. Programmed interruptions reach IHEERR by means of the normal branch and link instruction.

In order to locate the ON-unit (if any) specified in the PL/I program, the error handler searches through the stack until it finds the necessary information in the interruption control area of a DSA. If the search is unsuccessful, then SYSTEM action is taken.

INITIAL ENTRY TO PROCEDURES WITH THE MAIN OPTION

In order to achieve the proper initialization of PL/I programs, the primary entry is always to a compiler generated control section, IHENTRY. This calls the appropriate PL/I library initialization routine (IHESA or IHETSA); the choice depends on the OPT parameter and whether the main procedure has the TASK option or not. The routine selected provides the PRV and Library workspace, issues a SPIE macro, and then transfers control to the address contained in a control section named IHEMAIN. This address constant is produced by the compiler for each external procedure with the MAIN option. The situation is illustrated in Figure 44. If an argument list is to be passed to the PL/I program, IHESA or IHETSA must be entered at the appropriate entry point.

If more than one module has the MAIN option, the linkage editor will accept the first appearance of the control section IHEMAIN in its input stream and ignore the rest.

COMBINATION OF PL/I WITH OTHER LANGUAGES

The programming example in Appendix D illustrates the inclusion of a subroutine written in assembler language. This subroutine performs some of the functions of a PL/I procedure and shows how to make use of PL/I library subroutines. The functional capabilities of the subroutine are fully described in the commentary contained within it.

CALLING SEQUENCES AND REGISTER USAGE

Linkage between PL/I procedures is provided by the standard operating system System/360 calling sequence with the exception that the last argument in the parameter list is not indicated by a '1' in the high-order bit. (Refer to the publication IBM System/360 Operating System, Supervisor and Data Management Services). In addition, the trace forward chain is maintained.

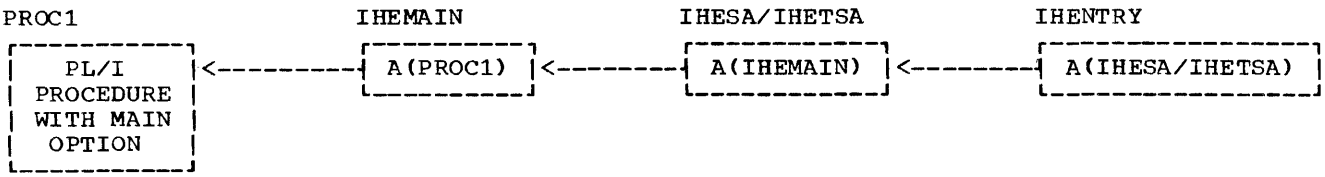


Figure 44. Initial Entry to Procedures with the MAIN Option

Register usage in object code produced by the PL/I (F) compiler observes the following conventions:

Register	Description of Use
0	Work Register
1	Work Register and Parameter List Pointer
2-8	Work Registers
9	Address Register
10	Program Base
11	Static Data Pointer
12	Pseudo-Register Vector Pointer
13	Current DSA Pointer
14	Arithmetic and Return Register
15	Arithmetic and Branch Register

The linkage conventions for library modules, some of which employ an internal standard, are described below.

LINKAGE CONVENTIONS FOR LIBRARY MODULES

Intermodule linkage conforms to either the System/360 operating system calling sequence standards or an internal PL/I standard. The latter applies only to certain library modules which require a higher degree of efficiency. The internal standard passes the addresses of arguments in registers rather than through a parameter list; in all other respects, it is identical to the System/360 operating system standards. The internal standard is optionally employed under the following constraints:

1. The number of arguments must be less than eight
2. The arguments must be fixed in number

If these constraints cannot be met, the operating system standard is employed. Definition of operating system linkage conventions is provided in the publication IBM System/360 Operating System, Supervisor and Data Management Services. Several salient features of these conventions are:

1. Arguments are passed by name, not by value
2. Caller provides a standard format register save area addressed through register DR(13)
3. Registers used by the called program are saved in the save area provided
4. If the called module in turn calls another module, it provides that module with a save area, updating register DR to address the new save area, and chaining it to the old save area
5. Upon return to the calling module, registers RB(2) through LR(14), the program mask and PICA will be unchanged, while registers R0(0),RA(1), BR(15), the floating-point registers, and the condition code may be changed

PRESENTATION OF ARGUMENTS

The normal System/360 operating system standard of passing parameters, by pointing at a list of addresses, is employed in PL/I object code. These addresses point directly at data in the case of scalar variables only, provided they are not strings. In all other cases, the address passed is that of a control block known as a "dope vector." Various types of dope vector are described below.

STRING DOPE VECTOR (SDV)

This control block specifies storage requirements for string data. An SDV consists of eight bytes (word-aligned), in the format shown in Figure 45.

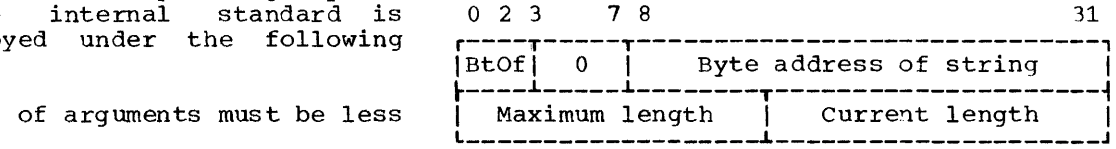


Figure 45. Format of the String Dope Vector (SDV)

Definition of SDV fields:

BtOf (Bit offset): If the string is a bit string, positions 0 to 2 of the SDV specify the offset of the first bit of the string within the addressed byte. The bit offset is only applicable to bit strings which form part of a data aggregate, and then only if that aggregate has the UNALIGNED attribute.

Byte address of string: For both character and bit string this three-byte field specifies the address of the initial byte of the string.

Maximum length: Halfword binary integer which specifies the number of storage units allocated for the string; byte count if character, bit count if bit. This value does not vary for a particular generation of its associated string.

Current length: Halfword binary integer which specifies the number of storage units, within the maximum length, currently occupied by the string.

The two length fields exist to accommodate strings with the VARYING attributes; in the instance of a fixed-length string, the two fields contain identical values. For both fields the maximum value is 32,767.

AREA DOPE VECTOR

This has the same format as the string dope vector, except that both the length fields are the same and contain the length of the AREA (not including the 16-byte control area).

ARRAY DOPE VECTOR (ADV)

This control block, shown in Figure 46, contains information required in the derivation of elemental addresses within an array data aggregate. The ADV has three functions:

1. Given an array, to step through the array in row-major order
2. Given the subscript values of an array element, to determine the element address
3. Given an element address, to determine its subscript values

Within PL/I implementation, arrays are stored in row-major order, upward in storage. The elements of an array are normally held in contiguous storage, although this may not be so if the array is a member of a structure. However, any lack of contiguity is transparent to algorithms which employ an array dope vector.

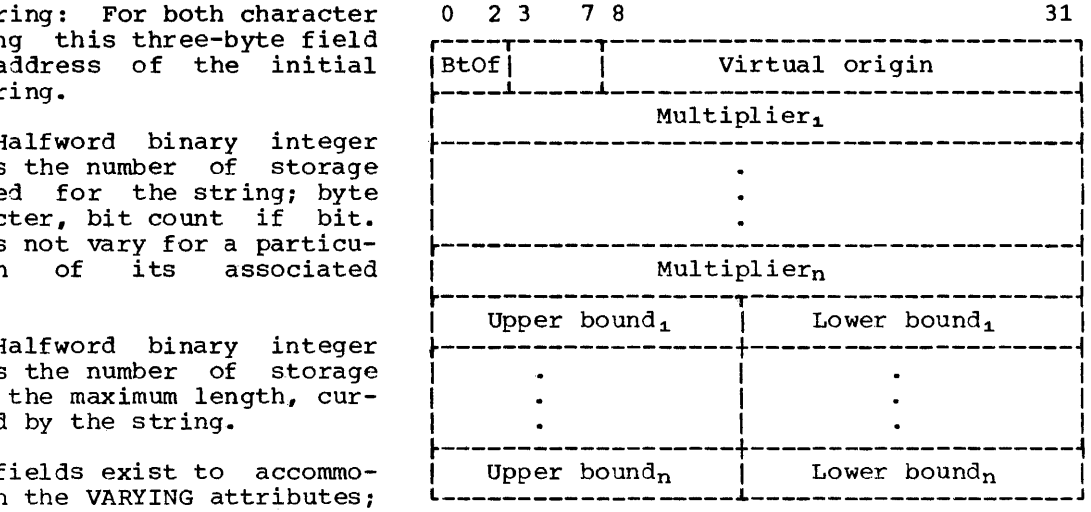


Figure 46. Format of the Array Dope Vector (ADV)

The ADV contains (2n + 1) 32-bit words, where n is the number of dimensions of the array. The number of dimensions in the array is not described within the ADV, but is passed to the library as an additional argument.

Definitions of ADV fields:

BtOf (= Bit offset): For an array of bit strings with the UNALIGNED attribute, this is the bit offset from the byte address of the virtual origin.

Virtual origin: The byte address of the array element whose subscript values are all zero, i.e.,X(0,...,0);this element need not be an actual member of the array, in which case the virtual origin will address a location in storage outside the actual bounds of the array.

Multiplier: Multipliers are fullword binary integers which in the standard ADV algorithm effect dimensional incrementation or decrementation to locate an element. Multipliers are in bits for fixed-length, bit string arrays, otherwise in bytes.

Upper Bound: Halfword binary integer, specifying the maximum value permitted for a subscript in the ith dimension. This value may be negative.

Lower Bound: Halfword binary integer, specifying the minimum value permitted for a subscript in the ith dimension. The value may be negative.

ADV Algorithm: Given subscript values for an n-dimensional array, the address of any element is computed as:

$$\text{Address} = \text{virtual origin} + \sum_{i=1}^{n} S_i * M_i$$

where S_i = value of the ith subcript
M_i = value of the ith multiplier

For an array of bit strings with the UNALIGNED attribute, the origin is a bit address formed by concatenating the virtual origin and the bit offset. For all other arrays, the origin is the virtual origin.

STRUCTURE DOPE VECTOR

This control block contains information required to derive, directly or indirectly, the address of all elements of the structure.

The format of a structure dope vector is determined as follows. The dimensions which have been applied to the major structure or to minor structures are inherited by the contained structure base elements; undimensioned non-string base elements are assigned a dope vector consisting only of a single-word address field. The structure dope vector is then derived by concatenating the dope vectors which the base elements would have if they were not part of a structure, in the order in which the elements appear in the structure.

Example:

The following structure would have a dope vector of the form shown in Figure 47:

```
1 A, 2 B (10), 3 C (10) CHAR (6),
               3 D BIT (10) VARYING,
     2 E,      3 F FLOAT (5),
               3 G (10) FIXED,
               3 H CHAR (3);
```

STRING ARRAY DOPE VECTOR (SADV)

This control block contains information required to derive, directly or indirectly (through a secondary array of SDV entries),

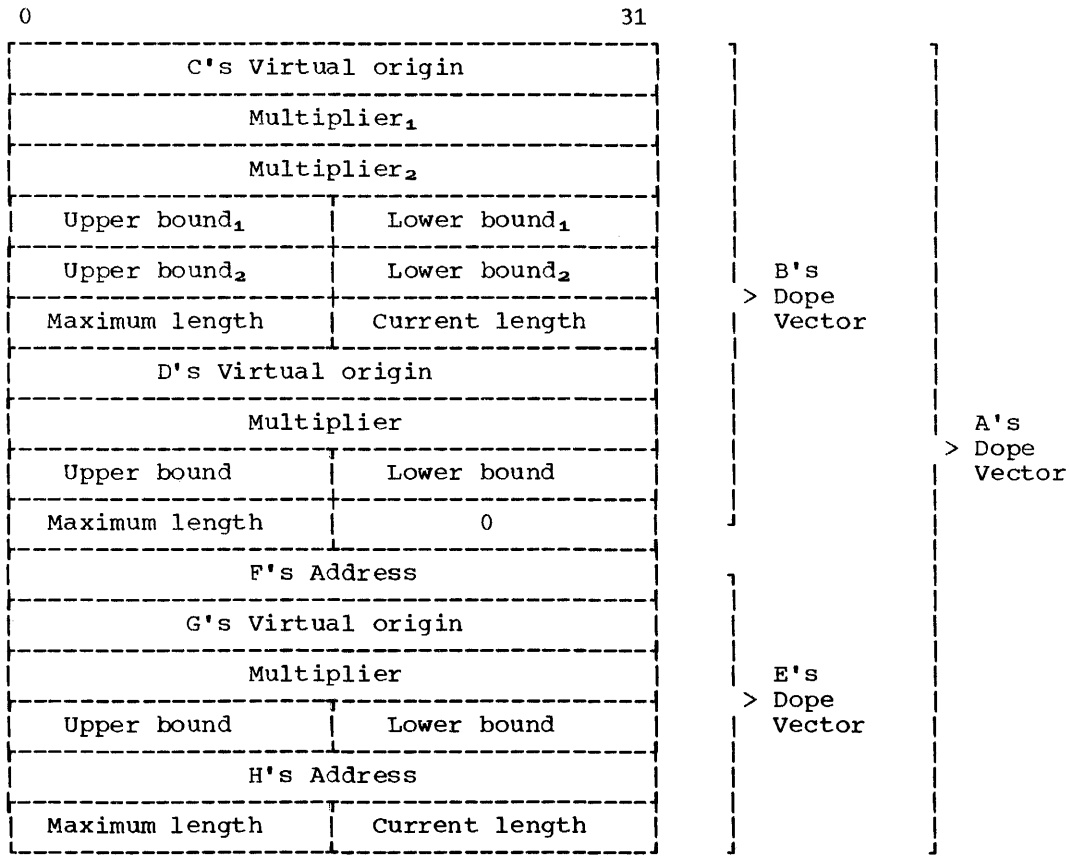


Figure 47. Format of the Structure Dope Vector (SDV)

the address of elemental strings. The SADV is identical to the basic ADV, with the addition of a fullword which describes the string length, as shown in Figure 48.

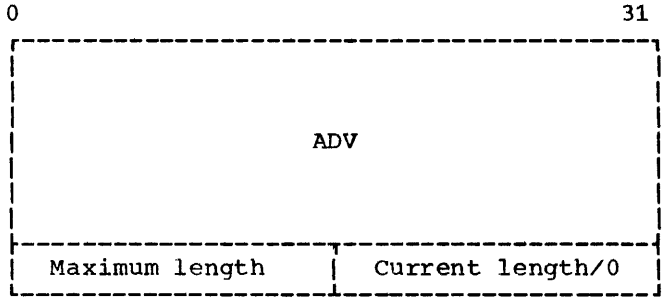


Figure 48. Format of the Primary String Array Dope Vector (SADV)

Fixed-length strings require only a primary dope vector. The two length fields are set to the same value, which is the declared length of the string.

Variable-length strings require, in addition to the primary dope vector, a secondary dope vector which consists of SDV entries for each elemental string within the array. The secondary dope vector is addressed via the primary dope vector by the standard ADV algorithm; having located the relevant SDV, the actual string data is directly addressable. The maximum-length field appended to the ADV is set to the maximum length of each array element. The current-length field is set to zero for arrays of VARYING strings, while for fixed-length strings it is set to the maximum length. Whereas the multipliers of the ADV for a fixed-length string apply to the actual string data, those of the ADV for a variable-length string apply to the secondary dope vector of SDV entries.

STRUCTURE MAPPING

For any structure (major or minor), the length, alignment requirement, and position relative to a doubleword boundary will depend on the lengths, alignment requirements, and relative positions of its members. The process of determining these requirements for each level in turn and finally for the complete structure, is known as structure mapping.

During the structure mapping process, the F compiler minimizes the amount of unused storage (padding) between members of the structure. It completes the entire process before the structure is allocated, according (in effect) to the rules discussed in the following paragraphs. It is necessary for the user to understand these rules for such purposes as determining the record length required for a structure when record-oriented input/output is used, and for determining the amount of padding or rearrangement required to ensure correct alignment of a structure for locate-mode input/output (see "Record Alignment" in Appendix B).

Structure mapping is not a physical process. Although during this discussion such terms as "shifted" and "offset" are used, these terms are used purely for ease of discussion, and do not imply actual movement in storage; when the structure is allocated, the relative locations are already known as a result of the mapping process.

The mapping for a complete structure reduces to successively combining pairs of items (elements, or minor structures whose individual mappings have already been determined). Once a pair has been combined, it becomes a unit to be paired with another unit, and so on until the complete structure has been mapped. The rules for the process are therefore categorized as:

Rules for determining the order of pairing

Rules for mapping one pair

These rules are described below, and the example at the end of this section shows an application of the rules in detail.

Note: To follow these rules, it is necessary to appreciate the difference between logical level and level number. The item with the greatest level number is not necessarily the item with the deepest logical level. If the structure declaration is written with consistent level numbers or suitable indention (as in the detailed example given after the rules), the logical levels are immediately apparent. In any case, the logical level of each item in the structure can be determined by applying the following rule to each item in turn, starting at the beginning of the structure declaration:

The logical level of a given item is always one unit deeper than that of the nearest preceding item that has a lesser level number than the given item.

For example:

```
DCL 1 A, 4 B, 5 C, 5 D, 3 E, 8 F, 7 G;

    1   2   3   3   2   3   3
```

The lower line shows the logical level for each item in the declaration.

Rules for Order of Pairing

The steps in determining the order of pairing are as follows:

1. Find the minor structure with the deepest logical level (which we will call logical level n).

2. If the number of minor structures at logical level n exceeds one, take the first one of them as it appears in the declaration.

3. Using the rules for mapping one pair (see below), pair the first two elements appearing in this minor structure, thus forming a unit.

4. Pair this unit with the next element (if any) appearing in the declaration for the minor structure, thus forming a larger unit.

5. Repeat rule 4 until all the elements in the minor structure have been combined into one unit. This completes the mapping for this minor structure; its alignment requirement and length, including any padding, are now determined and will not change (unless the programmer changes the structure declaration). Its offset from a doubleword boundary will also have been determined; note that this offset will be significant during mapping of any containing structure, and it may change as a result of such mapping.

6. Repeat rules 3 through 5 for the next minor structure (if any) appearing at logical level n in the declaration.

7. Repeat rule 6 until all minor structures at logical level n have been mapped. Each of these minor structures can now be thought of as an element for structure mapping purposes.

8. Repeat the process for minor structures at the next higher logical level; that is, make n equal to (n - 1) and repeat rules 2 through 7.

9. Repeat rule 8 until n = 1; then repeat rules 3 through 5 for the major structure.

Rules for Mapping One Pair

(As stated earlier, terms apparently implying physical storage are used here only for ease of discussion; the storage thus implied may be thought of as an imaginary model consisting of a number of contiguous doublewords. Each doubleword has eight bytes numbered zero through 7, so that the offset from a doubleword boundary can be given; in addition, the bytes in the model may be numbered continuously from zero onwards, starting at any byte, so that lengths and offsets from the start of a structure can be given.)

1. Begin the first item of the pair on a doubleword boundary; or, if the item is a minor structure that has already been mapped, offset it from the doubleword boundary by the amount indicated.

2. Begin the other item of the pair at the first valid position following the end of the first item. This position will depend on the alignment requirement of the second item. Alignment and length requirements for elements are given in Figures 49 and 49.1. (If the item is a minor structure, its alignment requirement will have been determined already.)

3. Shift the first item towards the second item as far as the alignment requirement of the first item will allow. The amount of shift determines the offset of this pair from a doubleword boundary.

After this process has been completed, any padding between the two items will have been minimized and will remain unchanged throughout the rest of the operation. The pair can now be considered to be a unit of fixed length and alignment requirement; its length is the sum of the two lengths plus padding, and its alignment requirement is the higher of the two alignment requirements (if they differ).

Effect of UNALIGNED Attribute

The example of structure mapping given below shows the rules applied to a structure declared ALIGNED, because mapping of aligned structures is more complex owing to the number of different alignment requirements. Briefly, with the F compiler, the general effect of the UNALIGNED attribute is to reduce fullword and doubleword alignment requirements down to byte, and to reduce the alignment requirement for bit strings from byte down to bit. The same structure mapping rules apply, but the reduced alignment requirements are used. This means that unused storage between items can only be bit padding within a byte, and never a complete byte; bit pad-

Variable Type	Stored Internally as	Storage Requirements (in Bytes)	Alignment Requirements	Explanation
BIT (n)	One byte for each group of 8 bits (or part thereof)	n rounded - up 8		
CHARACTER (n)	One byte per character	n		Data may begin on any byte 0 through 7
PICTURE .	One byte for each PICTURE character (except M, V, K, G)	Number of PICTURE characters other than M, V, K, and G	Byte	
DECIMAL FIXED (p,q)	1/2 byte per digit plus 1/2 byte for sign	p + 1 rounded ----- up 2		
BINARY FIXED (p,q)	Binary integer			
BINARY FLOAT (p) p < 22	Short floating point	4	Full word	Data may begin on byte 0 or 4 only
DECIMAL FLOAT (p) p < 7				
POINTER	-			
OFFSET	-			
LABEL	-	8		
TASK	-	28		
EVENT	-	32		
BINARY FLOAT (p) 21 < p < 54	Long floating point	8	Double word	Data may begin on byte 0 only
DECIMAL FLOAT (p) 6 < p < 17				
AREA	-	16+length		

•Figure 49. Summary of Alignment Requirements for ALIGNED Data

ding may occur when the structure contains bit strings.

POINTER, OFFSET, LABEL, TASK, EVENT, and AREA data cannot be unaligned. If a structure has the UNALIGNED attribute and it contains an element that cannot be unaligned, then UNALIGNED is ignored for that element; the element is aligned by the compiler and error message IEM3181I is put out. For example, in a program with the declaration

```
DECLARE 1 A UNALIGNED,
          2 B,
          2 C AREA(100);
```

C is given the attribute ALIGNED, as the inherited attribute UNALIGNED conflicts with AREA.

Variable Type	Stored Internally as	Storage Requirements (in Bytes)	Alignment Requirements	Explanation
BIT (n)	As many bits as are required, regardless of byte boundaries	n bits	Bit	Data may begin on any bit in any byte 0 through 7
CHARACTER (n)	One byte per character	n		
PICTURE	One byte for each PICTURE character (except M,V,K,G)	Number of PICTURE characters other than M,V,K and G		
DECIMAL FIXED (p,q)	1/2 byte per digit, plus 1/2 byte for sign	p + 1 rounded ----- up 2		Data may begin on any byte 0 through 7
BINARY FIXED (p,q)	Binary integer		Byte	
BINARY FLOAT (p) p < 22 DECIMAL FLOAT (p) p < 7	Short floating point	4		
BINARY FLOAT (p) 21 < p < 54 DECIMAL FLOAT (p) 6 < p < 17	Long floating point	8		

Note: POINTER, OFFSET, LABEL, TASK, EVENT, and AREA data cannot be UNALIGNED.

•Figure 49.1. Summary of Alignment Requirements for UNALIGNED Data

Example of Structure Mapping

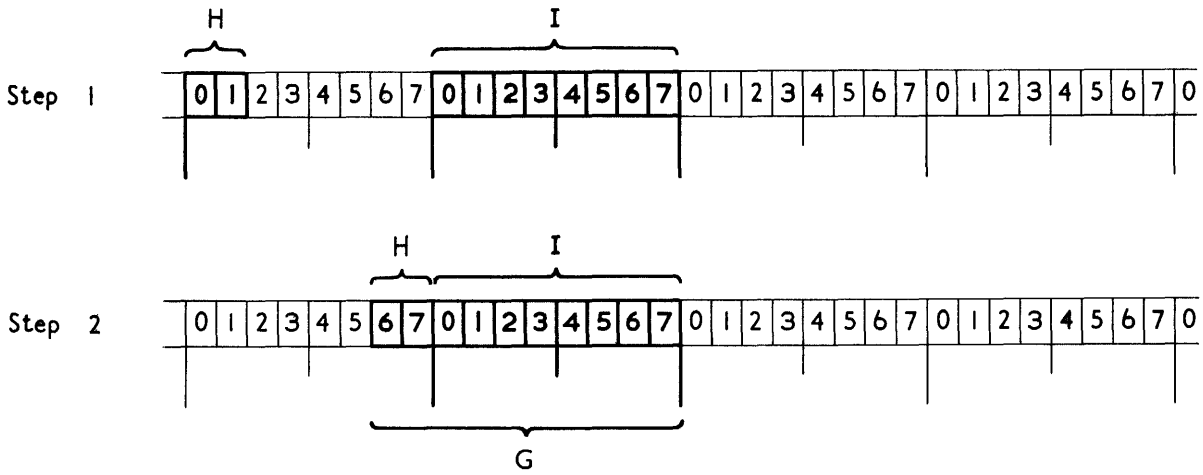
This example shows the application of the structure mapping rules for a structure declared as follows:

```
DECLARE 1 A ALIGNED,
          2 B POINTER,
          2 C,
            3 D FLOAT DECIMAL(14),
            3 E,
              4 F LABEL,
              4 G,
                5 H CHARACTER(2),
                5 I FLOAT DECIMAL(13),
              4 J FIXED BINARY(15,0),
            3 K CHARACTER(2),
            3 L FIXED BINARY(15,0),
          2 M,
            3 N,
              4 P FIXED BINARY(15,0),
              4 Q CHARACTER(2),
              4 R FLOAT DECIMAL(2),
            3 S,
              4 T FLOAT DECIMAL(15),
              4 U CHARACTER(2),
            3 V POINTER,
          2 W PICTURE '$9V99';
```

The minor structure at the deepest logical level is G, so that this is mapped first. Then E is mapped, followed by N, S, C, and M, in that order. Finally, the major structure A is mapped. For each structure, a table is given showing the steps in the process, accompanied by a diagram giving a visual interpretation of the process. At the end of the example, the structure map for A is set out in the form of a table showing the offset of each member from the start of A.

	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from G
				Begin	End		
Step 1	H	Byte	2	0	1		
	I	Double	8	0	7		
Step 2	*H	Byte	2	6	7		0
	I	Double	8	0	7	0	2
	G	Double	10	6	7		

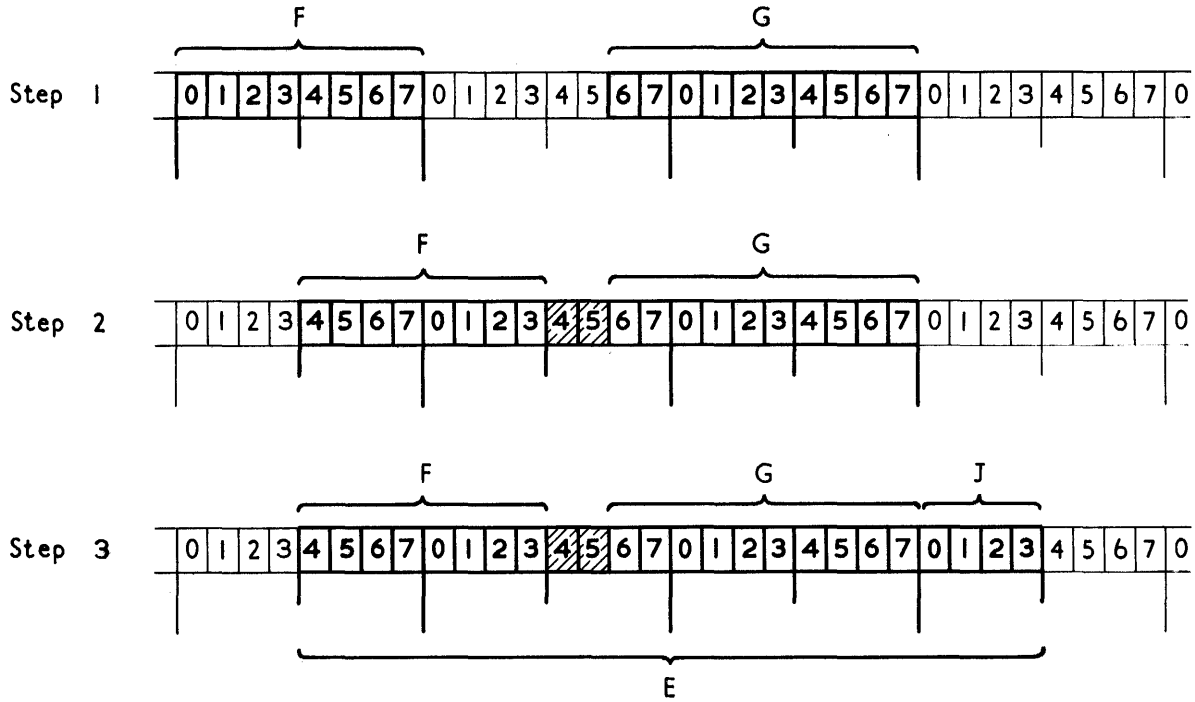
*First item shifted right



●Figure 50. Mapping of Minor Structure G

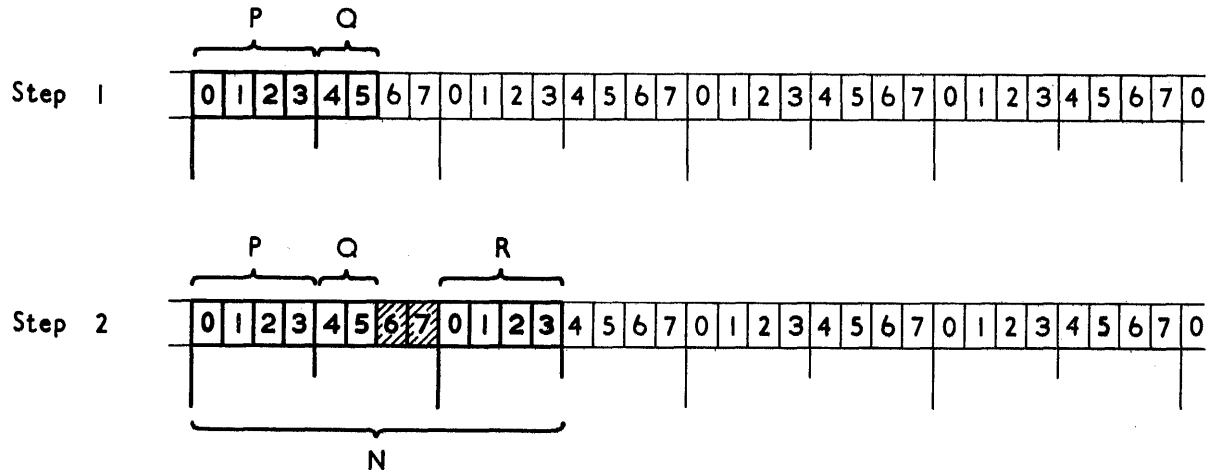
	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from E
				Begin	End		
Step 1	F	Word	8	0	7		
	G	Double	10	6	7		
Step 2	*F	Word	8	4	3		0
	G	Double	10	6	7	2	10
Step 3	F through G	Double	20	4	7		
	J	Word	4	0	3	0	20
	E	Double	24	4	3		

*First item shifted right



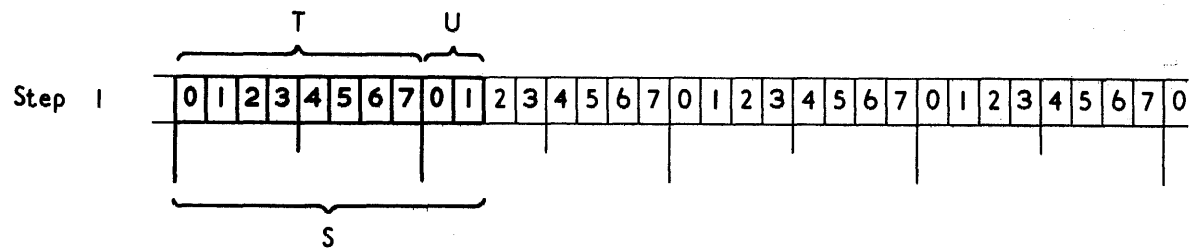
●Figure 51. Mapping of Minor Structure E

	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from N
				Begin	End		
Step 1	P	Word	4	0	3		0
	Q	Byte	2	4	5		4
Step 2	P through Q	Word	6	0	5		
	R	Word	4	0	3	2	8
	N	Word	12	0	3		



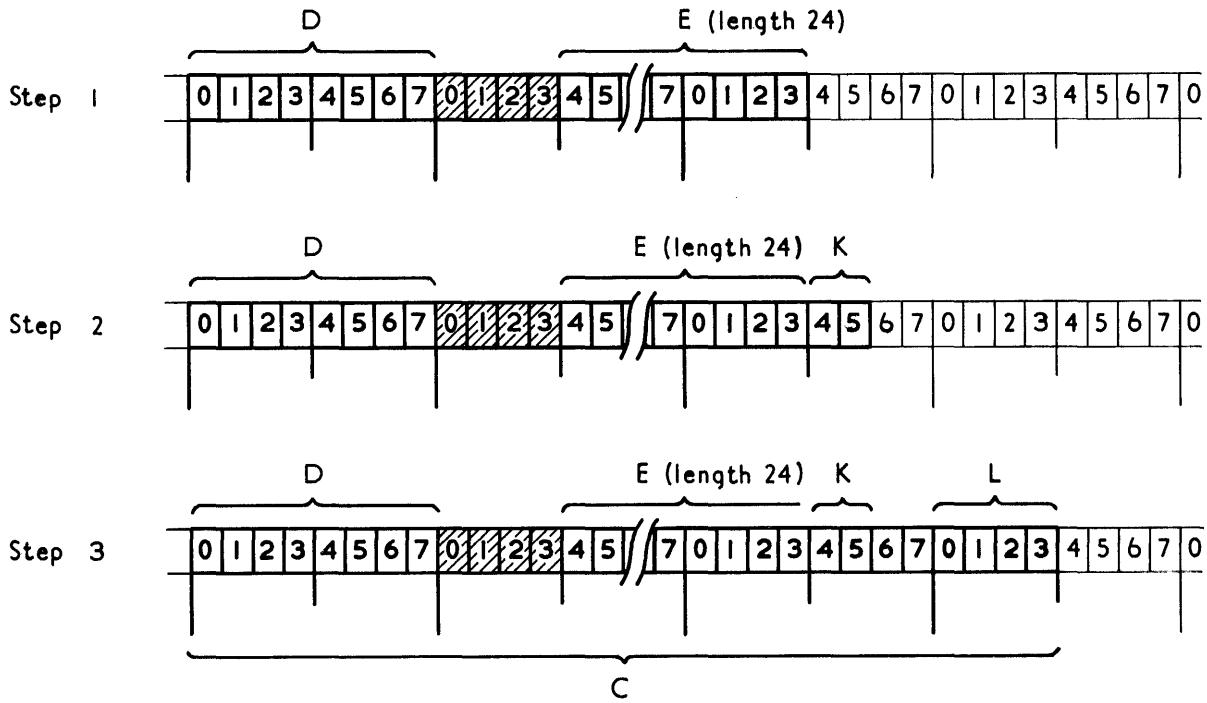
•Figure 52. Mapping of Minor Structure N

	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from S
				Begin	End		
Step 1	T	Double	8	0	7		0
	U	Byte	2	0	1	0	8
	S	Double	10	0	1		



•Figure 53. Mapping of Minor Structure S

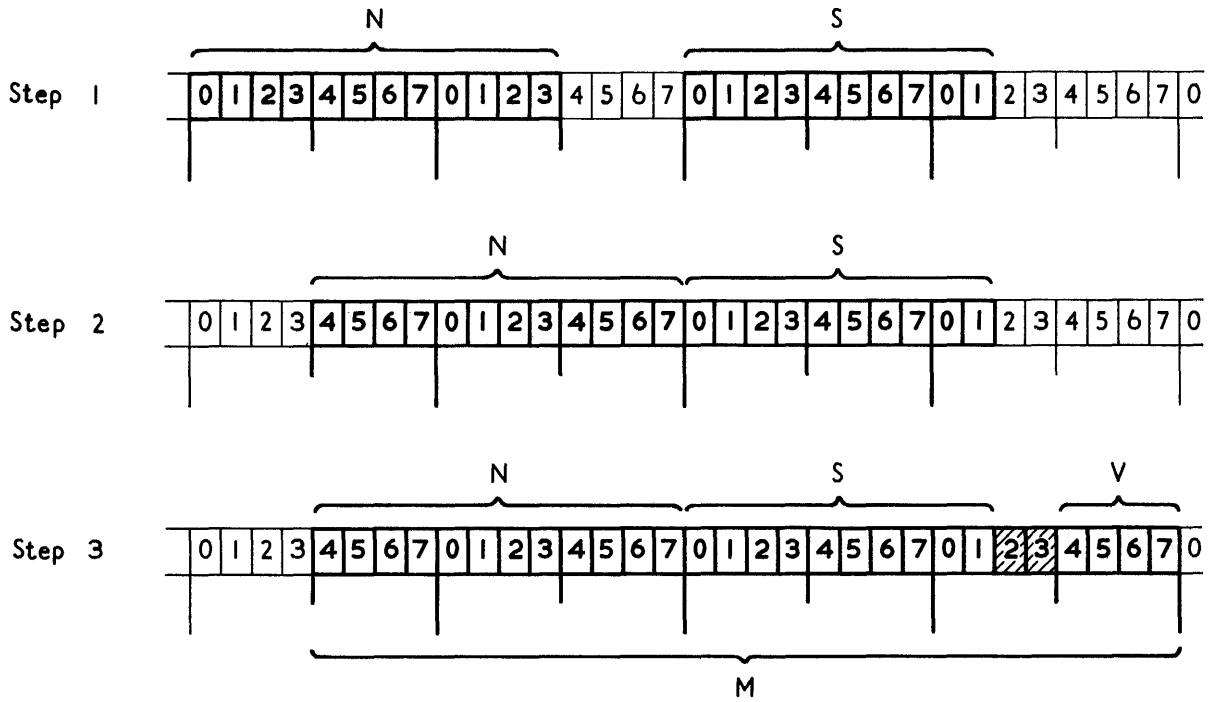
	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from C
				Begin	End		
Step 1	D	Double	8	0	7		0
	E	Double	24	4	3	4	12
Step 2	D through E	Double	36	0	3		
	K	Byte	2	4	5	0	36
Step 3	D through K	Double	38	0	5		
	L	Word	4	0	3	2	40
	C	Double	44	0	3		



•Figure 54. Mapping of Minor Structure C

	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from M
				Begin	End		
Step 1	N	Word	12	0	3		
	S	Double	10	0	1		
Step 2	*N	Word	12	4	7		0
	S	Double	10	0	1	0	12
Step 3	N through S	Double	22	4	1		
	V	Word	4	4	7	2	24
	M	Double	28	4	7		

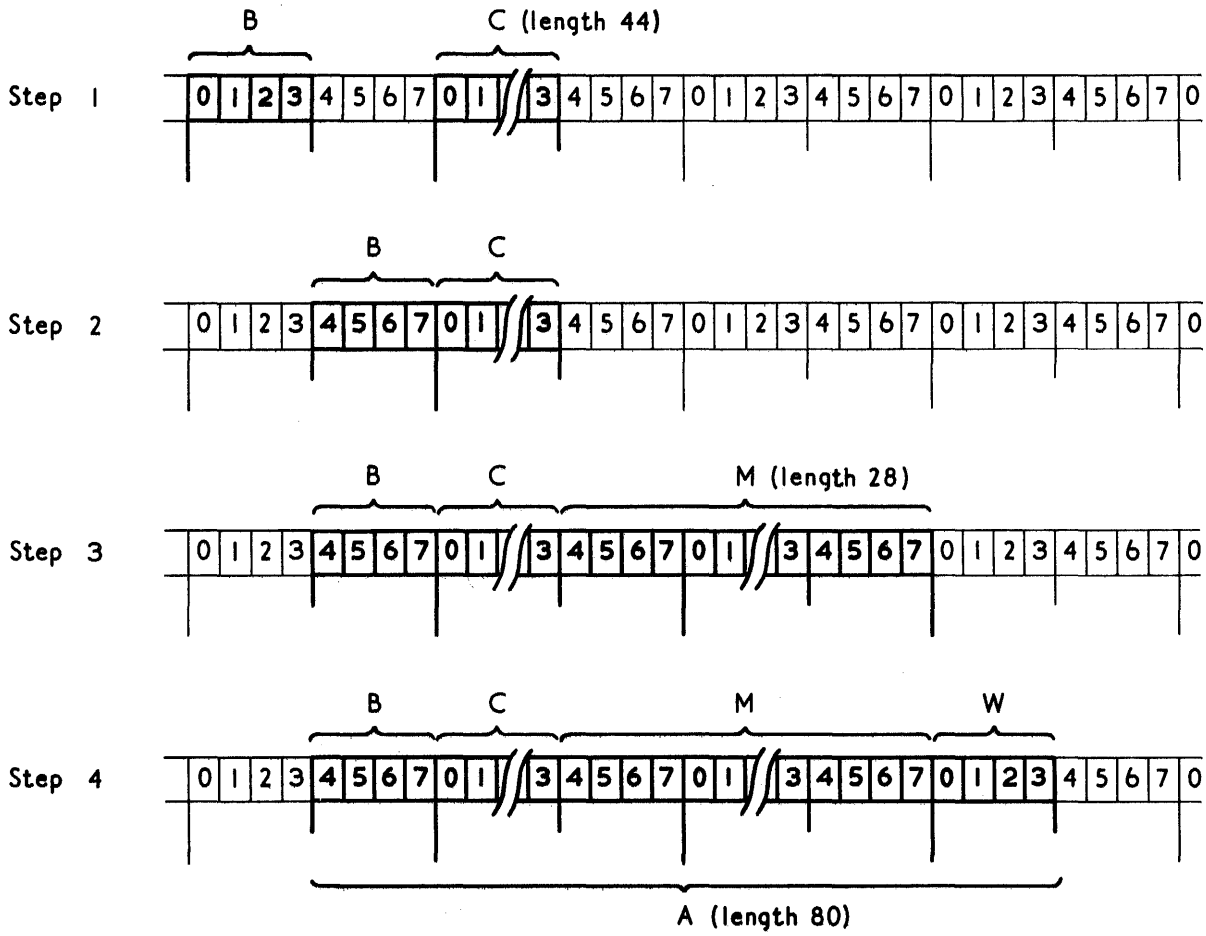
*First item shifted right



•Figure 55. Mapping of Minor Structure M

	Name of Item	Alignment Requirement	Length	Offset from Doubleword		Length of Padding	Offset from A
				Begin	End		
Step 1	B	Word	4	0	3		
	C	Double	44	0	3		
Step 2	*B	Word	4	4	7		0
	C	Double	44	0	3	0	4
Step 3	B through C	Double	48	4	3		
	M	Double	28	4	7	0	48
Step 4	B through M	Double	76	4	7		
	W	Byte	4	0	3	0	76
	A	Double	80	4	3		

*First item shifted right



•Figure 56. Mapping of Major Structure A

A				From A
B				0
C			From C	4
D			0	4
padding (4)			8	12
E		From E	12	16
F		0	12	16
padding (2)		8	20	24
G	From G	10	22	26
H	0	10	22	26
I	2	12	24	28
J		20	32	36
K			36	40
padding (2)			38	42
L			40	44
M			From M	48
N		From N	0	48
P		0	0	48
Q		4	4	52
padding(2)		6	6	54
R		8	8	56
S		From S	12	60
T		0	12	60
U		8	20	68
padding (2)			22	70
V			24	72
W				76

•Figure 56.1. Offsets in Final Mapping of Structure A

ALLOCATION AND RELEASE OF STORAGE IN AN AREA

The AREA attribute defines storage reserved for the allocation of based variables. In this implementation the amount reaserved consists of sixteen bytes of control information plus the amount requested for each allocation (see Figure 57). The control information and each allocation start on a doubleword boundary.

The amount of storage required is defined in one of two ways:

1. Area size: This is the amount of storage reserved in the declaration of an area. For example, in the statement:

   DCL Z AREA(500);

   The size of area Z is 500 bytes. The maximum size that can be declared is 32767 bytes. The effective minimum size is 8 bytes; 0 bytes can be declared but the AREA condition would be raised if an attempt was made to allocate a based variable into it. If a size is not declared, a default value of 1000 bytes is assumed.

2. Area length This is the amount of storage needed for an area allocation. It is the area size plus the 16 bytes of control information. For example:

   DCL Z AREA(500) CONTROLLED;
   ALLOCATE Z;

   creates an allocated area Z of 516 bytes. The maximum length is 32783 bytes; the effective minimum length is 24 bytes. The default length is 1016 bytes. When an area is written using RECORD I/O, the amount transmitted is the area length, not the area size.

The storage allocated for a variable always starts and ends on a doubleword boundary, whatever the amount requested. The amount allocated is

8*(CEIL((L+H)/8))

when L = Variable length (in bytes)

H = Offset (in bytes) of beginning of this variable from a doubleword boundary. Maximum value of H is 7 bytes.

The total amount of storage allocated for variables in an area is the extent; the amount at a particular time in the current extent. The value of the current extent is increased every time an allocation is made beyond the end of the current extent. It is decreased only if the allocation that forms the end of the current extent is

freed; then the value of the current extent is reduced accordingly. The current extent is not changed when an allocation within the extent is freed.

When an allocation within the extent is freed, the freed area is called a free element. Each free element starts with two control words:

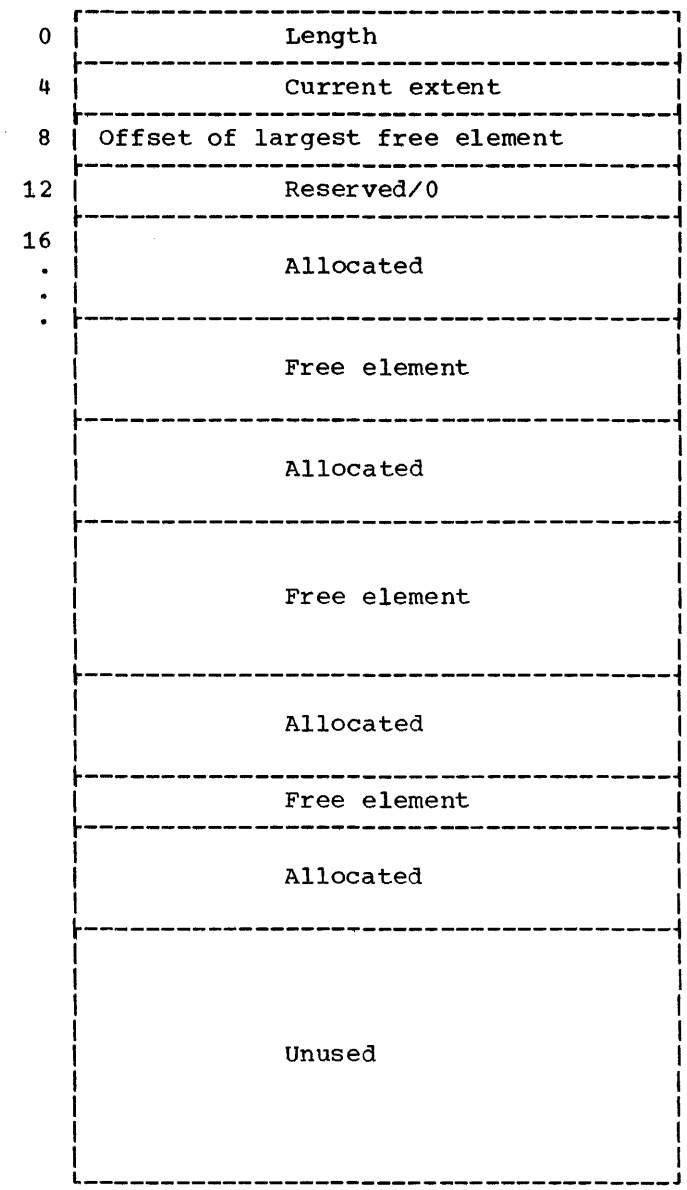
Word 1: length of free element(in bytes)

Word 2: Offset (in bytes) of next smallest free element. The last free element contains the offset of the fourth word of the control information.

The chain of free elements created in this way in the free list.

When an allocation is required, the free list is examined to find the smallest free element in which the variable will fit. If no free element is large enough, then the unused section at the end of the current extent is used. If this in turn is not large enough, the AREA condition is raised.

When an element is freed, the free list is scanned to find if this element is contiguous with another free element. If so, the two are combined and then placed in the free list according to the combined size. If there is no contiguous element, the freed element is placed in the free list according to its size.



Note: If there is a free list, the fourth control word contains 0; otherwise it is unused.

Figure 57. AREA Format, Showing Example of Allocated Storage and Free Elements

APPENDIX D: PROGRAMMING EXAMPLE

```
                                                                        PAGE      1
VERSION 4 RELEASE 17                    OS/360 PL/I COMPILER (F)        DATE 68.306

PL/I F COMPILER OPTIONS SPECIFIED ARE AS FOLLOWS--

     LIST,ATR,XREF,OPT=1,LC=60,SIZE=96000

THE COMPLETE LIST OF OPTIONS USED DURING THIS COMPILATION IS--   EBCDIC
                                                                CHAR60
                                                              NOMACRO
                                                                SOURCE2
                                                              NOMACDCK
                                                                COMP
                                                                SOURCE
                                                                ATR
                                                                XREF
                                                              NOEXTREF
                                                                LIST
                                                                LOAD
                                                              NODECK
                                                                FLAGW
                                                                STMT
                                                                SIZE=096000
                                                                LINECNT=060
                                                                OPT=01
                                                                SORMGIN=(002,072)
                                                              NOEXTDIC
                                                              NONEST
                                                                OPLIST


     /*THIS IS A SMALL PROGRAMMING EXAMPLE                          */          PAGE      2

               /*THIS IS A SMALL PROGRAMMING EXAMPLE                                    */
               /*TO SHOW HOW ASSEMBLER LANGUAGE SUBPROGRAMS CAN BE INCORPORATED         */
               /*INTO A PL/I MAIN PROGRAM.                                              */

               /*THIS MODULE IS DESIGNED TO ACCEPT A CHARACTER STRING ARGUMENT          */
               /*OF ANY SIZE AND WRITE IT IN THE DATA SET DEFINED ON THE SYSPRINT       */
               /*DD CARD.                                                               */

               /*THE ACTUAL OUTPUT OPERATION IS CARRIED OUT BY THE PL/I LIBRARY         */
               /*MODULE IHEPRTB, WHICH IS INVOKED BY THE ASSEMBLER LANGUAGE             */
               /*MODULE (PRINTER).                                                      */

               /*'PRINTER' IS INVOKED BY WRITING THE STANDARD CALL STATEMENT            */
               /*IN PL/I, USING THE ENTRY NAME TO WHICH CONTROL IS TO BE PASSED.        */

               /*TO RUN THIS JOB, THERE MUST BE A DD CARD TO DESCRIBE THE DATA SET      */
               /*SYSPRINT.  THE PL/I LIBRARY CONTAINS A DCB FOR THIS DATA SET, AND      */
               /*WILL OPEN THE FILE FOR OUTPUT WHEN IT IS INVOKED, IF IT IS NOT         */
               /*ALREADY OPEN.                                                          */

               /*TO EXECUTE THIS PROGRAM, THE PL/I AND ASSEMBLER LANGUAGE OBJECT        */
               /*MODULES MUST BE LINKAGE EDITED TOGETHER INTO ONE MODULE.  THE          */
               /*MODULES MUST BE PREPARED FOR LINKAGE EDITING IN THE ORDER PL/I         */
               /*MODULE FIRST, THEN ASSEMBLER LANGUAGE MODULE.                          */

1              TEST:  PROCEDURE OPTIONS(MAIN);

2                     DECLARE A1 CHAR(35);       /*A1 IS THE VARIABLE TO BE          */
                                                 /*WRITTEN ON SYSPRINT.              */

3                     C,B = 2;                   /*INITIALIZE C AND B                */
4                     D = 4;                     /*AND D.                            */

5                     A1 = (B*(C**2))/D;         /*CALCULATE A VALUE IN FLOATING     */
                                                 /*POINT AND CONVERT IT TO A CHAR-   */
                                                 /*ACTER STRING.                     */

6                     CALL PRINTER(A1);          /*TO USE FACILITIES PROVIDED BY     */
                                                 /*THE LIBRARY ROUTINE TO PUT OUT    */
                                                 /*ERROR MESSAGES, IT IS NECESSARY   */
                                                 /*TO CALL THE SUBROUTINE PRINTER.   */
                                                 /*THIS IS WRITTEN IN ASSEMBLER      */
                                                 /*LANGUAGE, AND ILLUSTRATES TWO     */
                                                 /*IMPORTANT  ASPECTS OF THIS SORT   */
                                                 /*OF LINKAGE.                       */

7                     END TEST;
```

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
2	A1	AUTOMATIC,STRING,CHARACTER 5,6
	B	AUTOMATIC,DECIMAL,FLOAT(SINGLE) 3,5
	C	AUTOMATIC,DECIMAL,FLOAT(SINGLE) 3,5
	D	AUTOMATIC,DECIMAL,FLOAT(SINGLE) 4,5
	PRINTER	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 6
1	TEST	ENTRY,DECIMAL,FLOAT(SINGLE)

STORAGE REQUIREMENTS

THE STORAGE AREA FOR THE PROCEDURE LABELED TEST IS 240 BYTES LONG
THE PROGRAM CSECT IS NAMED TEST AND IS 184 BYTES LONG
THE STATIC CSECT IS NAMED ***TEST AND IS 100 BYTES LONG

STATIC INTERNAL STORAGE MAP

```
000038  8A0680              DED FOR TEMP
000040  00000000            A..   TEST
000044  00000000            A..   IHEDNCA
000030  00000000            A..   IHESAFA
000048  00000000            A..   PRINTER
000050  4140000041200000    CONSTANTS
000058  000000AC00230023    D.V. SKELETON
000060  8A0600              DED
000068  2C                  DED
```

```
  OBJECT LISTING

* STATEMENT NUMBER  1

* PROCEDURE                              TEST

* REAL ENTRY                             TEST
000000  47 F0 F 00A             B       10(0,15)
000004                          DC      AL1(4)
000005                          DC      C'TEST'
00000A  90 EB D 00C             STM     14,11,12(13)
00000E  41 A0 F 00A             LA      10,10(0,15)
000012  41 80 A 026             LA      8,38(0,10)
000016  58 B0 A 01E             L       11,30(0,10)
00001A  58 F0 B 020             L       15,32(0,11)
00001E  58 00 A 022             L       0,34(0,10)
000022  05 EF                   BALR    14,15
000024  07 F8                   BR      8
000026  07 00                   NOPR    0
000028  00000000                DC      A(SI.)
00002C  000000F0                DC      F'240'
000030  05 A0                   BALR    10,0
000032  05 A0                   BALR    10,0

* PROLOGUE BASE
000034  41 90 D 0D0             LA      9,208(0,13)
000038                  CL.4    EQU     *
000038  50 DC 0 000             ST      13,PR..TEST(12)
00003C  92 00 D 062             MVI     98(13),X'00'
000040  92 01 D 063             MVI     99(13),X'01'
000044  92 C0 D 000             MVI     0(13),X'C0'
000048  D2 07 D 098 B 050       MVC     DV..A1(8),SKDV..04
                                CC
00004E  41 F0 D 0AC             LA      15,A1
000052  50 F0 D 098             ST      15,DV..A1
000056  41 A0 A 028             LA      10,CL.2
00005A  07 00                   NOPR    0

* PROCEDURE BASE
00005C                  CL.2    EQU     *

* APPARENT ENTRY                         TEST

* STATEMENT NUMBER  3
00005C  92 03 D 063             MVI     99(13),X'30'
000060  D2 03 D 0A8 B 04C       MVC     C(4),C..0414
000066  D2 03 D 0A4 B 04C       MVC     B(4),C..0414

* STATEMENT NUMBER  4
00006C  92 04 D 063             MVI     99(13),X'04'
000070  D2 03 D 0A0 B 048       MVC     D(4),C..0440

* STATEMENT NUMBER  5
000076  92 05 D 063             MVI     99(13),X'05'
00007A  78 00 D 0A8             LE      0,C
00007E  7C 00 D 0A8             ME      0,C
000082  7C 00 D 0A4             ME      0,B
000086  7D 00 D 0A0             DE      0,D
00008A  70 00 D 090             STE     0,WS1.1
00008E  41 10 D 090             LA      1,WS1.1
000092  41 20 B 030             LA      2,DED..04A0
000096  41 30 D 098             LA      3,DV..A1
00009A  41 40 B 05B             LA      4,DED..A1
00009E  58 F0 B 038             L       15,A..IHEDNCA
0000A2  05 EF                   BALR    14,15

* STATEMENT NUMBER  6
0000A4  92 06 D 063             MVI     99(13),X'06'
0000A8  41 80 D 098             LA      8,DV..A1
0000AC  50 80 D 090             ST      8,WS1.1
0000B0  41 10 D 090             LA      1,WS1.1
0000B4  58 F0 B 040             L       15,A..PRINTER
0000B8  05 EF                   BALR    14,15

* STATEMENT NUMBER  7
0000BA  92 07 D 063             MVI     99(13),X'07'
0000BE  58 F0 B 03C             L       15,A..IHESAFA
0000C2  05 EF                   BALR    14,15

* END PROCEDURE                          TEST
                                END
```



```
NO ERRORS OR WARNINGS DETECTED.

COMPILE TIME            2.09 MINS
```

```
 SYMBOL   TYPE ID   ADDR   LENGTH
PRINTER    SD  01 000000 000044
IHESADA    ER  02
IHEPRTB    ER  03
IHESAFA    ER  04
```



```
LOC  OBJECT CODE ADDR1 ADDR2 STMT        SOURCE STATEMENT

                               1*
                               2*            THIS SUBROUTINE ILLUSTRATES SOME OF THE FUNCTIONS
                               3*          NECESSARY IN ORDER TO COMMUNICATE BETWEEN A PL/I MAIN
                               4*          PROGRAM AND AN ASSEMBLER LANGUAGE SUBPROGRAM.
                               5*
                               6*            THE COMMENTS INCLUDED EXPLAIN FULLY WHAT HAS BEEN
                               7*          DONE TOGETHER WITH AN INDICATION OF WHAT MIGHT HAVE
                               8*          BEEN DONE.
                               9*
                              10*            WHILE THIS IS A TRIVIAL EXAMPLE IT DOES
                              11*          SERVE TO DEMONSTRATE MOST OF THE LINKAGE
                              12*          PROBLEMS. IT SHOULD BE NOTED THAT THE STANDARD
                              13*          SAVE MACRO COULD HAVE BEEN EMPLOYED IN THIS
                              14*          SUBPROGRAM.
                              15*
                              16*
                              17*            THIS PROGRAM IS LIMITED TO EXTRACTING THE
                              18*          ADDRESS AND CURRENT LENGTH OF A CHARACTER STRING
                              19*          FROM ITS DOPE VECTOR, AND PRESENTING THESE ITEMS AS
                              20*          ARGUMENTS TO A LIBRARY PRINT ROUTINE.
                              21*
                              22*          THE PROGRAM AND STORAGE MANAGEMENT ROUTINES USE
                                           DIFFERENT MODULES FOR MULTITASKING AND
                                           NON-MULTITASKING
```

```
 LOC  OBJECT CODE ADDR1 ADDR2 STMT        SOURCE STATEMENT

000000                        23PRINTER  START 0
000000                        24         USING *,15
000000  47F0 F00C       0000C 25         BC    15,PRINT1
000004  07                    26         DC    AL1(7)            LENGTH OF THE CHARACTER STRING
000005  D7D9C9D5E3C5D9        27         DC    C'PRINTER'      WHICH FOLLOWS. ITS PURPOSE IS TO
                              28*                              ENABLE THE PL/I SNAP OPTION TO
                              29*                              PRINT OUT A TRACE IF SO REQUIRED.
                              30*
00000C  90EB D00C      00000C 31PRINT1   STM   14,11,12(13)      SAVE REGISTERS IN CALLER'S SAVE
                              32*                              AREA. NOTE THAT THIS SUBPROGRAM IS
                              33*                              PREPARED TO PRESERVE REGISTER 12.
                              34*                              IN THE EVENT OF AN INTERRUPTION
                              35*                              THE PL/I EXECUTION ERROR PACKAGE
                              36*                              WOULD BE INVOKED.
                              37*
                              38*                                 (IF IT WERE FELT TO BE NECESSARY
                              39*                              THIS SUBROUTINE COULD HAVE HANDLED
                              40*                              INTERRUPTIONS ITSELF BY SAVING
                              41*                              REGISTER 12 AS WELL, ISSUING A
                              42*                              SPIE MACRO, AND SAVING THE FORMER
                              43*                              PICA ADDRESS. ON EXIT THESE ITEMS
                              44*                              MUST BE RESTORED.)
                              45*
                              46         DROP  15
000010  05A0                  47         BALR  10,0              ESTABLISH ADDRESSIBILITY FOR
000012                        48         USING *,10            REST OF CONTROL SECTION. THIS
                              49*                              FOLLOWS THE SYSTEM USED BY THE
                              50*                              OBJECT CODE PRODUCED BY THE F
                              51*                              COMPILER.
                              52*
                              53*           USE IS NOW MADE OF THE PL/I LIBRARY IN ORDER TO
                              54*         OBTAIN A SAVE AREA. THIS HAS BEEN DONE DYNAMICALLY AS
                              55*         THIS IS EXACTLY WHAT IS DONE BY PL/I OBJECT CODE. IF
                              56*         THERE IS NO ABSOLUTE REQUIREMENT FOR THIS CODE TO BE
                              57*         EITHER REENTRANT OR RECURSIVE THEN STORAGE COULD HAVE
                              58*         BEEN RESERVED FOR IT BY MEANS OF DC'S OR DS'S.
                              59*
000012  4100 0064      00064  60         LA    0,100               LENGTH OF DYNAMIC SAVE AREA
                                                                 (LENGTH IS 108 IF MULTITASKING IS
                                                                 USED)
000016  58F0 A026      00038  61         L     15,ADDR1          GET ADDRESS OF LIBRARY GETDSA
00001A  05EF                  62         BALR  14,15             ROUTINE, AND BRANCH TO IT.
                              63*
                              64*              IT IS NOW NECESSARY TO INITIALIZE THE SAVE AREA.
                              65*         NOT MUCH WORK IS DONE IN THIS EXAMPLE BUT PL/I OBJECT
                              66*         CODE USUALLY PERFORMS MANY MORE FUNCTIONS. IT IS NOT
                              67*         ABSOLUTELY NECESSARY TO DO MORE THAN IS INDICATED HERE
                              68*         BUT IF THE READER WISHES TO OBSERVE ALL THE PL/I
                              69*         CONVENTIONS THEN CONSIDERABLY MORE CODE WOULD BE
                              70*         REQUIRED.
                              71*
00001C  9280 D000 00000       72         MVI   0(13),X'80'           MOVE IN FLAG BYTE AS REQUIRED
                              73*                              BY LIBRARY FREEDSA ROUTINE.
                              74*
                              75*              AT THIS STAGE THE SAVING CONVENTIONS HAVE BEEN
                              76*         DEALT WITH AND ATTENTION CAN BE GIVEN TO PARAMETERS
```

```
 LOC   OBJECT CODE ADDR1 ADDR2 STMT        SOURCE STATEMENT

000020  5860 1000        00000  77         L    14,0(0,1)          GET ADDRESS OF ARGUMENT - NOTE
                                 78*                             THAT THIS IS NOT THE STRING
                                 79*                             ITSELF BUT ITS DOPE VECTOR.
000024  5810 E000        00000  80         L    1,0(0,14)          GET ADDRESS OF STRING FROM THE
                                 81*                             DOPE VECTOR.
000028  4120 E006        00006  82         LA   2,6(0,14)          GET ADDRESS OF CURRENT LENGTH
                                 83*                             OF STRING FROM DOPE VECTOR.
00002C  58F0 A02A        0003C  84         L    15,ADDR2           GET ADDRESS OF LIBRARY PRINT
000030  05EF                    85         BALR 14,15              MODULE, AND BRANCH TO IT.
                                 86*
                                 87*               UPON RETURN THIS SUBPROGRAM HAS COMPLETED
                                 88*            ITS TASK AND NOW MAKES USE OF THE LIBRARY
                                 89*            FREEDSA ROUTINE IN ORDER TO RELEASE ITS DYNAMIC
                                 90*            STORAGE (USED AS A SAVE AREA), AND TO RETURN TO
                                 91*            ITS CALLER.
                                 92*
000032  58F0 A02E        00040  93         L    15,ADDR3           GET ADDRESS OF LIBRARY FREEDSA
000036  07FF                    94         BCR  15,15              MODULE AND BRANCH TO IT.
                                 95*
                                 96*
                                 97*
000038  00000000                98ADDR1    DC   V(IHESADA)            ADDRESS OF LIBRARY GETDSA RTN
00003C  00000000                99ADDR2    DC   V(IHEPRTB)            ADDRESS OF LIBRARY PRINT RTN
000040  00000000               100ADDR3    DC   V(IHESAFA)            ADDRESS OF LIBRARY FREEDSA RTN
                                101*
                                102*
                                103         END
```

RELOCATION DICTIONARY

POS.ID	REL.ID	FLAGS	ADDRESS
01	02	1C	000038
01	03	1C	00003C
01	04	1C	000040

APPENDIX E: CATALOGED PROCEDURES

This section lists the PL/I (F) cataloged procedures. It also describes statements used to override statements and parameters in any cataloged procedure. (The use of cataloged procedures is discussed in "Job Processing".)

The procedures may be used with any of the System/360 operating system job schedulers. When parameters required by a particular scheduler are encountered by another scheduler not requiring those parameters, either they are ignored or alternative parameters are substituted automatically. For example, if these procedures are used with a sequential scheduler the following parameters, which are required for the multiprogramming option with variable number of tasks (MVT), are treated as follows:

REGION=xxxxK is ignored
DISP=SHR is interpreted as DISP=(OLD)

Installation Modifications

Before use, these procedures should be studied with a view to modifying them for greater efficiency within the particular environment of the installation. Some such modifications are detailed below.

In installations using the MVT option of the operating system, the REGION specifications for the compilation and link-editing steps must be altered where necessary to suit the available storage. The REGION specification for the compilation step must be at least 4K bytes greater than the storage specified in the compiler SIZE option (unless SIZE=999999). Failure to ensure this may result in a system abnormal termination before the compiler can provide any diagnostic aid. In the three procedures in which the Linkage Editor is invoked, a REGION of 96K has been specified for the link-editing step. If necessary, this REGION specification may be reduced to conserve storage. The minimum REGION specifications for the various design levels of the Linkage Editor are:

Linkage Editor	REGION Specification
E15	24K
E18	26K
E44	54K

Installations using the MVT option should also insert a REGION specification for the execution step in procedures PL1LFCLG and PL1LFLG, unless the default value is acceptable. The default value is established by the input reader procedure.

Installations not using the MVT option should remove the superfluous parameters.

In addition, the following general recommendations should be considered:

- under MVT, the system task initiator requires 52K bytes to initiate or terminate a step. If REGION<52K, the initiator may be held up during step termination until 52K bytes are available.
- when the MVT option is used, a SPACE parameter may be required for SYSPRINT if the device is other than a printer.
- the PARM fields for compilation and linkage editing steps should follow installation conventions.
- the SPACE and UNIT parameters for temporary data sets should be modified according to installation configuration and conventions.
- blocking factors should be specified for output data sets.

For further information on writing installation cataloged procedures, see the publication IBM System/360 Operating System, System Programmer's Guide.

COMPILATION WITH DECK OUTPUT

The cataloged procedure for compilation with deck output is shown in Figure 58.

The cataloged procedure specified in the user's EXEC statement that invokes the PL/I (F) compiler is named PL1DFC. In turn, the EXEC statement PL1D within the cataloged procedure itself, indicates that the operating system is to execute the program IEMAA (the name for the PL/I (F) compiler). If compiler options are not explicitly supplied with the procedure, default options are assumed. The programmer can override the default options by specifying the required options in the EXEC statement invoking the cataloged procedure.

```
//PL1D   EXEC  PGM=IEMAA,
               PARM='DECK,NOLOAD',
               REGION=52K
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSUT3   DD  UNIT=SYSSQ,
               SPACE=(80,(250,250)),
               SEP=SYSPRINT
//SYSUT1   DD  UNIT=SYSDA,
               SPACE=(1024,(60,60),,
               CONTIG),
               SEP=(SYSUT3,
               SYSPRINT,SYSPUNCH)
```

•Figure 58. Cataloged Procedure (PL1DFC) for Compilation with Deck Output

COMPILATION WITH OBJECT MODULE OUTPUT

The cataloged procedure for compilation with object module output is shown in Figure 59.

The cataloged procedure specified in the user's EXEC statement that invokes the PL/I (F) compiler is named PL1LFC. In turn, the EXEC statement PL1L within the cataloged procedure itself indicates that the operating system is to execute the program IEMAA (the name for the PL/I (F) compiler).

```
//PL1L   EXEC  PGM=IEMAA,
               PARM='LOAD,NODECK',
               REGION=52K
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DSNAME=&&LOADSET,
               DISP=(MOD,PASS),
               UNIT=SYSSQ,
               SPACE=(80,(250,100))
//SYSUT3   DD  UNIT=SYSSQ,
               SPACE=(80,(250,250)),
               SEP=SYSPRINT
//SYSUT1   DD  UNIT=SYSDA,
               SPACE=(1024,(60,60),,
               CONTIG),
               SEP=(SYSUT3,
               SYSPRINT,SYSLIN)
```

•Figure 59. Cataloged Procedure (PL1LFC) for Compilation with Object Module Output

COMPILATION AND LINK-EDITING

The cataloged procedure for compiling and link-editing a PL/I source program is shown in Figure 60.

The EXEC statement invoking the linkage editor is named LKED, and specifies that the operating system is to execute IEWL, the name for the linkage editor program. If linkage editor options other than those in the cataloged procedure are required, the parameters on the //LKED EXEC card must be overridden as described below.

```
//PL1L   EXEC  PGM=IEMAA,
               PARM='LOAD,NODECK',
               REGION=52K
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DSNAME=&&LOADSET,
               DISP=(MOD,PASS),
               UNIT=SYSSQ,
               SPACE=(80,(250,100))
//SYSUT3   DD  UNIT=SYSSQ,
               SPACE=(80,(250,250)),
               SEP=SYSPRINT
//SYSUT1   DD  UNIT=SYSDA,
               SPACE=(1024,(60,60),,
               CONTIG),
               SEP=(SYSUT3,SYSPRINT,
               SYSLIN)
//LKED   EXEC  PGM=IEWL,
               PARM='XREF,LIST',
               COND=(9,LT,PL1L),
               REGION=96K
//SYSLIB   DD  DSNAME=SYS1.PL1LIB,
               DISP=SHR
//SYSLMOD  DD  DSNAME=&&GOSET(GO),
               DISP=(MOD,PASS),
               UNIT=SYSDA,
               SPACE=(1024,(50,20,1),
               RLSE)
//SYSUT1   DD  UNIT=SYSDA,
               SEP=(SYSLMOD,SYSLIB),
               SPACE=(1024,(200,20))
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DSNAME=&&LOADSET,
               DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
```

•Figure 60. Compilation and Link-Editing Cataloged Procedure (PL1LFCL)

COMPILATION, LINK-EDITING, AND EXECUTION

The cataloged procedure for compiling, link-editing, and executing PL/I source programs (PL1LFCLG) is shown in Figure 61.

```
//PL1L    EXEC  PGM=IEMAA,
                PARM='LOAD,NODECK'
                REGION=52K

//SYSPRINT DD   SYSOUT=A

//SYSLIN    DD  DSNAME=&&LOADSET,
                DISP=(MOD,PASS),
                UNIT=SYSSQ,
                SPACE=(80,(250,100))

//SYSUT3    DD  UNIT=SYSSQ,
                SPACE=(80,(250,250)),
                SEP=SYSPRINT

//SYSUT1    DD  UNIT=SYSDA,
                SPACE=(1024,(60,60),,
                CONTIG),
                SEP=(SYSUT3,SYSPRINT,
                SYSLIN)

//LKED    EXEC  PGM=IEWL,
                PARM='XREF,LIST',
                COND=(9,LT,PL1L),
                REGION=96K

//SYSLIB    DD  DSNAME=SYS1.PL1LIB,
                DISP=SHR

//SYSLMOD   DD  DSNAME=&&GOSET(GO),
                DISP=(MOD,PASS),
                UNIT=SYSDA,
                SPACE=(1024,(50,20,1),
                RLSE)

//SYSUT1    DD  UNIT=SYSDA,
                SEP=(SYSLMOD,SYSLIB),
                SPACE=(1024,(200,20))

//SYSPRINT DD   SYSOUT=A

//SYSLIN    DD  DSNAME=&&LOADSET,
                DISP=(OLD,DELETE)

//          DD  DDNAME=SYSIN

//GO      EXEC  PGM=*.LKED.SYSLMOD,
                COND=((9,LT,LKED),
                (9,LT,PL1L))

//SYSPRINT DD   SYSOUT=A
```

●Figure 61. Compilation, Link-Editing, and Execution Cataloged Procedure (PL1LFCLG)

LINK-EDITING AND EXECUTION

The cataloged procedure for link-editing and executing a previously compiled PL/I program is shown in Figure 62.

```
//LKED    EXEC  PGM=IEWL,
                PARM='XREF,LIST',
                REGION=96K

//SYSLIB    DD  DSNAME=SYS1.PL1LIB,
                DISP=SHR

//SYSLMOD   DD  DSNAME=&&GOSET(GO),
                DISP=(MOD,PASS),
                UNIT=SYSDA,
                SPACE=(1024,(50,20,1),
                RLSE)

//SYSUT1    DD  UNIT=SYSDA,
                SEP=(SYSLMOD,SYSLIB),
                SPACE=(1024,(200,20))

//SYSPRINT DD   SYSOUT=A

//SYSLIN    DD  DDNAME=SYSIN

//GO      EXEC  PGM=*.LKED.SYSLMOD,
                COND=(9,LT,LKED)

//SYSPRINT DD   SYSOUT=A
```

●Figure 62. Link-Editing and Execution Cataloged Procedure (PL1LFLG)

OVERRIDING CATALOGED PROCEDURES

Cataloged procedures are composed of EXEC and DD statements. A feature of the operating system is its ability to read control statements and to modify a cataloged procedure for the duration of the current job. Overriding is only temporary; that is, the parameters added or modified are in effect only for the duration of the job; the cataloged procedure is not permanently modified. This section discusses the techniques used in modifying cataloged procedures.

Overriding Parameters in the EXEC Statement

The PL/I programmer can change compiler or linkage editor options for execution of a cataloged procedure, or he can state different conditions for bypassing a job step. For full details, see the publication IBM System/360 Operating System, Job Control Language.

Example:

Assume a source program is compiled, link-edited, and executed using the cataloged procedure PL1LFCLG. Assume further that the compiler option SIZE and the linkage editor option XREF are used, that the linkage editor option LIST is not required, and that the time for execution of the load module is charged to account number 123. The following EXEC statement adds and overrides parameters to the procedure:

```
//STEP1  EXEC  PL1LFCLG,PARM.PL1L=
                'SIZE=065536,
                LOAD,NODECK',
                REGION.PL1L=68K,
                PARM.LKED='XREF'
                ACCT.GO='123'
```

Note that if the COND parameter is to be overwritten, it must be specified as COND.catprocname.stepname, where catprocname is the name of the cataloged procedure.

Overriding and Adding DD Statements

The programmer may also override and add parameters in DD statements. The form "procstep.ddname" is used for this purpose. The "procstep" identifies the step in the cataloged procedure. If "ddname" is the name of a DD statement present in the step, the parameters in the new DD statement override parameters in the DD statement in the step.

If "ddname" is the name of a DD statement not present in the step, the new DD statement is added to the step.

In any case, the modification is effective only for the current execution of the cataloged procedure.

When overriding takes place, the original DD statement in the cataloged procedure is copied, and the parameters specified in the original DD statement are replaced by the corresponding parameters in the new DD statement. Therefore, only parameters that must be changed are specified in the new DD statement.

If more than one DD statement is modified, the overriding DD statements must be in the same order as the original DD statements in the cataloged procedure. Any DD statements that are added to the procedure must follow the overriding statements within the job step to which they belong.

In procedure steps, the programmer can catalog data sets, assign names to data sets, supply DCB information for data sets, add data sets, or specify particular volumes for data sets by using overriding DD statements.

Note: A temporary name assigned to a data set should be preceded by a double ampersand (&&). A single ampersand (&) denotes a symbolic parameter. Since the occurence of a single ampersand defines a symbolic parameter, two consective ampersands should be coded when a symbolic parameter is not being defined (e.g., a DSNAME value or a PARM field value). However, if a character string preceded by a single ampersand is encountered and no symbolic value has been assigned to it, the single ampersand is processed as if double ampersands had been coded. For further information on the use of single and double ampersands see the section on symbolic parameters in the publication IBM System/360 Operating System: Job Control Language.

APPENDIX F: DYNAMIC INVOCATION OF THE COMPILER

The PL/I (F) compiler can be invoked by using one of the assembler language macro instructions CALL, LINK, XCTL, or ATTACH. If the XCTL macro is used to invoke the compiler, then no user options may be stated. The compiler will use the standard default, as set during system generation, for each option.

If the compiler is invoked by CALL, LINK, or ATTACH, the user may supply:

The compiler options

The ddnames of the data sets to be used during processing

The first page number to be used on the output listing

Name	Operation	Operand
[symbol]	CALL	IEMAA,([optionlist][,[ddnamelist][,pagenbr]]), VL
[symbol]	LINK ATTACH	EP=IEMAA, PARAM=([optionlist][,[ddnamelist][,pagenbr]]), VL=1

EP - specifies the symbolic name of the compiler. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM - specifies, as a sublist, address parameters to be passed from the problem program to the compiler. The first word in the address parameter list contains the address of the option list. The second word contains the address of the ddname list. The third word contains the address of the page number parameter. If standard pagination is to be used, this parameter may be omitted.

optionlist - specifies the address of a variable length list containing the options. If no list is provided, the position must be marked by a comma.

The option list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form with each field separated by a comma. No blanks or zeros should appear in the list.

ddnamelist - specifies the address of a variable length list containing alternate ddnames for the data sets used during compiler processing. If standard ddnames are used and the page number operand is not used, then this operand may be omitted. If it is omitted and the page number operand is specified, a comma must be included to represent the omission. If the page number operand is specified, the address must be written whether or not standard ddnames are used.

The ddname list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If no ddnames are specified but the page number operand is used, then the count must be zero. Each name of less than eight bytes must be left-justified and padded with blanks. If an alternate ddname is omitted, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list. The sequence of the 8-byte entries in the ddname list is as follows:

Entry	Alternate Name for:
1	SYSLIN
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	not applicable
10	SYSUT3

pagenbr - specifies the address of a 6-byte fixed length list containing a number to be used as the initial page number on the output listing of the compiler. If standard pagination is to be used, then this operand may be omitted.

The page number list must begin on a halfword boundary, the halfword containing the value four, followed by four bytes containing the page number in binary.

VL - specifies that the sign bit is to be set to 1 in the last word of the address parameter list.

APPENDIX G : DIAGNOSTIC MESSAGES

SOURCE PROGRAM DIAGNOSTIC MESSAGES

All source program diagnostic messages produced are written in a group following the source program listing and any other listings specified as a parameter on the EXEC statement card.

Each message number is of the form IEMnnnnI, where the code IEM indicates the PL/I (F) Compiler, and nnnn the number of the message. The letter I is a system standard action code indicating an informative message for the programmer.

There are four types of diagnostic message: warning, error, severe error, and terminal error.

A Warning is a message that calls attention to a possible error, although the statement to which it refers is syntactically valid. In addition to alerting the programmer, it may assist him in writing more efficient programs in the future.

An Error message describes an attempt to correct an erroneous statement; the programmer is informed of the correction. Errors do not normally terminate processing of the text.

A Severe error message indicates an error which cannot be corrected by the compiler. The incorrect section of the program is deleted, but compilation is continued. Where reasonable, the ERROR condition will be raised at object time, if execution of an incorrect source statement is attempted. If a severe error occurs during compile-time processing, compilation will be terminated after the SOURCE listing has been produced.

A Terminal error message describes an error which, when discovered, forces the termination of the compilation.

The choice of the severity level at and above which diagnostic messages appear on the output is an option which may be selected by the programmer. FLAGW is assumed if no level is specified.

In the list of diagnostic messages below, the abbreviations W, E, S, and T, respectively, are used to indicate the severity of the message, and appear immediately before the number of the message. They do not appear in this way in the compiler output listings; instead, the messages are printed in separate groups according to severity.

In the following text, messages are followed where necessary by an explanation, a description of the action taken by the system, and the response required from the user. "Explanation" and "System Action" are given only when this information is not contained in the text of the message. When no "User Response" is stated explicitly, the user should assume that he must correct the error in his source program unless the action taken by the system makes it unnecessary for him to do so. However, even when system action successfully corrects an error, the user should remember that if he subsequently recompiles the same program, he will get the same diagnostic message again unless he has corrected the source error.

E IEM0002I INVALID PREFIX OPERATOR IN STATEMENT NUMBER xxx. REPLACED BY PLUS.

E IEM0003I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

E IEM0004I OPERATOR .NOT. IN STATEMENT NUMBER xxx USED AS AN INFIX OPERATOR. IT HAS BEEN REPLACED BY .NE.

E IEM0005I RIGHT PARENTHESIS INSERTED AFTER SINGLE PARENTHESIZED EXPRESSION IN STATEMENT NUMBER xxx

E IEM0006I RIGHT PARENTHESIS INSERTED AT END OF SUBSCRIPT, ARGUMENT OR CHECK LIST IN STATEMENT NUMBER xxx

S IEM0007I IDENTIFIER MISSING IN STATEMENT NUMBER xxx . A DUMMY IDENTIFIER HAS BEEN INSERTED.

E IEM0008I RIGHT PARENTHESIS INSERTED AT END OF CALL ARGUMENT LIST OR OTHER EXPRESSION LIST IN STATEMENT NUMBER xxx

W IEM0009I A LETTER IMMEDIATELY FOLLOWS CONSTANT IN STATEMENT NUMBER xxx. AN INTERVENING BLANK IS ASSUMED.

E IEM0010I IMPLEMENTATION RESTRICTION. IDENTIFIER yyyy IN OR NEAR STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN SHORTENED.

Explanation: Implementation restriction. Identifiers may not exceed 31 characters in length.

System Action: Identifier has been shortened by concatenating first 16 characters with last 15.

W IEM0011I CONSTANT IMMEDIATELY FOLLOWS IDENTIFIER IN STATEMENT NUMBER xxx. AN INTERVENING BLANK IS ASSUMED.

E IEM0012I EXPONENT MISSING IN FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . ZERO HAS BEEN INSERTED.

E IEM0013I INTEGER yyyy TOO LONG IN STATEMENT NUMBER xxx . IT HAS BEEN TRUNCATED ON THE RIGHT.

E IEM0014I EXPONENT TOO LONG IN FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . IT HAS BEEN TRUNCATED.

E IEM0015I SOLITARY DECIMAL POINT FOUND IN OPERAND POSITION IN STATEMENT NUMBER xxx . A FIXED-POINT ZERO HAS BEEN INSERTED.

E IEM0016I FLOATING-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED ON THE RIGHT.

E IEM0017I ZERO INSERTED IN FLOATING-POINT CONSTANT BEGINNING .E IN STATEMENT NUMBER xxx

E IEM0018I ZERO INSERTED IN PENCE FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0019I POUNDS FIELD IN STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

E IEM0020I ZERO INSERTED IN POUNDS FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0021I DECIMAL POINT IN EXPONENT FIELD OF CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx . FIELD TRUNCATED AT DECIMAL POINT.

E IEM0022I DECIMAL PENCE TRUNCATED IN STERLING CONSTANT BEGINNING yyyy STATEMENT NUMBER xxx

E IEM0023I LETTER L MISSING FROM STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

System Action: None

E IEM0024I SHILLINGS FIELD TRUNCATED IN STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0025I ZERO INSERTED IN SHILLINGS FIELD OF STERLING CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0026I ILLEGAL CHARACTER IN APPARENT BIT STRING yyyy IN STATEMENT NUMBER xxx . STRING TREATED AS A CHARACTER STRING.

E IEM0027I FIXED-POINT CONSTANT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED ON THE RIGHT.

S IEM0028I LABEL REFERENCED ON END STATEMENT NUMBER xxx CANNOT BE FOUND. END TREATED AS HAVING NO OPERAND.

S IEM0029I INVALID CHARACTER IN BINARY CONSTANT IN STATEMENT NUMBER xxx . CONSTANT TREATED AS DECIMAL CONSTANT.

S IEM0030I POINTER QUALIFIER FOLLOWS EITHER A SUBSCRIPT OR ANOTHER POINTER QUALIFIER IN STATEMENT NUMBER xxx.

System Action: As stated in a further message referring to the same statement.

S IEM0031I OPERAND MISSING IN OR FOLLOWING STATEMENT NUMBER xxx . DUMMY OPERAND INSERTED.

Explanation: Something invalid has been found in an expression, or where an expression was expected but not found. In order that further diagnosis can be made, the compiler has inserted a dummy operand. This may cause further error messages to appear for this statement.

T IEM0032I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG. COMPILATION TERMINATED AT THIS POINT.

User Response: Subdivide statement and recompile

E IEM0033I AN INVALID PICTURE CHARACTER IMMEDIATELY FOLLOWS TEXT yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THIS POINT.

W IEM0034I A LETTER IMMEDIATELY FOLLOWS A CONSTANT AT nnnn SEPARATE POSITION(S) IN STATEMENT NUMBER xxx. AN INTERVENING BLANK HAS BEEN ASSUMED IN EACH CASE.

User Response: Check that the system action will have the required effect

E IEM0035I LETTER F IS NOT FOLLOWED BY LEFT PARENTHESIS IN PICTURE IN STATEMENT NUMBER xxx. ONE HAS BEEN INSERTED.

E IEM0037I ZERO INSERTED IN SCALING FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx

E IEM0038I RIGHT PARENTHESIS INSERTED AFTER SCALING OR REPLICATION FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx

E IEM0039I NO CHARACTER FOLLOWS REPLICATION FACTOR IN PICTURE yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE REPLICATION FACTOR.

E IEM0040I A REPLICATION FACTOR OF 1 HAS BEEN INSERTED IN PICTURE yyyy IN STATEMENT NUMBER xxx

S IEM0043I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Explanation: Right parenthesis missing from length attached to character or bit string.

E IEM0044I IN STATEMENT NUMBER xxx PRECISION NOT AN INTEGER

Explanation: Precision should be an unsigned integer

System Action: The action taken depends on whether the precision is found in a DECLARE statement or a PROCEDURE statement. A further message will be produced.

E IEM0045I ZERO INSERTED IN FIXED PRECISION SPECIFICATION IN STATEMENT NUMBER xxx

E IEM0046I RIGHT PARENTHESIS INSERTED AFTER PRECISION SPECIFICATION IN STATEMENT NUMBER xxx

E IEM0048I RIGHT PARENTHESIS INSERTED IN FILE NAME LIST IN STATEMENT NUMBER xxx

E IEM0049I THE COMMENT FOLLOWING THE LOGICAL END OF PROGRAM HAS NOT BEEN TERMINATED.

Explanation: A /* was found following the logical end of the program and was interpreted as the start of a comment, but end-of-file was reached before the comment was terminated.

System Action: All text following the /* is read as a comment.

User Response: Check if this is a delimiter in the wrong column of the record.

S IEM0050I INVALID STATEMENT LABEL CONSTANT IN LABEL ATTRIBUTE IN STATEMENT NUMBER xxx. THE STATEMENT LABEL CONSTANT LIST HAS BEEN DELETED.

W IEM0051I MISSING RIGHT PARENTHESIS INSERTED FOLLOWING STATEMENT LABEL CONSTANT IN LABEL ATTRIBUTE IN STATEMENT NUMBER xxx

S IEM0052I INVALID ATTRIBUTE IN RETURNS ATTRIBUTE LIST IN STATEMENT NUMBER xxx. THE INVALID ATTRIBUTE HAS BEEN DELETED FROM THE LIST.

W IEM0053I SURPLUS COMMA HAS BEEN FOUND IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx. THIS COMMA HAS BEEN DELETED.

S IEM0054I ILLEGAL FORM OF CALL STATEMENT. STATEMENT NUMBER xxx DELETED.

W IEM0055I LABEL OR LABELS ON DECLARE STATEMENT NUMBER xxx HAVE BEEN IGNORED.

E IEM0056I NULL PICTURE FORMAT ITEM IN STATEMENT NUMBER xxx. THE CHARACTER 9 HAS BEEN INSERTED IN THE PICTURE.

Explanation: The null picture may be the result of the compiler truncating an invalid picture.

E IEM0057I INVALID CHARACTER FOLLOWING ITERATION FACTOR IN PICTURE BEGINNING yyyy IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE ITERATION FACTOR.

E IEM0058I ITERATION FACTOR IN PICTURE BEGINNING yyyy NOT AN UNSIGNED

INTEGER IN STATEMENT NUMBER xxx. THE PICTURE HAS BEEN TRUNCATED AT THE LEFT PARENTHESIS OF THE ITERATION FACTOR.

E IEM0059I MISSING RIGHT PARENTHESIS INSERTED IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx.

E IEM0060I POSITION MISSING IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx. POSITION OF 1 INSERTED.

E IEM0061I MISSING LEFT PARENTHESIS INSERTED IN POSITION ATTRIBUTE IN STATEMENT NUMBER xxx.

W IEM0062I THE ATTRIBUTE 'PACKED' IN DECLARATION STATEMENT NUMBER xxx IS NOW OBSOLETE, AND HAS BEEN IGNORED.

Explanation: PACKED has been removed from the language; the complementary attribute to ALIGNED is now UNALIGNED.

System Action: Since PACKED applied only to arrays and major structures, the new alignment defaults will be compatible with those of earlier versions of the compiler, except for bit string arrays that are not members of structures.

User Response: Correct source, and recompile if necessary.

E IEM0063I MISSING LEFT PARENTHESIS INSERTED IN RETURNS STATEMENT NUMBER xxx.

S IEM0064I ILLEGAL STATEMENT FOLLOWS THE THEN IN STATEMENT NUMBER xxx. SEMICOLON HAS BEEN INSERTED AFTER THE THEN.

S IEM0066I TEXT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error is detailed in another message referring to the same statement.

E IEM0067I EQUAL SYMBOL HAS BEEN INSERTED IN DO STATEMENT NUMBER xxx

T IEM0069I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS.

System Action: Compilation is terminated

User Response: Rewrite program with fewer blocks, or divide into more than one separate compilation.

T IEM0070I BEGIN STATEMENT NUMBER xxx IS NESTED BEYOND THE PERMITTED LEVEL. COMPILATION TERMINATED.

User Response: Reduce level of nesting of blocks to 50 or less.

T IEM0071I TOO MANY PROCEDURE, BEGIN, ITERATIVE DO, ON STATEMENTS IN THIS PROGRAM. COMPILATION TERMINATED.

Explanation: There is an implementation restriction on the number of blocks in a compilation. Refer to Appendix B of this publication for details.

User Response: Subdivide program into two or more compilations.

S IEM0072I DO STATEMENT NUMBER xxx REPLACED BY BEGIN STATEMENT.

E IEM0074I THEN INSERTED IN IF STATEMENT NUMBER xxx

S IEM0075I NO STATEMENT FOLLOWS THEN IN IF STATEMENT NUMBER xxx

S IEM0076I NO STATEMENT FOLLOWS ELSE IN OR FOLLOWING STATEMENT NUMBER xxx

S IEM0077I ELSE DELETED IN OR FOLLOWING STATEMENT NUMBER xxx

E IEM0078I IMPLEMENTATION RESTRICTION. TOO MANY CHARACTERS IN INITIAL LABEL ON STATEMENT NUMBER xxx. LABEL IGNORED.

Explanation: There is an implementation restriction on the number of characters in the subscript of a subscripted identifier. The maximum permissible number is 225.

E IEM0080I EQUAL SYMBOL HAS BEEN INSERTED IN ASSIGNMENT STATEMENT NUMBER xxx

S IEM0081I LABELS OR PREFIX OPTIONS BEFORE ELSE TRANSFERRED TO STATEMENT NUMBER xxx

Explanation: Labels or prefix options illegal before ELSE and therefore transferred to following statement.

S IEM0082I OPERAND MISSING IN CHECK LIST IN STATEMENT NUMBER xxx. DUMMY INSERTED.

S IEM0083I ON-CONDITION INVALID OR MISSING IN STATEMENT NUMBER xxx. ON ERROR HAS BEEN ASSUMED.

System Action: ON ERROR inserted in place of invalid condition

E IEM0084I THE I/O ON-CONDITION IN STATEMENT NUMBER xxx HAS NO FILENAME FOLLOWING IT. SYSIN IS ASSUMED.

E IEM0085I COLON MISSING AFTER PREFIX OPTION IN OR FOLLOWING STATEMENT NUMBER xxx. ONE HAS BEEN ASSUMED.

T IEM0090I THERE ARE NO COMPLETE STATEMENTS IN THIS PROGRAM. COMPILATION TERMINATED.

W IEM0094I RECORD IN OR FOLLOWING STATEMENT NUMBER xxx IS SHORTER THAN THE SPECIFIED SOURCE START. THE OUTPUT RECORD HAS BEEN MARKED WITH AN ASTERISK AND IGNORED.

E IEM0095I LABEL ON STATEMENT NUMBER xxx HAS NO COLON. ONE IS ASSUMED.

Explanation: The compiler has encountered an identifier which appears to be a statement label, but without a colon.

System Action: A colon is inserted

E IEM0096I SEMI-COLON NOT FOUND WHEN EXPECTED IN STATEMENT NUMBER xxx . ONE HAS BEEN INSERTED.

E IEM0097I INVALID CHARACTER HAS BEEN REPLACED BY BLANK IN OR FOLLOWING STATEMENT NUMBER xxx. THE CONTAINING OUTPUT RECORD IS MARKED BY AN ASTERISK.

S IEM0099I LOGICAL END OF PROGRAM OCCURS AT STATEMENT NUMBER xxx. THIS STATEMENT HAS BEEN IGNORED SO THAT SUBSEQUENT STATEMENTS MAY BE PROCESSED.

Explanation: Although the compiler has detected the end of the program, there is more text following it. The programmer appears to have made an error in matching END statements with PROCEDURE, BEGIN, DO or ON statements.

System Action: The END statement is ignored

S IEM0100I END OF FILE FOUND IN OR AFTER STATEMENT NUMBER xxx, BEFORE THE LOGICAL END OF PROGRAM.

System Action: If the statement is incomplete, it is deleted. Whether or not the statement is incomplete, the required number of END statements are added to the program so that compilation can continue.

User Response: Correct the source code. Possible causes of this error include:

1. Unmatched quote marks
2. Insufficient END statements
3. Omission of final semicolon.

S IEM0101I PARAMETER MISSING IN STATEMENT NUMBER xxx . A DUMMY HAS BEEN INSERTED.

S IEM0102I LABEL MISSING FROM PROCEDURE STATEMENT NUMBER xxx. A DUMMY LABEL HAS BEEN INSERTED.

S IEM0103I LABEL MISSING FROM ENTRY STATEMENT NUMBER xxx

S IEM0104I ILLEGAL STATEMENT FOLLOWS ELSE IN STATEMENT NUMBER xxx

System Action: Null statement inserted

S IEM0105I ILLEGAL STATEMENT FOLLOWS ON IN STATEMENT NUMBER xxx

System Action: Null statement inserted

T IEM0106I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS.

System Action: Compilation is terminated

User Response: Rewrite program with fewer blocks, or divide into more than one separate compilation.

T IEM0107I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG. THIS STATEMENT MAY CONTAIN UNMATCHED QUOTE MARKS.

User Response: Subdivide statement and recompile

S IEM0108I ENTRY STATEMENT NUMBER xxx IN AN ITERATIVE DO GROUP HAS BEEN DELETED.

S IEM0109I TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error is detailed in another message referring to the same statement.

S IEM0110I TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error is detailed in another message referring to the same statement.

S IEM0111I FIRST STATEMENT NOT A PROCEDURE STATEMENT. A DUMMY PROCEDURE STATEMENT HAS BEEN INSERTED.

S IEM0112I ENTRY STATEMENT NUMBER xxx IN BEGIN BLOCK HAS BEEN DELETED.

S IEM0113I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

Explanation: Parenthesized list in ON statement is either not closed or contains an error and has been truncated.

E IEM0114I RIGHT PARENTHESIS INSERTED IN PREFIX OPTION IN OR FOLLOWING STATEMENT NUMBER xxx

E IEM0115I LEFT PARENTHESIS INSERTED AFTER WHILE IN STATEMENT NUMBER xxx

E IEM0116I PREFIX OPTION FOLLOWS LABEL IN STATEMENT NUMBER xxx. PREFIX OPTION IS IGNORED.

S IEM0118I OFFSET ATTRIBUTE NOT FOLLOWED BY PARENTHESIZED BASED VARIABLE IN STATEMENT NUMBER xxx. THE ATTRIBUTE IS IGNORED.

S IEM0128I LENGTH OF BIT OR CHARACTER STRING MISSING IN STATEMENT NUMBER xxx. LENGTH 1 INSERTED.

S IEM0129I INVALID WAIT STATEMENT NUMBER xxx DELETED.

E IEM0130I OPERAND MISSING. COMMA DELETED IN WAIT STATEMENT NUMBER xxx

S IEM0131I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0132I DUMMY OPERAND INSERTED IN STATEMENT NUMBER xxx

S IEM0133I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0134I IMPLEMENTATION RESTRICTION. TOO MANY LEVELS OF REPLICATION IN INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx. THE ATTRIBUTE HAS BEEN DELETED.

Explanation: The implementation restriction on levels of nesting has been contravened. For details, refer to Appendix B of this publication.

User Response: Rewrite INITIAL attribute with lower level of replication

E IEM0136I 'IN' CLAUSE IN STATEMENT NUMBER xxx HAS NO ASSOCIATED 'SET' CLAUSE.

Explanation: An IN clause must be accompanied by a SET clause in the same statement.

System Action: The IN clause is ignored

E IEM0138I SOLITARY I FOUND WHERE A CONSTANT IS EXPECTED IN INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx. FIXED DECIMAL IMAGINARY 1I HAS BEEN ASSUMED.

Explanation: The programmer has initialized an element using the variable I where the constant 1I was expected.

System Action: 1I is assumed

S IEM0139I TEXT IMMEDIATELY FOLLOWING yyyy IN INITIAL ATTRIBUTE IS ILLEGAL. INITIAL ATTRIBUTE DELETED IN STATEMENT NUMBER xxx

Explanation: A language feature has been used that is not supported by this version of the compiler. For details, refer to Appendix H of this publication. Although the message states that the error follows the quoted text, the quoted text may itself be invalid, and the compiler may have attempted to correct the source error. In this case, there will usually be another diagnostic message associated with the statement.

W IEM0140I NO IDENTIFIER APPEARS IN DECLARE STATEMENT NUMBER xxx

System Action: The statement is ignored

S IEM0142I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0143I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0144I RETURNS ATTRIBUTE IS NOT FOLLOWED BY A DATA DESCRIPTION IN STATEMENT NUMBER xxx. THE RETURNS ATTRIBUTE HAS BEEN DELETED.

S IEM0145I DUMMY IDENTIFIER INSERTED IN GENERIC ATTRIBUTE LIST IN STATEMENT NUMBER xxx

S IEM0146I RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0147I THE USE OF REFER IN STATEMENT NUMBER xxx IS EITHER INVALID OR IS NOT IMPLEMENTED IN THIS RELEASE

Explanation: The implementation of the REFER option is restricted; see Appendix H, 'Language features Not Supported'.

System Action: Ignore the REFER clause. A further message identifying the invalid text will usually accompany this message.

E IEM0148I LEFT PARENTHESIS MISSING IN STATEMENT NUMBER xxx

System Action: See further messages relating to this statement

E IEM0149I COMMA HAS BEEN DELETED FROM LIST IN STATEMENT NUMBER xxx

E IEM0150I STATEMENT NUMBER xxx IS AN INVALID FREE STATEMENT. THE STATEMENT HAS BEEN DELETED.

Explanation: The format of the statement is invalid

S IEM0151I SEMI-COLON INSERTED IN STATEMENT NUMBER xxx

S IEM0152I TEXT BEGINNING yyyy IN STATEMENT NUMBER xxx HAS BEEN DELETED.

Explanation: The source error is detailed in another message referring to the same statement.

E IEM0153I THE ATTRIBUTED BASED HAS BEEN ASSUMED IN STATEMENT NUMBER xxx WHERE CONTROLLED WAS SPECIFIED.

Explanation: The PL/I feature CONTROLLED (pointer) has been changed to BASED (pointer).

S IEM0154I IMPLEMENTATION RESTRICTION IN STATEMENT NUMBER xxx. BASED MUST BE FOLLOWED BY AN IDENTIFIER IN PARENTHESIS.

System Action: Text is deleted. See further error message for this statement.

User Response: Correct source statement

E IEM0158I ZERO STRUCTURE LEVEL NUMBER DELETED IN DECLARE STATEMENT NUMBER xxx

Explanation: Zero level number not allowed

E IEM0159I SIGN DELETED PRECEDING STRUCTURE LEVEL NUMBER IN DECLARE STATEMENT NUMBER xxx

Explanation: The level number must be an unsigned integer

S IEM0163I FORMAT LIST MISSING, (A) INSERTED IN STATEMENT NUMBER xxx

E IEM0164I MISSING RIGHT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

S IEM0166I OPERAND MISSING IN GO TO STATEMENT NUMBER xxx. DUMMY IS INSERTED.

E IEM0172I LEFT PARENTHESIS INSERTED IN DELAY STATEMENT NUMBER xxx

Explanation: The expression in a DELAY statement should be contained in parentheses

E IEM0180I EQUAL SYMBOL HAS BEEN INSERTED IN DO SPECIFICATIONS IN STATEMENT NUMBER xxx

E IEM0181I SEMICOLON INSERTED IN STATEMENT NUMBER xxx

Explanation: An error has been discovered. A semi-colon is therefore inserted and the rest of the statement is skipped.

S IEM0182I TEXT BEGINNING yyyy SKIPPED IN OR FOLLOWING STATEMENT NUMBER xxx

Explanation: The source error is detailed in another message referring to the same statement.

S IEM0185I OPTION IN GET/PUT STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

S IEM0187I DATA LIST MISSING IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0191I DUMMY OPERAND INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

E IEM0193I RIGHT PARENTHESIS INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

E IEM0194I MISSING RIGHT PARENTHESIS INSERTED IN FORMAT LIST IN STATEMENT NUMBER xxx

S IEM0195I INVALID FORMAT LIST DELETED IN STATEMENT NUMBER xxx. (A) INSERTED.

S IEM0198I COMPLEX FORMAT ITEM yyyy IN STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

S IEM0202I DEFERRED FEATURE. STATEMENT NUMBER xxx NOT IMPLEMENTED IN THIS VERSION.

Explanation: The statement referred to is of a type not supported by this version of the compiler. For details, refer to Appendix H of this publication.

System Action: Compilation continues

User Response: Rewrite source program avoiding use of unsupported feature

E IEM0207I COMMA REPLACED BY EQUAL SYMBOL IN ASSIGNMENT STATEMENT NUMBER xxx

E IEM0208I LEFT PARENTHESIS INSERTED IN CHECK LIST IN STATEMENT NUMBER xxx

T IEM0209I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO COMPLEX

Explanation: The level of nesting exceeds the implementation restriction. Refer to Appendix B of this publication for details.

System Action: Terminates compilation

User Response: Divide statement into two or more statements

E IEM0211I LEFT PARENTHESIS INSERTED IN STATEMENT NUMBER xxx

E IEM0212I MULTIPLE TASK OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first

E IEM0213I MULTIPLE EVENT OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first

E IEM0214I MULTIPLE PRIORITY OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: Ignores options other than the first

E IEM0216I INVALID EVENT OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM0217I INVALID PRIORITY OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM0220I IDENTIFIER MISSING OR INCORRECT AFTER OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0221I NUMBER OF LINES NOT GIVEN AFTER LINE OPTION IN STATEMENT NUMBER xxx. (1) INSERTED.

S IEM0222I DEFERRED FEATURE. THE IDENT OPTION ON OPEN/CLOSE STATEMENT NUMBER xxx IS NOT IMPLEMENTED BY THIS VERSION.

Explanation: A language feature has been used that is not supported by this version of the compiler. Refer to Appendix H of this publication for details.

System Action: Option ignored

S IEM0223I EXPRESSION MISSING AFTER IDENT/TITLE/LINESIZE/PAGESIZE OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

Explanation: No left parenthesis found following keyword

S IEM0224I INVALID OPTION DELETED IN I/O STATEMENT NUMBER xxx

S IEM0225I OPTION AFTER OPEN/CLOSE IN STATEMENT NUMBER xxx IS INVALID OR MISSING.

S IEM0226I EXPRESSION MISSING AFTER FORMAT ITEM IN STATEMENT NUMBER xxx. ITEM DELETED.

W IEM0227I NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN ASSUMED IN EACH CASE

Explanation: One or more GET or PUT statements have appeared in the program with no specified FILE option or STRING option.

System Action: The compiler has assumed the appropriate default file (SYSIN for GET, SYSPRINT for PUT).

S IEM0228I EXPRESSION MISSING AFTER OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0229I FORMAT ITEM IN STATEMENT NUMBER xxx IS INVALID AND HAS BEEN DELETED.

S IEM0230I INVALID DATA LIST IN STATEMENT NUMBER xxx. STATEMENT DELETED.

E IEM0231I MISSING COMMA INSERTED IN DATA LIST IN STATEMENT NUMBER xxx

Explanation: Comma missing between elements of a data list

E IEM0232I KEYWORD DO MISSING IN DATA LIST IN STATEMENT NUMBER xxx. DO IS INSERTED.

S IEM0233I RETURN STATEMENT NUMBER xxx IS WITHIN AN ON-UNIT . IT IS REPLACED BY A NULL STATEMENT.

S IEM0235I ARGUMENT OMITTED FOLLOWING yyyy OPTION IN STATEMENT NUMBER xxx. OPTION DELETED.

S IEM0236I THE OPTION yyyy IN STATEMENT NUMBER xxx IS UNSUPPORTED OR INVALID.

S IEM0237I INSUFFICIENT OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE STATEMENT HAS BEEN REPLACED BY A NULL STATEMENT.

S IEM0238I THE LOCATE-VARIABLE IN LOCATE STATEMENT NUMBER xxx IS OMITTED OR SUBSCRIPTED. THE STATEMENT HAS BEEN DELETED.

Explanation: The omission of the locate variable renders the statement meaningless. Subscripted locate variables are invalid.

System Action: Replaces invalid statement with a null statement.

T IEM0240I COMPILER ERROR IN PHASE CV. SCAN CANNOT IDENTIFY DICTIONARY ENTRY.

Explanation: The main scan of fifth pass of read-in has found something in the dictionary which it cannot recognize

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0241I MULTIPLE USE OF A PREFIX OPTION HAS OCCURRED IN STATEMENT NUMBER xxx. THE LAST NAMED OPTION IS USED.

S IEM0242I INVALID PREFIX OPTION IN STATEMENT NUMBER xxx. THE OPTION HAS BEEN IGNORED.

T IEM0243I COMPILER ERROR. PHASE CS HAS FOUND AN UNMATCHED END.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0244I CHECK PREFIX OPTION IN STATEMENT NUMBER xxx IS NOT FOLLOWED BY A PARENTHESIZED LIST. THE OPTION HAS BEEN IGNORED.

E IEM0245I A CHECK PREFIX OPTION IS GIVEN FOR STATEMENT NUMBER xxx WHICH IS NOT A PROCEDURE OR BEGIN. THE OPTION HAS BEEN IGNORED.

S IEM0247I ALL SUBSCRIPTED LABELS PREFIXING PROCEDURE OR ENTRY STATEMENT NUMBER xxx HAVE BEEN IGNORED.

Explanation: Subscripted labels may not be used as pre-

fixes on PROCEDURE or ENTRY statements.

T IEM0254I COMPILER UNABLE TO RECOVER FROM I/O ERROR - PLEASE RETRY JOB.

System Action: Terminates compilation

User Response: Re-attempt compilation

T IEM0255I THERE ARE NO COMPLETE STATEMENTS IN THIS PROGRAM

Explanation: Compiler cannot reconcile END statements with stack entries. Usually caused by a program containing only comments.

System Action: Compilation is terminated

User Response: Check source for completed statements. If these are present then save relevant data and call your local IBM representative.

W IEM0510I THE TASK OPTION HAS BEEN ASSUMED TO APPLY TO THE EXTERNAL PROCEDURE STATEMENT NUMBER xxx

Explanation: TASK, EVENT or PRIORITY options have been detected in a CALL statement, but the TASK option has not been specified in the external procedure.

System Action: The TASK option is correctly applied

W IEM0511I OPTIONS MAIN AND/OR TASK ARE NOT ALLOWED ON THE INTERNAL PROCEDURE STATEMENT NUMBER xxx

System Action: The invalid options are ignored

S IEM0512I IDENTIFIER yyyy IN STATEMENT NUMBER xxx IN INITIAL ATTRIBUTE LIST IS NOT A KNOWN LABEL CONSTANT AND HAS BEEN IGNORED.

System Action: Identifier changed to * in the list.

S IEM0513I REPEATED LABEL IN SAME BLOCK ON STATEMENT NUMBER xxx. LABEL DELETED.

Explanation: A label may not be used more than once in the same block.

S IEM0514I PARAMETER yyyy IN STATEMENT NUMBER xxx IS SAME AS LABEL. PARAMETER REPLACED BY DUMMY.

S IEM0515I IMPLEMENTATION RESTRICTION. CHARACTER STRING LENGTH IN STATEMENT NUMBER xxx REDUCED TO 32,767.

S IEM0516I ILLEGAL OPTIONS LIST ON STATEMENT NUMBER xxx. LIST IGNORED.

System Action: Compiler scans for next right bracket. If this is not the bracket closing the illegal options list, a compiler error will probably follow.

S IEM0517I CONFLICTING ATTRIBUTE DELETED IN STATEMENT NUMBER xxx

S IEM0518I IMPLEMENTATION RESTRICTION. PRECISION TOO LARGE IN STATEMENT NUMBER xxx . DEFAULT PRECISION GIVEN.

Explanation: If later a valid precision is given, this will be accepted in place of the default.

System Action: Attribute ignored. Attribute test mask restored so that later attribute will not be found to conflict with deleted one.

S IEM0519I ILLEGAL ATTRIBUTE ON STATEMENT NUMBER xxx IGNORED.

Explanation: Only data attributes allowed on procedure or entry statements. (No dimensions allowed).

T IEM0520I COMPILER ERROR CODE nnnn

Explanation: A compiler error has occurred.

System Action: Terminates immediately

User Response: Save relevant data. Call your local IBM representative.

S IEM0521I INVALID STRING LENGTH IN STATEMENT NUMBER xxx. LENGTH OF 1 ASSUMED.

Explanation: Either no length has been given or string length * has been used in source code.

System Action: Assumes length

of 1 and skips to next attribute

S IEM0522I IMPLEMENTATION RESTRICTION. NUMBER OF PARAMETERS IN PROCEDURE OR ENTRY STATEMENT NUMBER xxx TRUNCATED TO 64.

S IEM0523I PARAMETER zzzz IN STATEMENT NUMBER xxx APPEARS TWICE. SECOND ONE REPLACED BY DUMMY.

S IEM0524I IDENTIFIER yyyy IN LABEL LIST IN STATEMENT NUMBER xxx IS NOT A LABEL OR IS NOT KNOWN.

System Action: Ignores identifier

T IEM0525I IMPLEMENTATION RESTRICTION. MORE THAN 73 PAIRS OF FACTORED ATTRIBUTE BRACKETS.

Explanation: There is an implementation restriction limiting the number of left parentheses used for factoring attributes in DECLARE statements to 73 within a compilation.

System Action: Terminates compilation

User Response: Reduce factoring by expanding declarations.

W IEM526I OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE STATEMENT NUMBER xxx

S IEM0527I IMPLEMENTATION RESTRICTION. ARRAY BOUND IN STATEMENT NUMBER xxx IS TOO LARGE AND HAS BEEN REPLACED BY THE MAXIMUM PERMITTED VALUE (32767 OR -32768).

T IEM0528I COMPILER ERROR CODE nnnn IN STATEMENT NUMBER xxx

Explanation: Compiler error found in processing a DECLARE statement

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

S IEM0529I IMPLEMENTATION RESTRICTION. STRUCTURE LEVEL NUMBER IN STATEMENT NUMBER xxx REDUCED TO 255.

S IEM0530I IMPLEMENTATION RESTRICTION. TOO MANY LABELS IN LABEL LIST IN STATEMENT NUMBER xxx. THE LABEL zzzz AND ANY FOLLOWING IT HAVE BEEN IGNORED.

Explanation: There is an implementation restriction limiting the number of label constants following the LABEL attribute to 125.

S IEM0532I ILLEGAL ASTERISK AS SUBSCRIPT IN DEFINING LIST IN STATEMENT NUMBER xxx . LIST TRUNCATED.

System Action: Compilation continues with truncated iSUB list, possibly causing cascade errors.

S IEM0533I IMPLEMENTATION RESTRICTION. I-SUB VALUE IN STATEMENT NUMBER xxx TOO LARGE. REDUCED TO 32.

Explanation: There is an implementation restriction limiting the number of dimensions to a maximum of 32.

S IEM0534I IMPLEMENTATION RESTRICTION. STRING LENGTH IN STATEMENT NUMBER xxx REDUCED TO 32,767.

S IEM0535I IMPLEMENTATION RESTRICTION. KEYLENGTH IN STATEMENT NUMBER xxx REDUCED TO 255.

S IEM0536I IDENTIFIER yyyy IN STATEMENT NUMBER xxx IS NOT A LABEL CONSTANT OR IS NOT KNOWN. IT IS IGNORED.

Explanation: Identifiers following the LABEL attribute must be LABEL constants and must be known.

S IEM0537I IMPLEMENTATION RESTRICTION. POSITION CONSTANT IN STATEMENT NUMBER xxx REDUCED TO 32,767.

E IEM0538I IMPLEMENTATION RESTRICTION. PRECISION SPECIFICATION IN STATEMENT NUMBER xxx TOO LARGE. DEFAULT PRECISION GIVEN.

E IEM0539I ILLEGAL NEGATIVE PRECISION IN STATEMENT NUMBER xxx . DEFAULT PRECISION GIVEN.

S IEM0540I * BOUNDS ARE MIXED WITH NON-* BOUNDS IN DECLARE STATEMENT NUMBER xxx . ALL THE BOUNDS ARE MADE *.

E IEM0541I LOWER BOUND GREATER THAN UPPER BOUND IN DECLARE OR ALLOCATE STATEMENT NUMBER xxx . THE BOUNDS ARE INTERCHANGED.

S IEM0542I IMPLEMENTATION RESTRICTION. NUMBER OF DIMENSIONS DECLARED TRUNCATED TO 32 IN STATEMENT NUMBER xxx

T IEM0543I COMPILER ERROR. ILLEGAL STATEMENT FOUND IN THE DECLARE CHAIN.

Explanation: Compiler error found in scanning chain of DECLARE statements

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0544I COMPILER ERROR. INITIAL CODE BYTE OF DECLARE STATEMENT IS NEITHER STATEMENT NUMBER NOR STATEMENT LABEL.

Explanation: Compiler error found in first byte of DECLARE statements

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0545I COMPILER ERROR. ILLEGAL INITIAL CHARACTER TO DECLARED ITEM IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scanning start of declared item

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0546I COMPILER ERROR. ILLEGAL CHARACTER FOUND AFTER LEVEL NUMBER IN DECLARE STATEMENT NUMBER xxx

Explanation: Compiler error found after structure level number in DECLARE statement

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

W IEM0547I THE IDENTIFIER yyyy DECLARED IN STATEMENT NUMBER xxx IS A NON-MAJOR STRUCTURE MEMBER AND HAS THE SAME NAME AS A PARAMETER IN THE SAME BLOCK. THESE ARE DIFFERENT IDENTIFIERS, AND UNQUALIFIED REFERENCES REFER TO THE PARAMETER.

System Action: Same BCD treated as different identifiers

T IEM0548I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN DECLARATION LIST.

Explanation: Compiler error found in list of declarations in DECLARE statement

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0549I THE DECLARED LEVEL OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx SHOULD BE ONE. THIS HAS BEEN FORCED.

System Action: Illegal level number treated as 1

S IEM0550I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH A TRUE LEVEL NUMBER GREATER THAN THE IMPLEMENTATION RESTRICTION OF 63. THE DECLARATION OF THE IDENTIFIER IS IGNORED.

E IEM0551I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH ZERO PRECISION. THE DEFAULT VALUE HAS BEEN ASSUMED.

T IEM0552I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN FACTORED ATTRIBUTE LIST IN DECLARE STATEMENT NUMBER xxx

Explanation: Compiler error found in factored attribute list

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0553I THE IDENTIFIER yyyy HAS HAD A CONFLICTING ATTRIBUTE IGNORED IN DECLARE STATEMENT NUMBER xxx

Explanation: The two attributes may conflict as a result

of a feature not supported by this version of the compiler. For details of these features, refer to Appendix H of this publication.

T IEM0554I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN PARAMETER LIST FOLLOWING 'GENERIC' ATTRIBUTE.

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0555I STORAGE CLASS ATTRIBUTES MAY NOT BE SPECIFIED FOR STRUCTURE MEMBER yyyy . ATTRIBUTE IGNORED.

User Response: Delete illegal storage class attribute for the structure member.

T IEM0556I COMPILER ERROR. ILLEGAL CHARACTER FOUND IN PARAMETER LIST FOLLOWING AN 'ENTRY' ATTRIBUTE IN DECLARE STATEMENT NUMBER xxx

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0557I THE MULTIPLE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

S IEM0558I IMPLEMENTATION RESTRICTION. NUMBER OF PARAMETER DESCRIPTIONS DECLARED FOR PROCEDURE OR ENTRY NAME yyyy IN STATEMENT NUMBER xxx TRUNCATED TO 64.

E IEM0559I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH CONFLICTING FACTORED LEVEL NUMBERS. THE ONE AT DEEPEST FACTORING LEVEL HAS BEEN CHOSEN.

E IEM0560I IN STATEMENT NUMBER xxx A CONFLICTING ATTRIBUTE HAS BEEN IGNORED IN THE DECLARATION OF THE RETURNED VALUE OF ENTRY POINT yyyy

S IEM0561I IN STATEMENT NUMBER xxx THE IDENTIFIER yyyy IS A MULTIPLE DECLARATION OF AN INTERNAL ENTRY LABEL. THIS DECLARATION IS IGNORED.

S IEM0562I THE IDENTIFIER yyyy IS DECLARED IN STATEMENT NUMBER xxx AS AN INTERNAL ENTRY POINT. THE NUMBER OF PARAMETERS DECLARED IS DIFFERENT FROM THE NUMBER GIVEN AT THE ENTRY POINT.

S IEM0563I THE IDENTIFIER yyyy DECLARED 'BUILTIN' IN STATEMENT NUMBER xxx IS NOT A BUILT-IN FUNCTION. DECLARATION IGNORED.

E IEM0564I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH PRECISION GREATER THAN THE IMPLEMENTATION LIMITS. THE MAXIMUM VALUE HAS BEEN TAKEN.

E IEM0565I THE IDENTIFIER yyyy IS DECLARED IN STATEMENT NUMBER xxx AS A MEMBER OF A GENERIC LIST, BUT ITS ATTRIBUTES DO NOT MAKE IT AN ENTRY POINT. THE DECLARATION OF THE IDENTIFIER HAS BEEN IGNORED.

E IEM0566I ONE OF THE PARAMETERS DECLARED FOR ENTRY POINT yyyy IN STATEMENT NUMBER xxx SHOULD BE AT LEVEL ONE. THIS HAS BEEN FORCED.

W IEM0567I IF FUNCTION zzzz IN STATEMENT NUMBER xxx IS INVOKED,THE USE OF DEFAULT ATTRIBUTES FOR THE VALUE RETURNED WILL CONFLICT WITH THE DECLARED ATTRIBUTES FOR THAT VALUE.

Explanation: The data type to which a result will be converted at a RETURN (expression) will not be the same as that expected at an invocation of the entry label as a function.

System Action: None

User Response: Write an ENTRY declaration in the containing PROCEDURE or BEGIN block, giving the same data attributes as on the PROCEDURE or ENTRY statement.

S IEM0568I THE IDENTIFIER zzzz IS CALLED BUT IS EITHER A BUILTIN FUNCTION OR IS NOT AN ENTRY POINT.

System Action: The erroneous statement is deleted.

T IEM0569I COMPILER ERROR NUMBER nnnn IN MODULE EP.

Explanation: Compiler error found in scan of chain of CALL statements

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

W IEM0570I THE ENTRY POINT yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx TO HAVE A RETURNED VALUE DIFFERENT FROM THAT GIVEN ON THE PROCEDURE OR ENTRY STATEMENT.

System Action: None

User Response: Change the declaration, or the PROCEDURE or ENTRY statement.

S IEM0571I IMPLEMENTATION RESTRICTION. IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS MORE THAN 32 DIMENSIONS. DIMENSION ATTRIBUTE IGNORED.

S IEM0572I THE IDENTIFIER yyyy HAS BEEN DECLARED IN STATEMENT NUMBER xxx WITH THE ATTRIBUTE "NORMAL" OR "ABNORMAL". THE APPLICATION OF THIS ATTRIBUTE IS AN UNSUPPORTED FEATURE OF THE FOURTH VERSION, AND IT HAS BEEN IGNORED.

Explanation: A language feature has been used which is not supported by this version of the compiler. Refer to Appendix H of this publication for details.

S IEM0573I THE SELECTION OF GENERIC FAMILY MEMBERS WHOSE PARAMETERS HAVE A STRUCTURE DESCRIPTION IS DEFERRED. ENTRY NAME yyyy, DECLARED IN STATEMENT NUMBER xxx, IS SUCH A MEMBER AND HAS BEEN DELETED.

Explanation: The usage referred to is not supported by this version of the compiler. For details, refer to Appendix H of this publication.

T IEM0574I THE MULTIPLE DECLARATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx HAS BEEN IGNORED.

S IEM0589I COMPILER ERROR. ITEM zzzz IN LIKE CHAIN IS NOT A STRUCTURE. ITEM IS IGNORED.

User Response: Save relevant data. Call your local IBM representative.

S IEM0590I STRUCTURE ELEMENT zzzz WHICH HAS LIKE ATTRIBUTE ATTACHED TO IT, IS FOLLOWED BY AN ELEMENT WITH A NUMERICALLY GREATER STRUCTURE LEVEL NUMBER. LIKE ATTRIBUTE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0591I STRUCTURE ELEMENT zzzz IS LIKENED TO AN ITEM WHICH IS NOT A STRUCTURE VARIABLE. LIKE ATTRIBUTE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0592I STRUCTURE ELEMENT zzzz IS LIKENED TO A STRUCTURE WHICH CONTAINS ELEMENTS WHICH HAVE ALSO BEEN DECLARED WITH THE LIKE ATTRIBUTE. LIKE ATTRIBUTE ON ORIGINAL STRUCTURE IS IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0593I STRUCTURE NAME TO WHICH zzzz IS LIKENED IS NOT KNOWN. LIKE ATTRIBUTE IGNORED.

System Action: Self explanatory: may result in cascade errors.

E IEM0594I AMBIGUOUS QUALIFIED NAME yyyy USED AS A BASE IDENTIFIER. MOST RECENT DECLARATION USED.

E IEM0595I QUALIFIED NAME yyyy USED AS A BASE IDENTIFIER CONTAINS MORE THAN ONE IDENTIFIER AT THE SAME STRUCTURE LEVEL.

System Action: The erroneous statement is deleted.

S IEM0596I MAJOR STRUCTURE yyyy HAS BEEN LIKENED TO AN ITEM WHICH IS NOT A VALID STRUCTURE. DECLARATION OF STRUCTURE IGNORED.

System Action: Self explanatory: may result in cascade errors.

S IEM0597I IDENTIFIER zzzz WHICH IS NOT A FORMAL PARAMETER OR OF STORAGE CLASS CONTROLLED HAS BEEN LIKENED TO A STRUCTURE CONTAINING * DIMENSIONS OR LENGTH. * DIMENSIONS OR LENGTH HAVE BEEN IGNORED IN THE CONSTRUCTED STRUCTURE.

System Action: Self explanatory: may result in cascade errors from later phases.

S IEM0598I QUALIFIED NAME TO WHICH zzzz HAS BEEN LIKENED IS AN AMBIGUOUS REFERENCE. LIKE ATTRIBUTE HAS BEEN IGNORED.

S IEM0599I zzzz WHICH IS A PARAMETER OR A BASED VARIABLE, HAS BEEN DECLARED (USING THE LIKE ATTRIBUTE) AS A STRUCTURE WITH THE INITIAL ATTRIBUTE. THE INITIAL ATTRIBUTE IS INVALID AND HAS BEEN IGNORED.

User Response: Declare the parameter or based variable with the LIKE attribute specifying a structure without the INITIAL attribute.

S IEM0600I STATIC STRUCTURE zzzz HAS BEEN DECLARED BY MEANS OF THE LIKE ATTRIBUTE TO HAVE ADJUSTABLE EXTENTS. THE EXTENTS HAVE BEEN IGNORED.

Explanation: A STATIC variable cannot have adjustable extents

System Action: All bounds on the offending variable are set to zero

S IEM0601I OFFSET ATTRIBUTE ON PROCEDURE STATEMENT NUMBER xxx IS NOT BASED ON A BASED AREA. IT HAS BEEN CHANGED TO POINTER.

T IEM0602I IDENTIFIER IN BASED ATTRIBUTE ON zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT A NON-BASED POINTER

System Action: Compilation is terminated

T IEM0603I INVALID POINTER EXPRESSION IN BASED ATTRIBUTE ON zzzz IN STATEMENT NUMBER xxx

Explanation: The pointer associated with the based variable does not obey the implementation rules (e.g., it may be subscripted).

System Action: The compilation is terminated

T IEM0604I LENGTH DECLARED FOR BASED STRING zzzz IN STATEMENT NUMBER xxx IS INVALID

Explanation: The declaration violates the compiler implementation rules. (See Appendix B, 'Implementation Conventions and Restrictions').

System Action: Terminates compilation

T IEM0605I BOUNDS DECLARED FOR BASED ARRAY zzzz IN STATEMENT NUMBER xxx ARE INVALID

Explanation: The adjustable bounds declared are outside those permitted by this implementation.

System Action: Terminates compilation

S IEM0606I OFFSET VARIABLE zzzz HAS BEEN DECLARED IN STATEMENT NUMBER xxx RELATIVE TO AN IDENTIFIER WHICH IS NOT A BASED AREA. IT HAS BEEN CHANGED TO A POINTER VARIABLE.

System Action: The offset is changed to a pointer to prevent the compiler from producing further error messages.

W IEM0607I IF THE BASE OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx IS ALLOCATED WITH THE DECLARED BOUNDS THE DEFINING WILL BE IN ERROR.

Explanation: For correspondence defining not involving iSUB's, the bounds of the defined array must be a subset of the bounds of the base. In this case the bounds declared for the base do not satisfy this requirement. However, the base is of CONTROLLED storage class and if it is allocated with different bounds the defining may be legal.

System Action: Nothing further

T IEM0608I ILLEGAL DEFINING IN STATEMENT NUMBER xxx. BASE IDENTIFIER zzzz IS A MEMBER OF A DIMENSIONED STRUCTURE.

Explanation: In the case of string class overlay defining where the base is an array, it is an error if it is a member of an array of structures.

System Action: The compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The

DEFINED Attribute" - and correct error.

T IEM0609I DEFERRED FEATURE. DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx WITH A SUBSCRIPTED BASE.

Explanation: Overlay defining on a subscripted base is not supported by this version of the compiler.

System Action: The compilation is terminated.

User Response: Replace all references to the defined item by appropriate subscripted references to the base.

T IEM0610I DEFERRED FEATURE. DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ON A BASE OF CONTROLLED STORAGE CLASS.

Explanation: If the base is declared CONTROLLED, neither overlay defining nor correspondence defining is supported by this release of the compiler.

System Action: The compilation is terminated.

User Response: Replace all references to the defined item by appropriate references to the base.

T IEM0611I SCALAR zzzz DECLARED IN STATEMENT NUMBER xxx IS ILLEGALLY DEFINED WITH ISUBS.

Explanation: Only arrays may be correspondence defined using iSUB notation.

System Action: The compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct error.

E IEM0612I INITIAL ATTRIBUTE DECLARED FOR DEFINED ITEM zzzz IN STATEMENT NUMBER xxx WILL BE IGNORED.

Explanation: DEFINED items may not have the INITIAL attribute.

System Action: INITIAL attribute ignored

T IEM0623I THE BASE SUBSCRIPT LIST USED WITH THE DEFINED VARIABLE zzzz IN STATEMENT NUMBER xxx ILLEGALLY REFERS TO OR IS DEPENDENT ON THE DEFINED VARIABLE.

Explanation: It is illegal for a base subscript list in the DEFINED attribute to refer directly, or via any further level of defining, to the defined item.

System Action: The compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct error.

T IEM0624I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx IS ITSELF DEFINED.

Explanation: The base of DEFINED data may not itself be DEFINED

System Action: The compilation is terminated.

User Response: Replace the specified base by an appropriate reference to its base.

T IEM0625I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx HAS THE WRONG NUMBER OF SUBSCRIPTS.

Explanation: If the base reference in a DEFINED attribute is subscripted, it must have the same number of subscript expressions as the dimensionality of the base array.

System Action: The compilation is terminated.

User Response: Correct the subscript list, or declaration of the base, whichever appropriate.

T IEM0626I THE DEFINING BASE FOR zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT DATA.

Explanation: The only legal data types that may be used for defining bases are String, Arithmetic, Task, Event, and Label.

System Action: The compilation is terminated.

User Response: Check that the

defining base is correctly written and declared.

T IEM0627I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG.

Explanation: System Generation compiler restriction

System Action: The compilation is terminated.

User Response: Reduce statement size

T IEM0628I IMPLEMENTATION RESTRICTION. THE NESTING OF REFERENCES TO DATA DEFINED WITH A SUBSCRIPTED BASE IS TOO DEEP.

Explanation: The complexity of defining has resulted in a level of nesting which is too great for the compiler.

System Action: The compilation is terminated.

User Response: Reduce complexity of defining.

T IEM0629I ARRAY zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY HAS THE POS ATTRIBUTE WITH ISUB DEFINING.

Explanation: The POS attribute may not be specified for correspondence defining.

System Action: The compilation is terminated.

User Response: Delete POS attribute

T IEM0630I THE DESCRIPTION OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx DOES NOT MATCH THAT OF THE DEFINING BASE.

Explanation: For correspondence defining, if either the base or the defined item are arrays of structures, then both must be arrays of structures.

System Action: The compilation is terminated.

User Response: Correct the program. Note that POS (1) may be used to force overlay defining.

T IEM0631I IMPLEMENTATION RESTRICTION. THE CORRESPONDENCE DEFINING OF yyyy AN ARRAY OF STRUCTURES DECLARED IN STATEMENT NUMBER xxx.

Explanation: Correspondence defining with arrays of structures is not supported by the compiler.

System Action: The compilation is terminated.

User Response: Declare the base arrays of the defined structure as correspondence defined on the matching base arrays of the base structure. Note that for this to be valid, the base arrays of the defined structure must have unique names and be declared at level 1. This alternative method precludes structure operations with the defined item, but achieves the desired mapping.

T IEM0632I THE BOUNDS OF zzzz CORRESPONDENCE DEFINED IN STATEMENT NUMBER xxx ARE NOT A SUBSET OF THE BASE.

Explanation: For correspondence defining not involving iSUB's, the bounds of the defined array must be a subset of the corresponding bounds of the base array.

System Action: The compilation is terminated.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct program. Note that POS (1) may be used to force overlay defining.

S IEM0633I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx IS NOT AT LEVEL 1. THE STATEMENT HAS BEEN IGNORED.

Explanation: An identifier specified in an ALLOCATE statement must refer to a major structure or data not contained in a structure. A major structure identifier may optionally be followed by a full structure description.

System Action: The ALLOCATE statement is deleted

User Response: Replace erroneous identifier by that of the containing major structure.

S IEM0634I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx HAS NOT BEEN DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CONTROLLED data may be allocated. Data may only obtain the attribute CONTROLLED from an explicit declaration.

System Action: The ALLOCATE statement is deleted

User Response: Construct a DECLARE statement for the identifier

S IEM0636I ITEM TO BE ALLOCATED IN STATEMENT NUMBER xxx WAS NOT DECLARED CONTROLLED. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CONTROLLED data may be specified in ALLOCATE statements.

System Action: The ALLOCATE statement is deleted

User Response: Declare the identifier CONTROLLED

E IEM0637I A CONFLICTING ATTRIBUTE WAS GIVEN FOR zzzz IN STATEMENT NUMBER xxx. THE ATTRIBUTE HAS BEEN IGNORED.

Explanation: Attributes given for an identifier in an ALLOCATE statement may not conflict with those given explicitly or assumed by default from the declaration.

System Action: Ignores the attribute from the ALLOCATE

S IEM0638I THE STRUCTURE DESCRIPTION GIVEN ON STATEMENT NUMBER xxx DIFFERS FROM THAT DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: If a description of a major structure is given on an ALLOCATE statement, the description must match that declared.

System Action: The ALLOCATE statement is deleted

S IEM0640I AN INVALID ATTRIBUTE WAS GIVEN IN STATEMENT NUMBER xxx. THE STATEMENT HAS BEEN IGNORED.

Explanation: Only CHAR, BIT, INITIAL, and Dimension attributes are permitted in ALLOCATE statements.

System Action: The ALLOCATE statement is deleted

E IEM0641I CONFLICTING ATTRIBUTES HAVE BEEN GIVEN FOR zzzz IN STATEMENT NUMBER xxx. THE FIRST LEGAL ONE HAS BEEN USED.

Explanation: At most, one attribute in the following classes may be given for an identifier in an ALLOCATE statement: Dimension, String(CHAR or BIT), INITIAL.

System Action: All attributes after the first in a particular class are ignored.

S IEM0642I DIMENSIONALITY GIVEN IN STATEMENT NUMBER xxx DIFFERS FROM THAT DECLARED. THE STATEMENT HAS BEEN IGNORED.

Explanation: If a dimension attribute is given for an identifier in an ALLOCATE statement, the identifier must have been declared with the same dimensionality.

System Action: The ALLOCATE statement is deleted

User Response: Correct declaration or ALLOCATE Statement, whichever applicable.

W IEM0643I THE LEVEL NUMBER DECLARED FOR zzzz IS NOT THE SAME AS THAT GIVEN IN STATEMENT NUMBER xxx. THE FORMER HAS BEEN USED.

Explanation: If a structure description is given in an ALLOCATE Statement, it must match the declaration. The indicated level number discrepancy may be an error.

System Action: Nothing further

User Response: Check that ALLOCATE statement is as intended

S IEM0644I STATEMENT NUMBER xxx CONTAINS AN ILLEGAL PARENTHESIZED LIST. THE STATEMENT HAS BEEN IGNORED.

Explanation: Factored attributes are not allowed on ALLOCATE statements.

System Action: Statement ignored

User Response: Remove parentheses and any factored attributes

S IEM0645I ATTRIBUTE GIVEN WITH BASED VARIABLE zzzz IN ALLOCATE STATEMENT NUMBER xxx HAS BEEN IGNORED.

Explanations: Based variable may not be specified with attributes

User Response: Correct ALLOCATE statement

S IEM0646I IDENTIFIER yyyy PRECEDING POINTER QUALIFIER IN STATEMENT NUMBER xxx IS NOT A NON-BASED POINTER VARIABLE

System Action: The identifier is replaced by a dummy dictionary reference; a later phase will delete the statement.

User Response: Correct the invalid statement

S IEM0647I POINTER-QUALIFIED IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A BASED VARIABLE

System Action: Identifier is replaced by a dummy dictionary reference; a later phase will delete the statement.

User Response: Correct the invalid statement

T IEM0652I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG.

System Action: Compilation terminated

User Response: Rewrite statement in question and recompile

T IEM0653I COMPILER ERROR. ILLEGAL ENTRY IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of statement

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0654I COMPILER ERROR IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of statement

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM0655I QUALIFIED NAME BEGINNING yyyy USED IN STATEMENT NUMBER xxx BUT NO PREVIOUS STRUCTURE DECLARATION GIVEN. DUMMY REFERENCE INSERTED.

System Action: Reference to the illegal variable or the whole statement will be deleted by later phases.

User Response: Correct program by inserting DECLARE statement

E IEM0656I MOST RECENT DECLARATION USED OF AMBIGUOUS QUALIFIED NAME OR STRUCTURE MEMBER BEGINNING yyyy IN STATEMENT NUMBER xxx

E IEM0657I QUALIFIED NAME BEGINNING yyyy IN STATEMENT NUMBER xxx CONTAINS MORE THAN ONE IDENTIFIER AT THE SAME STRUCTURE LEVEL.

System Action: The statement is deleted

S IEM0658I QUALIFIED NAME BEGINNING yyyy IN STATEMENT NUMBER xxx IS AN AMBIGUOUS REFERENCE. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phase

S IEM0673I ILLEGAL USE OF FUNCTION NAME ON LEFT HAND SIDE OF EQUAL SYMBOL IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases

S IEM0674I STATEMENT NUMBER xxx CONTAINS ILLEGAL USE OF FUNCTION yyyy

System Action: Reference to function or whole statement will be deleted by later phases

S IEM0675I IN STATEMENT NUMBER xxx IDENTIFIER yyyy AFTER GO TO IS NOT A LABEL OR LABEL VARIABLE KNOWN IN THE BLOCK CONTAINING THE GO TO.

System Action: Statement deleted by later phases.

S IEM0676I DEFERRED FEATURE. IDENTIFIER yyyy NOT ALLOWED AS A BUILT-IN FUNCTION OR PSEUDO-VARIABLE . DUMMY REFERENCE INSERTED IN STATEMENT NUMBER xxx

Explanation: A language feature has been used that is not supported by this version of the compiler. For details, refer to Appendix H of this publication.

System Action: Statement deleted by later phases

User Response: Correct statement by removing reference to function in error

S IEM0677I ILLEGAL PARENTHESIZED LIST IN STATEMENT NUMBER xxx FOLLOWS AN IDENTIFIER WHICH IS NOT A FUNCTION OR ARRAY. LIST DELETED.

S IEM0682I IN STATEMENT NUMBER xxx GO TO TRANSFERS CONTROL ILLEGALLY TO A FORMAT STATEMENT.

S IEM0683I zzzz WAS FOUND WHERE A FILE VARIABLE IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY DICTIONARY REFERENCE REPLACES ILLEGAL ITEM.

System Action: Statement will be deleted by later phases

W IEM0684I USE OF LABEL VARIABLE zzzz MAY RESULT IN AN ILLEGAL BRANCH IN STATEMENT NUMBER xxx

Explanation: It is possible that the label variable may contain a value which would cause control to branch illegally into a block.

System Action: None

User Response: Check validity of possible branches.

S IEM0685I zzzz IS NOT A STATEMENT LABEL ON AN EXECUTABLE STATEMENT. DUMMY REFERENCE INSERTED AFTER GO TO IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases

S IEM0686I zzzz APPEARS IN A FREE OR ALLOCATE STATEMENT BUT HAS NOT BEEN DECLARED CONTROLLED. DUMMY REFERENCE INSERTED IN STATEMENT NUMBER xxx .

System Action: A dummy reference is inserted. The statement will be deleted by a later phase

W IEM0687I IN STATEMENT NUMBER xxx GO TO zzzz TRANSFERS CONTROL ILLEGALLY TO ANOTHER BLOCK OR GROUP.

System Action: Statement will be deleted by later phases

S IEM0688I COMPILER ERROR. TOO FEW LEFT PARENTHESES IN STATEMENT NUMBER xxx

Explanation: This is a compiler error

System Action: None taken. Cascade errors may result.

User Response: Save relevant data. Call your local IBM representative.

S IEM0689I zzzz WAS FOUND WHERE A TASK IDENTIFIER IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases

S IEM0690I zzzz WAS FOUND WHERE EVENT VARIABLE IS REQUIRED IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases

S IEM0691I INVALID ITEM zzzz IN DATA LIST, OR 'FROM' OR 'INTO' OPTION, IN STATEMENT NUMBER xxx

System Action: Statement will be deleted by later phases

S IEM0692I DATA DIRECTED I/O LIST, OR FROM, OR INTO OPTION IN STATEMENT NUMBER xxx CONTAINS FORMAL PARAMETER OR DEFINED ITEM zzzz

System Action: Statement will be deleted by later phases

S IEM0693I ILLEGAL USE OF FUNCTION zzzz IN INPUT LIST IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement will be deleted by later phases

S IEM0694I IN THE FORMAT LIST IN STATEMENT NUMBER xxx A REMOTE FORMAT ITEM REFERENCES zzzz, WHICH IS NOT A STATEMENT LABEL IN THE CURRENT BLOCK. DUMMY REFERENCE INSERTED.

System Action: Format item deleted by later phase

S IEM0695I LABEL ARRAY zzzz IS NOT FOLLOWED BY A SUBSCRIPT LIST AFTER GO TO IN STATEMENT NUMBER xxx. DUMMY REFERENCE REPLACES REFERENCE TO ARRAY.

System Action: Statement will be deleted by later phases

W IEM0696I IN STATEMENT NUMBER xxx IT IS AN ERROR IF THE PARAMETER zzzz IN A REMOTE FORMAT ITEM REFERS TO A FORMAT STATEMENT WHICH IS NOT INTERNAL TO THE SAME BLOCK AS THE REMOTE FORMAT ITEM.

Explanation: Remote formats become executable code, but not internal procedures. Therefore they must appear in the same block in which they are used.

System Action: Object-time error message is compiled

User Response: Correct program and recompile

S IEM0697I STATEMENT LABEL zzzz ATTACHED TO STATEMENT NUMBER xxx IS USED AS A REMOTE FORMAT ITEM IN THAT STATEMENT. A DUMMY REPLACES THE REMOTE FORMAT ITEM.

System Action: Statement will be deleted by a later phase

S IEM0698I THE BASED VARIABLE zzzz IN LOCATE STATEMENT NUMBER xxx IS NOT AT LEVEL 1. DUMMY REFERENCE INSERTED.

System Action: The statement will be deleted by a later phase.

S IEM0699I STRUCTURE ARGUMENT zzzz OF FROM OR INTO OPTION IN STATEMENT NUMBER xxx IS NOT A MAJOR STRUCTURE. DUMMY REFERENCE INSERTED

System Action: Statement deleted by later phase

S IEM0700I ILLEGAL USE OF FUNCTION, LABEL OR VARYING STRING zzzz AS ARGUMENT OF FROM OR INTO OPTION IN STATEMENT NUMBER xxx. DUMMY REFERENCE INSERTED.

System Action: Statement deleted by later phase

S IEM0701I ARGUMENT zzzz OF SET OPTION IS NOT A POINTER VARIABLE. DUMMY REFERENCE INSERTED.

System Action: Statement deleted by later phase.

W IEM0702I LABEL, TASK OR EVENT VARIABLE zzzz USED IN FROM OR INTO OPTION IN STATEMENT NUMBER xxx MAY LOSE ITS VALIDITY IN TRANSMISSION

S IEM0703I INVALID IDENTIFIER zzzz FREED IN STATEMENT NUMBER xxx

Explanation: An identifier in the FREE statement is not a CONTROLLED variable.

System Action: Invalid identifier replaced by dummy.

S IEM0704I STATEMENT NUMBER xxx CONTAINS INVALID USE OF FUNCTION zzzz

System Action: Statement deleted by later phase

W IEM0705I IF THE LABEL VARIABLE IN GO TO STATEMENT NUMBER xxx ASSUMES THE VALUE OF ITS VALUE-LIST MEMBER zzzz, THE STATEMENT WILL CONSTITUTE AN INVALID BRANCH INTO AN ITERATIVE DO GROUP.

System Action: None

User Response: Check that branch will be valid at execution time.

S IEM0706I VARIABLE zzzz IN LOCATE STATEMENT IS NOT A BASED VARIABLE. DUMMY REFERENCE INSERTED.

System Action: The statement is deleted by a later phase.

User Response: Correct the invalid statement

S IEM0707I ARGUMENT zzzz OF IN OPTION IS NOT AN AREA VARIABLE. DUMMY REFERENCE INSERTED.

System Action: The statement is deleted by a later phase.

User Response: Correct the invalid statement

E IEM0708I ASTERISK NOT ALONE IN SETS OR USES LIST. REMAINDER OF LIST IGNORED.

S IEM0709I IMPLEMENTATION RESTRICTION. ARGUMENT NUMBER IN SETS OR USES LIST TOO LARGE. REDUCED TO 255.

E IEM0710I REPEATED ARGUMENT NUMBER IGNORED IN SETS OR USES LIST.

T IEM0711I COMPILER ERROR IN SETS LIST.

Explanation: Compiler error found in scan of SETS list

System Action: Compilation terminated.

User Response: Save relevant data. Call your local IBM representative.

E IEM0712I IDENTIFIER yyyy IN SETS OR USES LIST NOT KNOWN AT POINT OF OCCURRENCE OF LIST. IDENTIFIER IGNORED.

E IEM0713I IDENTIFIER yyyy IN SETS LIST IS BASED VARIABLE. DUMMY REFERENCE INSERTED.

User Response: Correct source program by removing reference to based variable.

S IEM0715I TEXT yyyy ASSOCIATED WITH THE INITIAL ATTRIBUTE IN STATEMENT NUMBER xxx IS ILLEGAL AND HAS BEEN IGNORED.

Explanation: The INITIAL attribute has been used incorrectly.

System Action: The INITIAL attribute is deleted

S IEM0718I INVALID CHECK LIST IN STATEMENT NUMBER xxx. STATEMENT HAS BEEN CHANGED TO 'ON ERROR'.

S IEM0719I ELEMENT OF LABEL ARRAY zzzz WHICH IS DECLARED WITH INITIAL ATTRIBUTE USED AS STATEMENT LABEL ON STATEMENT NUMBER xxx

System Action: Label is deleted

S IEM0720I SUBSCRIPTED IDENTIFIER zzzz USED AS LABEL ON STATEMENT NUMBER xxx IS NOT A LABEL ARRAY

System Action: Label is deleted

S IEM0721I ELEMENT OF STATIC LABEL ARRAY zzzz USED AS LABEL ON STATEMENT NUMBER xxx

System Action: Label is deleted

S IEM0722I ELEMENT OF LABEL ARRAY zzzz USED AS LABEL ON STATEMENT NUMBER xxx IN BLOCK OTHER THAN THE ONE IN WHICH IT IS DECLARED.

System Action: An error statement is inserted in the text in place of the offending label.

T IEM0723I COMPILER ERROR IN STATEMENT NUMBER xxx

Explanation: Compiler error found in scan of text

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM0724I FORMAL PARAMETER zzzz IN CHECK LIST. PARAMETER IS IGNORED.

Explanation: The identifier list of a CHECK prefix must not contain formal parameters

S IEM0725I STATEMENT NUMBER xxx HAS BEEN DELETED DUE TO A SEVERE ERROR NOTED ELSEWHERE.

System Action: The whole statement is replaced by an error statement.

S IEM0726I IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A FILE NAME. THE STATEMENT IS DELETED.

Explanation: The identifier has been used previously in a different context and is therefore not recognized as a file name

S IEM0727I IDENTIFIER zzzz IN STATEMENT NUMBER xxx IS NOT A CONDITION NAME. THE STATEMENT IS DELETED.

Explanation: The identifier has been used previously in a different context and is therefore not recognized as a condition name

T IEM0728I COMPILATION TERMINATED DUE TO A PREVIOUSLY DETECTED SEVERE ERROR IN STATEMENT NUMBER xxx

Explanation: A previous module has inserted a dummy dictionary reference into the second file. The compiler cannot recover.

S IEM0729I COMPILER ERROR IN SCALE FACTOR IN PICTURE BEGINNING yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture halted. All references to picture deleted.

User Response: Save relevant data. Call your local IBM representative.

S IEM0730I MORE THAN ONE SIGN CHARACTER PRESENT IN A SUBFIELD OF PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0731I PICTURE CHARACTER M APPEARS IN NON-STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0732I FIELD MISSING IN STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0733I ILLEGAL EDIT CHARACTERS AT START OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0734I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN POUNDS FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0735I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN SHILLINGS FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0736I WRONG NUMBER OF DELIMITER CHARACTERS M IN STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0737I ILLEGAL CHARACTER OR ILLEGAL NUMBER OF CHARACTERS IN PENCE FIELD OF STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0739I STATIC PICTURE CHARACTER $ S + - NOT AT EXTREMITY OF SUBFIELD. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

E IEM0740I MULTIPLE USE OF E K OR V IN PICTURE. PICTURE TRUNCATED AT ILLEGAL CHARACTER. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

E IEM0741I CR OR DB INCORRECTLY POSITIONED IN SUBFIELD. PICTURE TRUNCATED AT THIS POINT. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

E IEM0742I CR OR DB GIVEN FOR A NON-REAL, NON-NUMERIC OR FLOATING FIELD.

PICTURE TRUNCATED BEFORE CR OR DB IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

S IEM0745I ILLEGAL USE OF PICTURE CHARACTER Z OR * IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0746I STERLING MARKER FOUND IN OTHER THAN FIRST POSITION IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0747I STERLING PICTURE CHARACTERS FOUND IN NON-STERLING PICTURE. SCANNING OF PICTURE STOPPED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

E IEM0748I ILLEGAL USE OF SCALING FACTOR IN PICTURE. SCALING FACTOR ONWARDS DELETED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Picture truncated at point indicated

S IEM0749I ILLEGAL USE OF SCALING FACTOR IN PICTURE. SCAN OF PICTURE TERMINATED. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture terminated; picture ignored by later phases.

S IEM0750I ILLEGAL CHARACTER PRESENT IN CHARACTER STRING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0751I NO MEANINGFUL CHARACTERS IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0752I ILLEGAL USE OF,OR ILLEGAL CHARACTERS IN, STERLING PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0754I ILLEGAL CHARACTER IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0755I ILLEGAL USE OF DRIFTING EDITING SYMBOLS S $ + - IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

S IEM0756I IMPLEMENTATION RESTRICTION. PRECISION TOO LARGE OR PICTURE TOO LONG IN PICTURE BEGINNING yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Scan of picture continued with item ignored, but picture will be ignored by later phases.

T IEM0758I COMPILER ERROR IN PHASE FT.

Explanation: Compiler error found in scan of dictionary

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM0759I IMPLEMENTATION RESTRICTION. STERLING CONSTANT EXCEEDS 4166666666666.13.3L. HIGH ORDER DIGITS LOST DURING CONVERSION TO DECIMAL.

System Action: High order digits lost somewhere in the following conversion process: shift pounds field left one digit, double by addition. Add shillings field. Add result, doubled by addition, to result shifted left one digit. Add pence field.

S IEM0760I IMPLEMENTATION RESTRICTION. EXPONENT FIELD TOO LARGE IN PICTURE yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS. .

System Action: Reference to picture deleted

S IEM0761I PICTURE CHARACTER E OR K APPEARS WITHOUT AN EXPONENT FIELD FOLLOWING IT. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: Reference to picture deleted from program

User Response: Correct picture

S IEM0762I PICTURE CHARACTER E OR K IS NOT PRECEDED BY A DIGIT POSITION CHARACTER. PICTURE IN ERROR IS yyyy. THIS PICTURE OCCURS IN STATEMENT NUMBER xxx, AND POSSIBLY IN OTHER STATEMENTS.

System Action: References to picture deleted

S IEM0769I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx AS EXPANDED IS TOO LONG AND HAS BEEN DELETED.

User Response: Simplify by splitting into two or more statements and recompile

T IEM0770I COMPILER ERROR IN INPUT TO PHASE GA IN STATEMENT NUMBER xxx

Explanation: The compiler has encountered meaningless input to phase GA.

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM0771I IMPLEMENTATION RESTRICTION. NESTING OF FORMAT LISTS IN STATEMENT NUMBER xxx EXCEEDS 20. STATEMENT DELETED.

S IEM0778I AN INTERMEDIATE VARIABLE HAS BEEN CREATED IN READ INTO STATEMENT NUMBER xxx. THIS STATEMENT SPECIFIES FILE zzzz, WHICH HAS BEEN DECLARED WITH THE ENV(COBOL) ATTRIBUTE AND THE EVENT OPTION. THE EVENT OPTION HAS BEEN DELETED.

Explanation: The intermediate variable has been created because there is a difference between the PL/I mapping and the COBOL mapping of the READ INTO variable. The READ INTO statement has been expanded into:

```
READ INTO (Intermediate
  variable);
Variable = Intermediate
  variable;
```

The READ statement must have been completed before the assignment takes place. The EVENT option has been deleted to ensure that the READ statement is complete before processing continues.

System Action: Delete the EVENT option and continue

User Response: Check the use of the EVENT option or of the COBOL file.

S IEM0779I AN INTERMEDIATE VARIABLE HAS BEEN CREATED IN WRITE/REWRITE FROM STATEMENT NUMBER xxx. THIS STATEMENT SPECIFIES FILE zzzz, WHICH HAS BEEN DECLARED WITH THE ENV(COBOL) ATTRIBUTE AND THE EVENT OPTION. THE EVENT OPTION HAS BEEN DELETED.

Explanation: The intermediate variable has been created because there is a difference between the PL/I mapping and the COBOL mapping of the

WRITE/REWRITE FROM variable. The WRITE/REWRITE FROM statement has been expanded to:

```
Intermediate
  variable = variable;
WRITE/REWRITE        FROM
  (Intermediate variable);
```

The WRITE/REWRITE statement must have been completed before the intermediate variable can be deleted. The EVENT option has been deleted to ensure that the WRITE/REWRITE statement is complete before processing continues.

System Action: Delete the EVENT option and continue

User Response: Check the use of the EVENT option or of the COBOL file.

S IEM0780I THE USE OF COBOL FILE zzzz IN LOCATE STATEMENT NUMBER xxx MAY LEAD TO ERRORS WHEN THE RECORD IS PROCESSED.

Explanation: The COBOL structure-mapping is not necessarily the same as the PL/I structure-mapping.

System Action: The statement is deleted

User Response: Either the LOCATE statement must be replaced by a WRITE FROM statement, or a non-COBOL file must be used.

S IEM0781I THE USE OF COBOL FILE zzzz IN READ SET STATEMENT NUMBER xxx MAY LEAD TO ERRORS WHEN THE RECORD IS PROCESSED.

Explanation: The COBOL structure-mapping is not necessarily the same as the PL/I structure-mapping.

System Action: The statement is deleted

User Response: Either the READ SET statement must be replaced by a READ INTO statement, or a non-COBOL file must be used.

S IEM0782I THE ATTRIBUTES OF zzzz USED IN RECORD I/O STATEMENT NUMBER XXX ARE NOT PERMITTED WHEN A COBOL FILE IS USED.

Explanation: The attributes referred to do not exist in COBOL

System Action: The statement is deleted

S IEM0784I INVALID ARGUMENT LIST FOR ALLOCATION FUNCTION IN STATEMENT NUMBER xxx HAS BEEN TRUNCATED OR DELETED.

Explanation: Only a single argument can be given in the ALLOCATION function, and it must be one of the following:

1. a major structure
2. an unsubscripted array or scalar variable, not in a structure

It must also be of nonbased CONTROLLED storage class.

System Action: If the argument list begins with a valid operand, that operand is used as the argument; otherwise, the argument list is deleted.

W IEM0786I NAME, NOT VALUE, OF FUNCTION zzzz PASSED AS ARGUMENT IN STATEMENT NUMBER xxx

S IEM0787I INCORRECT NUMBER OF ARGUMENTS FOR FUNCTION zzzz IN STATEMENT NUMBER xxx

T IEM0790I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: The first bytes of a statement to be moved out exceed the text block length.

System Action: The statement is truncated. Compilation will not be completed.

User Response: Subdivide statement into two or more separate statements, and recompile.

W IEM0791I NUMBER OF ARGUMENTS FOR zzzz STATEMENT NUMBER xxx INCONSISTENT WITH NUMBER USED ELSEWHERE

Explanation: The number of arguments for zzzz has not been explicitly declared. 'ELSEWHERE' refers either to the PROCEDURE or ENTRY statement for zzzz, or to a previous invocation of the function.

E IEM0792I IN STATEMENT NUMBER xxx IT IS IMPOSSIBLE TO CONVERT FROM THE ATTRIBUTES OF ARGUMENT NUMBER nnnn TO THOSE OF THE CORRESPONDING PARAMETER IN ENTRY zzzz. THE PARAMETER DESCRIPTION IS IGNORED.

Explanation: Self explanatory. Examples of circumstances under which the error message is generated are label arguments to data item parameters, array arguments to scalar parameters.

System Action: The parameter description is ignored. If the parameter description is correct, this will give rise to totally incorrect execution.

User Response: Correct the parameter description or the argument so that at least conversion is possible.

W IEM0793I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A SCALAR AND THE CORRESPONDING PARAMETER IS A STRUCTURE.

System Action: A temporary structure of the same type as the parameter description is created and the argument is assigned to each base element, converting where necessary.

S IEM0794I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz CONTAINS A SUBSCRIPTED VARIABLE WITH THE WRONG NUMBER OF SUBSCRIPTS. THE STATEMENT HAS BEEN DELETED.

S IEM0795I DEFERRED FEATURE. IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz CONTAINS A CROSS-SECTION OF AN ARRAY OF STRUCTURES. STATEMENT DELETED.

Explanation: The usage referred to is not supported by this version of the compiler. For details, refer to Appendix H of this publication.

S IEM0796I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A GENERIC ENTRY NAME AND THERE IS NO CORRESPONDING ENTRY DESCRIPTION. STATEMENT DELETED.

S IEM0797I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS A BUILT-IN FUNCTION WHICH MAY NOT BE PASSED AS AN ARGUMENT. STATEMENT DELETED.

S IEM0798I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS NOT PERMISSIBLE. THE STATEMENT HAS BEEN DELETED.

Explanation: The message is generated, for example, when scalar arguments are given for array built-in functions. For the ADDR function the message could indicate that the function cannot return a valid result because the argument does not satisfy the requirements of contiguous storage. This could occur, for example, if the argument was a cross-section of an array.

E IEM0799I IN STATEMENT NUMBER xxx A DUMMY ARGUMENT HAS BEEN CREATED FOR ARGUMENT NUMBER nnnn OF ENTRY zzzz . THIS ARGUMENT APPEARS IN A SETS LIST.

System Action: The value assigned to the temporary argument during the execution of the procedure is lost on return from the procedure.

W IEM0800I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn IN ENTRY zzzz IS A SCALAR AND THE CORRESPONDING PARAMETER IS AN ARRAY.

System Action: A temporary array with the attributes of the entry description is created and the scalar is assigned to each element of the array, converting the type if necessary.

S IEM0801I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn IN ENTRY zzzz IS SCALAR CORRESPONDING TO AN ARRAY PARAMETER WITH * BOUNDS. THE STATEMENT HAS BEEN DELETED.

W IEM0802I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz DOES NOT MATCH THE PARAMETER. A DUMMY ARGUMENT HAS BEEN CREATED.

T IEM0803I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx CONTAINS TOO MANY NESTED FUNCTION REFERENCES. LIMIT EXCEEDED AT ARGUMENT NUMBER nnnn OF ENTRY zzzz

System Action: Compilation terminated

User Response: Reduce depth of function call nesting.

T IEM0804I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

System Action: Compilation terminated

User Response: Reduce statement size

S IEM0805I DEFERRED FEATURE. IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS AN EVENT. STATEMENT DELETED.

Explanation: A language feature has been used that is not supported by this version of the compiler. For details, refer to Appendix H of this publication.

S IEM0806I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF ENTRY zzzz IS NOT CONTROLLED BUT THE CORRESPONDING PARAMETER IS.

System Action: An execution error will occur on entry to the called procedure

S IEM0807I IN STATEMENT NUMBER xxx ARGUMENT NUMBER nnnn OF BUILT-IN FUNCTION zzzz IS AN ENTRY NAME. THE STATEMENT HAS BEEN DELETED.

T IEM0816I COMPILER ERROR. INVALID END OF STATEMENT NUMBER xxx

Explanation: Compiler error in scan of input text

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0817I COMPILER ERROR IN LABEL CHAIN FOR STATEMENT NUMBER xxx

Explanation: Compiler error in scanning labels of a statement

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative. Note that this error can be avoided by using only one label on the statement.

T IEM0818I COMPILER ERROR IN DICTIONARY ENTRY FOR STATEMENT NUMBER xxx

Explanation: Compiler error in scanning source text

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0819I COMPILER ERROR IN CHECK/NOCHECK LIST ENTRY FOR STATEMENT NUMBER xxx

Explanation: Compiler error in CHECK or NOCHECK list dictionary entry

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0820I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx TOO LONG.

Explanation: Statement length exceeds text-block size.

System Action: Compilation is terminated

User Response: Subdivide statement and recompile

E IEM0821I INVALID ITEM zzzz IGNORED IN CHECK LIST IN STATEMENT NUMBER xxx

Explanation: Valid items in CHECK lists are: statement labels, entry labels, and scalar, array, structure, or label variables. Subscripted variable names, or data having the DEFINED attribute, are not allowed.

W IEM0823I IMPLEMENTATION RESTRICTION. NO ROOM FOR zzzz IN CHECK TABLE STATEMENT NUMBER xxx

Explanation: The CHECK list table has overflowed

System Action: The item mentioned is ignored

User Response: Do not CHECK so many items

T IEM0824I IMPLEMENTATION RESTRICTION. TOO MANY CHECKED ITEMS WITHIN STATEMENT NUMBER xxx

Explanation: A stack used to trace nested IF statements has overflowed

System Action: Compilation is terminated

User Response: Rephrase IF statements, or do not CHECK so many items.

T IEM0825I COMPILER ERROR IN READ DATA STATEMENT NUMBER xxx

Explanation: Compiler error in processing GET or READ DATA statement

System Action: Compilation is terminated

User Response: Avoid the error by supplying an explicit DATA list

W IEM0826I IMPLEMENTATION RESTRICTION. CHECK WILL NOT BE RAISED FOR zzzz IN STATEMENT NUMBER xxx BECAUSE OF EVENT OPTION

Explanation: The compiler does not raise the CHECK condition for variables when they are changed in statements containing an EVENT option.

S IEM0832I IN THE EXPANSION OF BY NAME ASSIGNMENT STATEMENT NUMBER xxx A SET OF MATCHING ELEMENTS HAS BEEN FOUND BUT THEY ARE NOT ALL BASE ELEMENTS. THE STATEMENT HAS BEEN DELETED.

Explanation: For a valid component scalar assignment to result from a BY NAME structure assignment, it is necessary that all the scalar names derived from original structure name operands have identical qualification relative to the structure name originally specified. e.g: DCL 1S, 2T, 2U, 3V; DCL 1W, 2T, 2U; W=S; gives rise to the component assignments W.T=S.T; W.U=S.U; the second of which is invalid.

System Action: The BY NAME assignment statement is deleted

User Response: Refer to the PL/I Reference Manual - rules for expansion of structure assignment BY NAME - and correct the error.

S IEM0833I THE EXPANSION OF BY NAME ASSIGNMENT STATEMENT NUMBER xxx HAS RESULTED IN NO COMPONENT ASSIGNMENTS.

System Action: The statement is treated as a null statement

S IEM0834I THE ASSIGNED OPERAND IN BY NAME ASSIGNMENT STATEMENT NUMBER xxx IS NOT A STRUCTURE OR AN ARRAY OF STRUCTURES. STATEMENT DELETED.

Explanation: In BY NAME assignment, the operand to the left of the equals sign must be a structure or an array of structures.

S IEM0835I ALL OPERANDS LEFT OF EQUAL SYMBOL IN MULTIPLE STRUCTURE ASSIGNMENT STATEMENT NUMBER xxx ARE NOT STRUCTURES. STATEMENT DELETED

Explanation: In multiple structure assignment, all the operands being assigned to must be structures.

System Action: Replaces statement by a null statement and continues

User Response: Break statement up into a series of separate statements

S IEM0836I ILLEGAL ARRAY REFERENCE IN STRUCTURE ASSIGNMENT OR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: In PL/I, arrays of scalars are invalid operands in structure, or array of structure, expressions.

T IEM0837I COMPILER ERROR IN INPUT TO PHASE IEMHF.

Explanation: Meaningless input. This message is also produced if an error is found in a DECLARE statement. In that case, a second message, of severity level severe, is issued giving details of the error.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM0838I EXPRESSION RIGHT OF EQUAL SYMBOL IN BY NAME ASSIGNMENT STATEMENT NUMBER xxx CONTAINS NO STRUCTURES. BY NAME OPTION DELETED.

Explanation: In an assignment statement having the BY NAME option, the expression to the right of the equal symbol should contain at least one structure or array of structures.

S IEM0848I IMPLEMENTATION RESTRICTION. THE EXPANSION OF STRUCTURE EXPRESSIONS IN STATEMENT NUMBER xxx HAS CAUSED A TABLE INTERNAL TO THE COMPILER TO OVERFLOW. STATEMENT DELETED.

Explanation: The nesting of structure expressions in argument lists is too deep.

User Response: Decrease the nesting.

S IEM0849I AN EXPRESSION OR ASSIGNMENT IN STATEMENT NUMBER xxx EITHER CONTAINS SEPARATE STRUCTURES WITH DIFFERENT STRUCTURING, OR CONTAINS BOTH A STRUCTURE AND AN ARRAY OF STRUCTURES. THE STATEMENT HAS BEEN DELETED.

Explanation: PL/I does not allow separate structures with different structuring within the same expression or assignment. The (F) compiler does not support reference to both a structure and an array of structures within the same expression or assignment.

User Response: Correct the source code. The two restrictions quoted above are detailed in the PL/I Reference Manual and in Appendix H of this publication, respectively.

S IEM0850I THE BOUNDS OF THE BASE ARRAYS OF THE STRUCTURE OPERANDS OF THE STRUCTURE EXPRESSION OR ASSIGNMENT IN STATEMENT NUMBER xxx ARE NOT THE SAME. THE STATEMENT HAS BEEN DELETED.

Explanation: In a structure assignment or expression, all structure operands must have the same number of contained elements at the next level. Corresponding sets of contained elements must all be arrays of structures, structures, arrays, or scalars. The arrays must have the same dimensionality and bounds.

User Response: Refer to the PL/I Reference Manual - the expansion of array and structure expressions and assignments - and correct the program.

S IEM0851I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

Explanation: The expansion of structure expressions or assignments has given rise to a statement which exceeds one text block in length

User Response: Decrease statement size

S IEM0852I A SUBSCRIPTED REFERENCE TO AN ARRAY OF STRUCTURES IN STATEMENT NUMBER xxx HAS THE WRONG NUMBER OF SUBSCRIPTS. THE STATEMENT HAS BEEN DELETED.

Explanation: Subscripted references to arrays must have subscript expressions equal in number to the dimensionality of the array

User Response: Correct subscripted reference

S IEM0853I DEFERRED FEATURE. A STRUCTURE ASSIGNMENT OR EXPRESSION IN STATEMENT NUMBER xxx INVOLVES CROSS SECTIONS OF ARRAYS. STATEMENT DELETED.

Explanation: This version of the compiler does not support reference to cross sections of arrays of structures.

User Response: Expand the statement in DO loops replacing '*'s in subscripts by the appropriate DO control variable.

S IEM0864I IMPLEMENTATION RESTRICTION. NESTING OF ARRAY ASSIGNMENTS OR I/O LISTS TOO DEEP. STATEMENT NUMBER xxx DELETED.

Explanation: Nesting of functions, combined with size of arrays involved, is too great

User Response: Simplify by splitting into two or more statements

S IEM0865I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN DELETED.

Explanation: Nesting of functions with array arguments involves a large expansion of text

User Response: Simplify by splitting into two or more statements

S IEM0866I NEITHER MULTIPLE ASSIGNMENT COMMA NOR ASSIGNMENT MARKER FOUND IN CORRECT POSITION. STATEMENT NUMBER xxx DELETED.

Explanation: An expression occurring on the left-hand side of an assignment must be contained within parentheses.

S IEM0867I NUMBER OF * SUBSCRIPTS SPECIFIED FOR zzzz IS NOT THE DIMENSIONALITY OF THE LEFTMOST ARRAY. STATEMENT NUMBER xxx DELETED

Explanation: Array references in expressions must have the same dimensionality

S IEM0868I BOUNDS OF ARRAY zzzz ARE NOT SAME AS FOR LEFTMOST ARRAY IN ASSIGNMENT OR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Array references in expressions must have the same bounds

S IEM0869I DIMENSIONS OF ARRAY zzzz ARE NOT THE SAME AS FOR LEFTMOST ARRAY IN ASSIGNMENT OR I/O EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Array references in expressions must have the same dimensionality

S IEM0870I ARGUMENT OF PSEUDO-VARIABLE INVALID. STATEMENT NUMBER xxx IS DELETED.

Explanation: Arguments of pseudo-variables must be variables which are not expressions

S IEM0871I SCALAR zzzz ON LEFT OF EQUAL SYMBOL IN ARRAY ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

S IEM0872I NUMBER OF SUBSCRIPTS SPECIFIED FOR LEFTMOST OPERAND zzzz IS NOT SAME AS DIMENSIONALITY. STATEMENT NUMBER xxx DELETED.

Explanation: The number of subscripts specified must be the same as the number of dimensions of the array

S IEM0873I ARGUMENTS OF PSEUDO-VARIABLE COMPLEX INCORRECT. STATEMENT NUMBER xxx DELETED.

Explanation: Only one argument given for COMPLEX, or expression illegally used as argument for pseudo-variable.

S IEM0874I SECOND ARGUMENT OF PSEUDO-VARIABLE COMPLEX OR FIRST ARGUMENT OF REAL, IMAG, OR UNSPEC EITHER IS NOT FOLLOWED BY A RIGHT PARENTHESIS OR IS AN EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Too many arguments given for this pseudo-variable, or expression used as argument.

S IEM0875I ONSOURCE OR ONCHAR APPEARS IN A DIMENSIONED ARRAY ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Explanation: ONSOURCE and ONCHAR may only appear in scalar assignments

S IEM0876I ARGUMENTS OF PSEUDO-VARIABLE SUBSTR INCORRECT. STATEMENT NUMBER xxx DELETED.

Explanation: Only one argument given for SUBSTR, or expression illegally used as argument for pseudo-variable.

S IEM0877I UNSUBSCRIPTED ARRAY zzzz IN SCALAR ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

Explanation: Only scalars can be assigned to scalars

S IEM0878I STRUCTURE zzzz FOUND IN ARRAY OR SCALAR EXPRESSION. STATEMENT NUMBER xxx DELETED.

Explanation: Structures may only be assigned to structures

S IEM0879I PSEUDO-VARIABLE COMPLEX, REAL, IMAG, UNSPEC, COMPLETION, OR SUBSTR LACKS ARGUMENTS. STATEMENT NUMBER xxx DELETED.

Explanation: Pseudo-variables COMPLEX, REAL, IMAG, UNSPEC, COMPLETION, and SUBSTR require arguments

User Response: Check whether the variable was intended as a pseudo-variable or whether it should have been declared otherwise

S IEM0880I NUMBER OF SUBSCRIPTS SPECIFIED FOR zzzz IS NOT SAME AS DIMENSIONALITY. STATEMENT NUMBER xxx DELETED.

Explanation: The number of subscripts specified must be the same as the number of dimensions of the array

S IEM0881I NUMBER OF DIMENSIONS IN REFERENCE TO zzzz IS NOT SAME AS THAT OF EXPRESSION OR ASSIGNMENT. STATEMENT NUMBER xxx DELETED.

T IEM0882I COMPILER ERROR. INVALID INPUT TO PHASE HK AT STATEMENT NUMBER xxx

Explanation: Illegal text has been encountered

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM0896I IMPLEMENTATION RESTRICTION. TOO MANY LEVELS OF ISUB NESTING IN STATEMENT NUMBER xxx

Explanation: Stack has overflowed scratch core. The maximum number of levels of nesting possible depends on the dimensionality of the arrays involved.

System Action: Compilation terminated

User Response: Reduce the number of levels of nesting in the statement

S IEM0897I ISUB DEFINED OPERAND zzzz HAS NOT BEEN DECLARED AS AN ARRAY. ISUBS IN STATEMENT NUMBER xxx DELETED.

User Response: Declare defined item with dimension attribute

T IEM0898I NO SUBSCRIPTS AFTER ISUB-DEFINED ITEM zzzz IN STATEMENT NUMBER xxx.

Explanation: This is a compiler error

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

E IEM0899I MULTIPLIER IN ISUB DEFINING LIST FOR zzzz IN STATEMENT NUMBER xxx IS NOT A SCALAR EXPRESSION.

Explanation: A comma has been found within the iSUB multiplier expression

System Action: Remainder of expression, after comma, ignored

User Response: Rewrite expression

E IEM0900I * USED AS SUBSCRIPT FOR ISUB DEFINED ITEM zzzz IN STATEMENT NUMBER xxx. ZERO SUBSTITUTED.

User Response: Rewrite without *

E IEM0901I ISUB NUMBER IN DEFINING LIST FOR zzzz IN STATEMENT NUMBER xxx IS TOO GREAT. MAXIMUM NUMBER USED.

System Action: The iSUB number is replaced by the number of dimensions of the defined array.

User Response: Rewrite defining DECLARE statement

E IEM0902I WRONG NUMBER OF SUBSCRIPTS FOR ISUB DEFINED ITEM zzzz IN STATEMENT NUMBER xxx. SUBSCRIPTS IGNORED OR ZERO SUPPLIED.

User Response: Rewrite with correct number of subscripts. The error may be in the reference to the defined item or in the defining DECLARE statement.

T IEM0903I COMPILER ERROR. ERROR DETECTED IN DEFINING ISUB LIST FOR zzzz IN STATEMENT NUMBER xxx

Explanation: Compiler error. Either (a) SUB not found where expected, or (b) SUB0 found without a multiplier expression.

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM0906I STATEMENT DELIMITER FOUND WITHIN SUBSCRIPT LIST FOR zzzz IN STATEMENT NUMBER xxx

Explanation: The subscript scan routine has found a statement marker

System Action: The present statement is dropped and the new one processed. Compilation will not be completed.

User Response: Check text, but this is probably a compiler error, in which case save relevant data and contact your your local IBM representative.

S IEM0907I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: Statement length exceeds text block size

System Action: Statement is truncated. Compilation will not be completed.

User Response: Simplify statement

S IEM1024I ILLEGAL USE OF zzzz IN STATEMENT NUMBER xxx. A FIXED BINARY ZERO CONSTANT IS SUBSTITUTED.

Explanation: A non-scalar identifier has been specified in a context that requires a scalar identifier.

System Action: Replaces illegal identifier with arithmetic constant zero

S IEM1025I IDENTIFIER zzzz ILLEGALLY USED AS SUBSCRIPT IN STATEMENT NUMBER xxx

Explanation: A subscript has been used which is not a scalar, a scalar expression, or a constant

System Action: Replaces illegal subscript with arithmetic constant zero

W IEM1026I STATEMENT NUMBER xxx IS AN UNLABELED FORMAT STATEMENT

Explanation: A FORMAT statement should have a label

T IEM1027I THE SUBSCRIPTED STRUCTURE ITEM zzzz IS ILLEGALLY USED IN STATEMENT NUMBER xxx

Explanation: The indicated structure item is used in a statement other than an assignment statement or an I/O data list.

System Action: Compilation is terminated

T IEM1028I COMPILER ERROR IN STATEMENT NUMBER xxx. ILLEGAL INPUT TEXT FOR PHASE IA.

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1029I IN STATEMENT NUMBER xxx AN ARRAY CROSS-SECTION APPEARS ILLEGALLY.

Explanation: A feature has been used that is not supported by this version of the compiler. For details, refer to Appendix H of this publication.

System Action: Terminates compilation

T IEM1040I DEFERRED FEATURE. STRUCTURE ARGUMENT IS BEING PASSED TO FUNCTION z zzz IN STATEMENT NUMBER xxx.

Explanation: In this version of the compiler, structures may not be passed as arguments to built-in functions.

System Action: Terminates compilation

User Response: Rewrite program, avoiding unsupported feature.

T IEM1051I DEFERRED FEATURE. STRUCTURE ARGUMENT IS BEING PASSED TO PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx.

Explanation: In this version of the compiler, structures may not be passed as arguments to pseudo-variables.

System Action: Terminates compilation

User Response: Rewrite program, avoiding unsupported feature.

T IEM1056I INVALID ARGUMENT IS BEING PASSED TO ENTRY NAME zzzz IN STATEMENT NUMBER xxx.

System Action: Terminates compilation

T IEM1057I DECIMAL INTEGER CONSTANT IS NOT BEING PASSED, AS REQUIRED, TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Argument to built-in function is not a decimal integer as expected.

System Action: Terminates compilation

T IEM1058I ARRAY OR STRUCTURE ARGUMENT IS NOT BEING PASSED, AS REQUIRED, TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Argument to built-in function is not an array or a structure, as expected.

System Action: Terminates compilation

User Response: Correct statement

T IEM1059I FIRST ARGUMENT BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx SHOULD BE AN ARRAY.

Explanation: Argument to built-in function is not an array as expected

System Action: Terminates compilation

T IEM1060I TOO MANY ARGUMENTS ARE BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Too many arguments are being passed to a built-in function

System Action: Terminates compilation

T IEM1061I TOO FEW ARGUMENTS ARE BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx.

Explanation: Too few arguments are being passed to a built-in function

System Action: Terminates compilation

T IEM1062I COMPILER ERROR. CORRECT GENERIC SELECTION FOR FUNCTION zzzz IN STATEMENT NUMBER xxx HAS NOT BEEN ACHIEVED.

Explanation: Compiler, althcugh being given a legal argument to a generic built-in function, is unable to make the selection.

System Action: Function result is set to zero and compilation terminated.

User Response: Save relevant data. Call your local IBM representative.

T IEM1063I COMPILER ERROR. UNEXPECTED SITUATION HAS ARISEN IN THE SCANNING OF THE ARGUMENTS PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx

Explanation: Compiler is unable to correctly scan an argument list

System Action: Function result is set to zero

User Response: Save relevant data. Call your local IBM representative.

T IEM1064I COMPILER ERROR. THE GENERIC FAMILIES ASSOCIATED WITH ENTRY NAME zzzz HAVE BEEN INCORRECTLY FORMED IN THE DICTIONARY.

Explanation: The dictionary entry for one or more of the generic families is not a recognizable entry type.

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1065I NO GENERIC SELECTION POSSIBLE FOR THE ENTRY NAME zzzz IN STATEMENT NUMBER xxx.

Explanation: Incorrect use of the GENERIC attribute resulting in no selection being possible

System Action: Terminates compilation

T IEM1066I MORE THAN ONE GENERIC SELECTION IS POSSIBLE FOR THE ENTRY NAME zzzz IN STATEMENT NUMBER xxx.

Explanation: Incorrect use of the GENERIC attribute resulting in more than one selection being possible

System Action: Terminates compilation

T IEM1067I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH AN ILLEGAL ARGUMENT.

Explanation: Argument to pseudo-variable cannot be converted to a legal type; or, structure argument being used with pseudo-variable.

System Action: Terminates compilation

T IEM1068I AN ARRAY IS BEING PASSED TO FUNCTION zzzz IN STATEMENT NUMBER xxx. THIS PRODUCES AN ARRAY EXPRESSION WHICH IS INVALID IN THIS CONTEXT.

System Action: Terminates compilation

T IEM1071I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH TOO MANY ARGUMENTS.

System Action: Terminates compilation

T IEM1072I PSEUDO-VARIABLE zzzz APPEARS IN STATEMENT NUMBER xxx WITH TOO FEW ARGUMENTS.

System Action: Terminates compilation

T IEM1073I COMPILER ERROR. CORRECT GENERIC SELECTION FOR PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx HAS NOT BEEN ACHIEVED.

Explanation: Compiler error. Although being given a legal argument to a generic pseudo-variable, is unable to make the selection.

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1074I COMPILER ERROR. UNEXPECTED SITUATION HAS ARISEN IN THE SCANNING OF THE ARGUMENTS PASSED TO PSEUDO-VARIABLE zzzz IN STATEMENT NUMBER xxx

Explanation: Unable to correctly scan an argument list of a pseudo-variable

System Action: Compilation terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1076I COMPILER ERROR IN PHASE JD

System Action: Compilation terminates

User Response: Save relevant data. Call your local IBM representative.

T IEM1078I IMPLEMENTATION RESTRICTION. THE NUMBER OF FAMILY MEMBERS AND ARGUMENTS ASSOCIATED WITH THE GENERIC ENTRY NAME yyyy EXCEEDS THE LIMITATION IMPOSED.

Explanation: There is an implementation restriction on the number of family members and arguments associated with GENERIC entry names. For details, refer to Appendix B of this publication.

System Action: Compilation terminated

User Response: Divide the generic family into two or more generic families.

S IEM1082I STATEMENT NUMBER xxx CONTAINS AN INVALID USE OF AREA OR POINTER DATA. PART OR ALL OF THE STATEMENT HAS BEEN DELETED.

Explanation: The statement contains an operation that:

1. is not permitted for AREA or POINTER data, or
2. can only be used with AREA or POINTER data but such

data is not the data specified for the operation.

System Action: Deletes the statement or clause responsible for the error.

S IEM1088I THE SIZE OF AGGREGATE zzzz IS GREATER THAN 8,388,607 BYTES. STORAGE ALLOCATION WILL BE UNSUCCESSFUL.

Explanation: The message is generated when an array or structure size exceeds $2^{23}-1$

System Action: Array or structure mapping for the item is terminated, but the compilation continues. Execution of object decks containing references to the item will give incorrect results.

User Response: Check source code

S IEM1089I THE RELATIVE VIRTUAL ORIGIN OF AGGREGATE zzzz IS LESS THAN -8,388,608 BYTES. STORAGE HAS NOT BEEN ALLOCATED.

Explanation: The low bounds of the arrays in the aggregate are too high.

System Action: The mapping of the aggregate is undefined, and will cause errors on execution.

User Response: Reduce the size of the aggregate, or reduce the value of the low bounds in the aggregate.

S IEM1090I THE STRUCTURE zzzz DECLARED IN STATEMENT NUMBER xxx CONTAINS VARYING STRINGS AND MAY APPEAR IN A RECORD I/O STATEMENT

Explanation: VARYING strings in structures are not permitted in RECORD I/O statements.

System Action: The RECORD I/O statement is processed but the record will contain erroneous information.

User Response: Correct the source code

W IEM1092I THE TASKS, EVENTS OR LABELS CONTAINED IN STRUCTURE zzzz DECLARED IN STATEMENT NUMBER xxx MAY LOSE THEIR VALIDITY IF USED IN A RECORD I/O STATEMENT.

Explanation: The TASK, EVENT, or LABEL variable may lose its validity in transmission.

User Response: Correct the source code if necessary

E IEM1104I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx INVOLVES DATA NOT ALLOWED FOR STRING CLASS OVERLAY DEFINING.

Explanation: The programmer's use of the DEFINED attribute contravenes the language rules concerned with the permitted data types and dimensionality of base and defined item.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1105I THE DATA CHARACTERISTICS OF zzzz DECLARED IN STATEMENT NUMBER xxx DO NOT MATCH THOSE OF THE DEFINING BASE.

Explanation: For valid use of the DEFINED attribute, both the defined item and the base must be of the same defining class.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1106I THE DIMENSIONALITY OF zzzz DECLARED IN STATEMENT NUMBER xxx IS NOT THE SAME AS THAT OF THE DEFINING BASE.

Explanation: With the exception of the case of string class defining, if either the base or the defined item are arrays, then both the base and the defined item must be arrays with the same dimensionality.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1107I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES VARYING STRINGS.

Explanation: In use of the DEFINED attribute, neither the base nor the defined item may involve strings declared VARYING.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1108I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES DATA AGGREGATES THAT ARE NOT UNALIGNED.

Explanation: In the case of string class overlay defining where either or both the base and the defined item are aggregates, then the aggregates must have the PACKED attribute.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1110I THE DEFINING BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS SHORTER THAN THE DEFINED ITEM.

Explanation: In the case of string class overlay defining, the defined item must occupy a subset of the base storage.

In the case of correspondence defining, the length of each defined element must not be greater than the length of each base element.

System Action: Compilation is aborted after examining other uses of the DEFINED attribute

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1111I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx INVOLVES A STRUCTURE HAVING ELEMENTS NOT ALL OF THE SAME DEFINING CLASS.

Explanation: In the case of string class overlay defining where the defined item or the base is a structure, then all the elements of the structure must be data of the same string defining class.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

T IEM1112I THE DEFINING OF zzzz DECLARED IN STATEMENT NUMBER xxx ILLEGALLY INVOLVES THE POS ATTRIBUTE.

Explanation: The POSITION attribute may only be declared for data of the string class which is overlay defined

System Action: Compilation is terminated

User Response: Refer to the PL/I Reference Manual - "The DEFINED Attribute" - and correct the error.

E IEM1113I THE STRUCTURE DESCRIPTION OF zzzz DECLARED IN STATEMENT NUMBER xxx DOES NOT MATCH THAT OF THE DEFINING BASE.

Explanation: Where a structure or an array of structures is defined on a structure or an array of structures, and it is not string class overlay defining, then the two structure descriptions must be identical.

System Action: Defined item mapped onto same storage as item defined on. Data and specification interrupts may occur at execution.

User Response: Refer to the PL/I Reference Manual - "The

DEFINED Attribute" - and correct the error.

W IEM1114I IF THE BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS ALLOCATED WITH THE DECLARED EXTENTS, THE DEFINING WILL BE IN ERROR.

Explanation: In the case of string class overlay defining, the defined item must occupy a subset of the base storage. If the base is of CONTROLLED storage class, its extents are not finally resolved until execution time.

System Action: No further action

User Response: Check that when the base is allocated it is of adequate size to accommodate the defined item

E IEM1115I THE DEFINING BASE OF zzzz DECLARED IN STATEMENT NUMBER xxx IS AN ARRAY FORMAL PARAMETER. IF THE MATCHING ARGUMENT IS AN ELEMENT OF AN ARRAY OF STRUCTURES OR A CROSS SECTION OF AN ARRAY, THE DEFINING WILL BE IN ERROR.

Explanation: The base for string class overlay defining must occupy contiguous storage

System Action: Comments and continues

User Response: Check validity of arguments

S IEM1120I COMPILER ERROR. INVALID SIGN FOUND IN INITIAL VALUE LIST FOR zzzz IN STATEMENT NUMBER xxx. TREATED AS PLUS.

User Response: Save relevant data. Call your local IBM representative.

S IEM1121I COMPILER ERROR. INVALID MARKER FOUND IN INITIAL VALUE LIST FOR zzzz IN STATEMENT NUMBER xxx. INITIAL VALUE LIST TRUNCATED.

User Response: Save relevant data. Call your local IBM representative.

S IEM1122I UNSUPPORTED FEATURE. AN EXPRESSION HAS BEEN USED TO INITIALIZE STATIC STRING zzzz IN STATEMENT NUMBER xxx. STRING INITIALIZED TO NULL.

Explanation: A complex expression has been used to initialize a STATIC string. This is a feature of PL/I not supported by this version of the compiler. See Appendix H of this publication for details.

System Action: The string is initialized to null.

User Response: Amend source code. The restriction can be overcome by using an assignment statement instead of the INITIAL attribute.

S IEM1123I INITIAL VALUE FOR STATIC DATA ITEM zzzz IN nnnn IS NOT A CONSTANT. INITIALIZATION TERMINATED.

User Response: Use a constant in the INITIAL string.

S IEM1125I ITERATION FACTOR USED IN INITIALIZATION OF STATIC ARRAY zzzz IN STATEMENT NUMBER xxx IS TOO LARGE. REPLACED BY ZERO.

Explanation: Iteration factors are converted by the compiler to REAL FIXED BINARY with a default precision of 15,0. The iteration factor referred to in the message has a value greater than 2^{15}, and therefore exceeds the default precision.

System Action: The iteration factor is replaced by zero

User Response: Amend source code so that iteration factor does not exceed 2^{15}.

T IEM1200I COMPILER ERROR. ILLEGAL TRIPLE IN TEXT. CURRENT STATEMENT NUMBER xxx

Explanation: Phase LA is out of step in scanning text

System Action: Recovery impossible. Compilation is terminated.

User Response: Save relevant data. Call your local IBM representative.

T IEM1569I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: The number of symbolic register names generated by the code generation section of the compiler has

exceeded the maximum number allowed

System Action: Compilation is terminated

User Response: Programmer should break down the compilation into smaller modules

T IEM1570I COMPILER ERROR. INVALID TRIPLE FOLLOWING WHILE PRIME TRIPLE.

Explanation: Input to phase LG of compiler is erroneous. A WHILE' triple is not followed by CV' or compiler label.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1571I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: No more core is available for the stack of nested DO statements (both in source language and those generated internally for array assignments etc.)

System Action: Compilation is terminated

User Response: Simplify nesting so as to reduce number of levels.

S IEM1572I ILLEGAL USE OF ARRAY OR STRUCTURE VARIABLE IN DO STATEMENT NUMBER xxx

Explanation: A non-scalar variable has been used as (1) the control variable, or (2) a control variable subscript, or (3) a loop limit or increment value.

System Action: Generates an error stop at execution time

S IEM1574I INVALID LOOP CONTROL EXPRESSION OR CONTROL VARIABLE SUBSCRIPT IN STATEMENT NUMBER xxx REPLACED BY FIXED BINARY TEMPORARY.

Explanation: Either something other than an arithmetic or string datum has been used as a subscript in the control variable, or something other than an arithmetic or string datum, label variable, or label constant has been used in an initial value, TO or BY clause.

System Action: Ignore the erroneous expression and use a fixed binary temporary

S IEM1575I DO LOOP CONTROL PSEUDO-VARIABLE IN STATEMENT NUMBER xxx HAS AN INVALID ARGUMENT. BINARY INTEGER TEMPORARY ASSUMED.

Explanation: An invalid argument, such as an expression or function, has been used in a pseudo-variable.

System Action: Assigns invalid argument to binary temporary, and uses the latter as argument.

W IEM1588I VARYING STRING HAS BEEN USED AS AN ARGUMENT TO ADDR FUNCTION IN STATEMENT NUMBER xxx

Explanation: If the VARYING string is declared BASED, the ADDR result is invalid

System Action: None

User Response: Correct source program

T IEM1600I COMPILER ERROR. ILLEGAL ABSOLUTE REGISTER NUMBER. STATEMENT NUMBER xxx

Explanation: Compiler error. Fixed binary arithmetic uses an unassigned general register number greater than 15, or floating arithmetic uses a floating register greater than 6.

System Action: Compilation is terminated and error messages printed.

User Response: Save relevant data. Call your local IBM representative.

T IEM1601I IMPLEMENTATION RESTRICTION. STATEMENT NUMBER xxx REQUIRES MORE THAN 200 INTERMEDIATE RESULT DESCRIPTIONS.

Explanation: Compiler limitation. The temporary result stack, which holds 200 items, is full.

System Action: Compilation is terminated and error messages printed

User Response: This error should only occur in very large statements. Divide the statement into two smaller statements.

T IEM1602I COMPILER ERROR. INSUFFICIENT NUMBER OF TEMPORARY RESULT DESCRIPTIONS. STATEMENT NUMBER xxx

Explanation: Compiler error. A temporary result is required but the temporary result stack is empty. This can happen if the triples are out of order or if extra triples have been inserted.

System Action: Compilation is aborted and error messages printed

User Response: Save relevant data. Call your local IBM representative.

T IEM1603I COMPILER ERROR. COUNT OF FREE FLOATING REGISTERS IS WRONG. STATEMENT NUMBER xxx

Explanation: Compiler error in expression evaluation phase. Error in control blocks for floating registers.

System Action: Compilation is terminated and error messages printed

User Response: Save relevant data. Call your local IBM representative.

T IEM1604I COMPILER ERROR. SECOND OPERAND FOR RS OR SS INSTRUCTION IS IN A REGISTER. STATEMENT NUMBER xxx

Explanation: Compiler error in expression evaluation phase. Attempt to generate an RS or SS type pseudo-code instruction using a register as the 2nd operand.

System Action: Compilation is terminated and error messages printed.

User Response: Save relevant data. Call your local IBM representative.

S IEM1605I IN STATEMENT NUMBER xxx FIXED DECIMAL VARIABLE CANNOT BE CORRECTLY CONVERTED TO BINARY DUE TO SIZE OF SCALE FACTOR.

Explanation: Error in source program. When a fixed decimal variable is corrected to fixed binary, the magnitude of its scale factor is multiplied by 3.31. If the original scale factor is >38 or <-38, then the fixed binary scale factor would be outside the range +127 to -128.

System Action: The fixed binary scale factor is set to +127 or -128. Processing continues.

User Response: The data in the expression must be re-declared with more suitable scale factors.

T IEM1606I COMPILER ERROR. FUNCTION NOT FOLLOWED BY RESULT DESCRIPTION. STATEMENT NUMBER xxx

Explanation: Compiler error. A function is not followed by TMPD or LEFT triples giving the result type.

System Action: Compilation is terminated and error messages printed

User Response: Save relevant data. Call your local IBM representative.

S IEM1607I LABEL, EVENT, FILE, OR TASK ITEM zzzz IN STATEMENT NUMBER xxx IS USED IN AN EXPRESSION WHICH IS ILLEGAL.

Explanation: Error in source program. A label, event, file, or task datum cannot be used in an expression. Alternatively, this can be a compiler error when an unrecognizable dictionary entry is used in an expression.

System Action: Substitute a fixed binary (31,0) data item (if the illegal item occurs in an arithmetic expression) or a null bit string (if it occurs in a string expression). Processing is continued.

User Response: If error in source program, correct it.

E IEM1608I LT, LE, GE, OR GT COMPARISON OPERATOR ILLEGALLY USED IN STATEMENT NUMBER xxx WITH COMPLEX OPERANDS. REPLACED WITH EQUALS OPERATOR.

Explanation: Error in source program. The only legal comparison between complex operands is '='.

System Action: The operator is replaced with '=' and processing continues

User Response: Correct source program using either the ABS function or possibly the REAL and IMAG functions.

T IEM1609I COMPILER ERROR. ILLEGAL DICTIONARY REFERENCE X'00..' . STATEMENT NUMBER xxx

Explanation: Compiler error. The symbolic dictionary reference is less than 256.

System Action: Compilation terminated and error messages printed

User Response: Save relevant data. Call your local IBM representative.

T IEM1610I COMPILER ERROR IN PHASE LW AT STATEMENT NUMBER xxx. INSUFFICIENT NUMBER OF TEMPORARY RESULT DESCRIPTIONS.

Explanation: Compiler error. A temporary result is required but the temporary result stack is empty. This can happen if the triples are out of order or if extra triples have been inserted.

System Action: Compilation is terminated and error messages printed

User Response: Save relevant data. Call your local IBM representative.

E IEM1611I IMPLEMENTATION RESTRICTION. A STRING RESULT LONGER THAN 32767 IS PRODUCED BY CONCATENATE IN STATEMENT NUMBER xxx. STRING TRUNCATED TO LENGTH 32767.

Explanation: Maximum string length for this implementation is 32767. This may be exceeded during concatenation, because the length of the intermediate result is the sum of the operand lengths.

System Action: Compilation continues with string result length truncated to 32767

User Response: Shorter strings must be used

W IEM1612I IMPLEMENTATION RESTRICTION IN STATEMENT NUMBER xxx. INTERMEDIATE WORK SPACE IS OBTAINED MORE THAN 50 TIMES IN A STRING EXPRESSION. SOME WORK SPACE WILL NOT BE RELEASED UNTIL THE END OF THE BLOCK.

Explanation: The intermediate work space is required each time a function returns a string result or each time a library module is called

System Action: The first 50 areas of work space are released. The remainder may not be released until the end of the block. Compilation continues and execution is valid.

User Response: Divide the string expression into several sub-expressions

S IEM1613I ILLEGAL USE OF ARRAY OR STRUCTURE VARIABLE IN STATEMENT NUMBER xxx

Explanation: Illegal source program

System Action: Severe error message and object program branch. Compilation continues, assuming scalar of same type if array, or fixed binary (31,0) type if structure.

User Response: Insert DO blocks for array, or break down structure into its components.

W IEM1614I IMPLEMENTATION RESTRICTION. A VARYING STRING RESULT LONGER THAN 32767 MAY BE PRODUCED BY CONCATENATE IN STATEMENT NUMBER xxx. STRING TRUNCATED TO LENGTH 32767.

Explanation: The sum of the maximum lengths of two strings in a concatenation operation exceeds the implementation restriction of 32767. Since one or both of the operands is a VARYING string, it is not known at compile-time whether the

restriction will be exceeded at execution time.

System Action: Compilation continues with string result maximum length truncated to 32767.

User Response: Shorter strings must be used if the sum of the execution-time current lengths will ever exceed 32767.

E IEM1615I SECOND ARGUMENT IN THE SUBSTR FUNCTION IN STATEMENT NUMBER xxx IS ZERO, WHICH IS INVALID. ZERO HAS BEEN REPLACED BY ONE.

E IEM1616I SECOND ARGUMENT IN THE SUBSTR PSEUDO-VARIABLE IN STATEMENT NUMBER xxx IS ZERO, WHICH IS INVALID. ZERO HAS BEEN REPLACED BY ONE.

T IEM1617I COMPILER ERROR. ILLEGAL RETURN FROM SCAN ROUTINE. STATEMENT NUMBER xxx

Explanation: An illegal return of control has been made by the SCAN routine which supports the code generation phases.

System Action: Compilation is terminated.

User Response: Save relevant data. Call your local IBM representative.

S IEM1618I PSEUDO-VARIABLE IN STATEMENT NUMBER xxx INCORRECTLY SPECIFIED. REPLACED BY FIXED BINARY TEMPORARY.

Explanation: A pseudo-variable in the given source statement has been incorrectly specified, e.g. has an incorrect number of arguments.

System Action: Ignores the pseudo-variable and uses a fixed binary temporary instead.

S IEM1619I RIGHT HAND SIDE OF STATEMENT NUMBER xxx CANNOT BE ASSIGNED TO A PSEUDO-VARIABLE. ASSIGNMENT IGNORED.

Explanation: The expression on the right-hand side of the specified statement cannot be assigned to a pseudo-variable, i.e. it is not an arithmetic or string datum.

System Action: The assignment is deleted from the text.

S IEM1620I 'IMAG' IN STATEMENT NUMBER xxx HAS REAL ARGUMENT. REPLACED BY ASSIGNMENT TO TEMPORARY FIXED BINARY INTEGER.

Explanation: The pseudo-variable 'IMAG' is meaningful only if its argument is of type complex.

System Action: A fixed binary temporary target is provided for the assignment or input data list item and the pseudo-variable is ignored

S IEM1621I ILLEGAL PSEUDO-VARIABLE ARGUMENT IN STATEMENT NUMBER xxx REPLACED BY BINARY TEMPORARY.

Explanation: A pseudo-variable in the specified statement has an illegal argument, i.e. one whose data type is not permissible in that context.

System Action: A temporary whose type is legal in the context is used to replace the erroneous argument and the latter is removed from the text

S IEM1622I FIRST ARGUMENT OF PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS NOT A STRING VARIABLE. ARGUMENT HAS BEEN CONVERTED TO STRING TEMPORARY AND THE ASSIGNMENT MADE THERETO.

Explanation: SUBSTR pseudo-variable cannot have a first argument which is not a string variable.

System Action: Code is compiled to assign to a string temporary. The original argument remains unchanged.

T IEM1623I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: The number of symbolic register names generated by the code generation section of the compiler has exceeded the maximum number allowed

System Action: Compilation is terminated

User Response: Programmer

should break down the compilation into smaller modules

W IEM1625I PSEUDO-VARIABLE REAL IN STATEMENT NUMBER xxx DOES NOT HAVE COMPLEX ARGUMENT. ARGUMENT HAS BEEN TREATED AS HAVING ZERO IMAGINARY PART.

System Action: Code is generated to perform assignment to the specified REAL argument

S IEM1626I ILLEGAL NEGATIVE SECOND ARGUMENT IS BEING PASSED TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1627I ILLEGAL NEGATIVE THIRD ARGUMENT IS BEING PASSED TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1628I THE SUBSTRING SPECIFIED BY THE SECOND AND THIRD ARGUMENTS TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx DOES NOT LIE WITHIN THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

S IEM1629I THE SECOND ARGUMENT TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

T IEM1630I COMPILER ERROR IN CEIL/FLOOR/TRUNC IN-LINE FUNCTION IN STATEMENT NUMBER xxx

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1631I COMPILER ERROR IN MOD IN-LINE FUNCTION IN STATEMENT NUMBER xxx

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

W IEM1632I THE INVOCATION OF THE ROUND FUNCTION IN STATEMENT NUMBER xxx WILL ALWAYS GIVE A ZERO RESULT.

Explanation: (p - q + r) is zero or negative, where p = precision, q = scale factor, and r = rounding position.

System Action: Result is set to zero

User Response: Check scale and precision of the first argument in ROUND function

S IEM1633I ILLEGAL NEGATIVE SECOND ARGUMENT IS BEING PASSED TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1634I ILLEGAL NEGATIVE THIRD ARGUMENT IS BEING PASSED TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx. AN EXECUTION ERROR WILL RESULT.

S IEM1635I THE SUBSTRING SPECIFIED BY THE SECOND AND THIRD ARGUMENTS TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx DOES NOT LIE WITHIN THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

S IEM1636I THE SECOND ARGUMENT TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

S IEM1637I THE THIRD ARGUMENT TO THE FUNCTION SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE FIRST ARGUMENT. AN EXECUTION ERROR WILL RESULT.

S IEM1638I THE THIRD ARGUMENT TO THE PSEUDO-VARIABLE SUBSTR IN STATEMENT NUMBER xxx IS GREATER THAN THE LENGTH OF THE STRING zzzz. AN EXECUTION ERROR WILL RESULT.

T IEM1639I COMPILER ERROR. INCORRECT INPUT TO SUBROUTINE 6 IN MODULE IEMMF IN STATEMENT NUMBER xxx.

System Action: Compilation is terminated

User Response: Save relevant data and call your local IBM representative.

T IEM1640I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz DOES NOT MATCH ANY OF THE FAMILY MEMBERS.

System Action: Terminates compilation

User Response: Provide correct parameter description

W IEM1641I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz DESCRIBES THE ENTRY NAME'S RESULT TYPE RATHER THAN ARGUMENT TYPE. IF POSSIBLE, GENERIC SELECTION WILL BE MADE ON THE BASIS OF THIS RESULT TYPE.

User Response: Provide fuller parameter description

T IEM1642I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz IS NOT SUFFICIENT FOR THE PURPOSES OF GENERIC SELECTION.

System Action: Terminates compilation

User Response: Provide fuller parameter description

T IEM1643I COMPILER ERROR. THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz IS INCORRECTLY FORMED IN THE DICTIONARY.

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1644I COMPILER ERROR. THE GENERIC FAMILIES ASSOCIATED WITH ENTRY NAME zzzz HAVE BEEN INCORRECTLY FORMED IN THE DICTIONARY.

Explanation: The dictionary entry for one or more of the generic families is not a recognizable entry type.

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1645I THE PARAMETER DESCRIPTION RELATING TO THE PASSING OF THE GENERIC ENTRY NAME zzzz RESULTS IN MORE THAN ONE POSSIBLE FAMILY MEMBER SELECTION.

System Action: Terminates compilation

User Response: Provide fuller parameter description

T IEM1648I COMPILER ERROR. FUNCTION REFERENCE MISSING FROM TEXT IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase.

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1649I COMPILER ERROR. INCORRECT FORMATION OF ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1650I COMPILER ERROR. INCORRECT HANDLING OF ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1651I COMPILER ERROR. ARGUMENT REFERENCE MISSING FROM ARGUMENT LIST ASSOCIATED WITH ENTRY NAME zzzz IN STATEMENT NUMBER xxx

Explanation: Incorrect handling of text by previous phase

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

T IEM1652I IMPLEMENTATION RESTRICTION. INVOCATIONS ARE NESTED BEYOND THE MAXIMUM PERMITTED LEVEL IN STATEMENT NUMBER xxx

Explanation: Nesting level exceeds implementation limit

System Action: Terminals compilation

User Response: Reduce nesting level

T IEM1653I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE. NUMBER OF SYMBOLIC REGISTERS EXCEEDS LIMIT.

Explanation: Too many symbolic registers required

System Action: Terminates compilation

User Response: Subdivide the program

T IEM1654I THE GENERIC PROCEDURE zzzz IS BEING INVOKED WITHOUT AN ARGUMENT LIST IN STATEMENT NUMBER xxx

System Action: Terminates compilation

User Response: Supply argument list

T IEM1655I IMPLEMENTATION RESTRICTION. TOO MUCH WORKSPACE REQUIRED FOR TEMPORARY RESULTS IN STATEMENT NUMBER xxx

System Action: Terminates compilation

User Response: Subdivide the statement in question into two or more separate statements.

T IEM1656I COMPILER ERROR. INCORRECT INPUT TO PHASE MF FOR COMPLETION BUILT-IN FUNCTION IN STATEMENT NUMBER xxx.

Explanation: The compiler has encountered incorrect input to phase MF.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM1657I THE FILE zzzz, WHICH HAS BEEN DECLARED WITH THE COBOL OPTION, IS BEING PASSED AS AN ARGUMENT IN STATEMENT NUMBER xxx.

Explanation: F Compiler restriction: files with the COBOL option may not be passed as arguments.

System Action: Comment and continue

User Response: Correct source program if necessary

T IEM1670I STATEMENT NUMBER xxx HAS CAUSED A TABLE INTERNAL TO THE COMPILER TO OVERFLOW.

Explanation: Either the nesting of procedure arguments requiring dummies is too deep, or too many temporary results are required between the assignment of an argument expression to a dummy and the procedure call.

System Action: Compilation is terminated

User Response: Reduce complexity of argument expressions

T IEM1671I COMPILER ERROR NUMBER MP nnnn IN STATEMENT NUMBER xxx

Explanation: This is a compiler error

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1680I COMPILER ERROR. TRIPLE OPERATOR NOT RECOGNIZED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1687I COMPILER ERROR. OPTIMIZED SUBSCRIPT INCORRECTLY FORMED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1688I COMPILER ERROR. ARRAY NAME

zzzz INCORRECTLY DESCRIBED AS DEFINED IN STATEMENT NUMBER xxx

Explanation: Array incorrectly described by a previous phase as having the DEFINED attribute

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1689I COMPILER ERROR. ARRAY zzzz IS INCORRECTLY SUBSCRIPTED IN STATEMENT NUMBER xxx

Explanation: Illegal input from a previous phase

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM1691I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: Too many symbolic registers have been requested

System Action: Compilation is terminated

User Response: Subdivide program and recompile

T IEM1692I IMPLEMENTATION RESTRICTION. SUBSCRIPT NESTED TO DEPTH GREATER THAN 50 LEVELS IN STATEMENT NUMBER xxx

Explanation: Subscript nesting exceeds fifty levels

System Action: Compilation is terminated

User Response: Reduce amount of nesting and recompile

T IEM1693I NUMBER OF SUBSCRIPTS ASSOCIATED WITH ARRAY zzzz IN STATEMENT NUMBER xxx IS INCORRECT.

Explanation: The number of subscripts given does not agree with the declared dimensionality of the array.

System Action: Compilation is terminated

User Response: Add or delete subscripts as appropriate

S IEM1750I zzzz IS AN ILLEGAL OPERAND IN AN IF STATEMENT OR WHILE CLAUSE IN STATEMENT NUMBER xxx. IT HAS BEEN REPLACED BY A ZERO BIT STRING.

S IEM1751I THE IDENTIFIER zzzz IS AN ILLEGAL ARGUMENT OF THE RETURN STATEMENT NUMBER xxx AND HAS BEEN DELETED.

Explanation: Illegal arguments include arrays and structures.

W IEM1752I THE ATTRIBUTES OF THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx CONFLICT WITH THE ATTRIBUTES OF SOME OR ALL OF THE ENTRY POINTS OF THE CONTAINING PROCEDURE. AN EXECUTION FAILURE MAY OCCUR AT THIS STATEMENT.

Explanation: After a call to a procedure through an entry point with POINTER, AREA or data attributes, any RETURN statement encountered must return a value of type POINTER or AREA or of a data type compatible with the data attributes of the entry point.

System Action: The ERROR condition is raised

E IEM1753I THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx AND THE ATTRIBUTES OF THE CONTAINING PROCEDURE ARE INCOMPATIBLE. EXECUTION OF THIS STATEMENT WILL RESULT IN A FAILURE.

Explanation: After a call to a procedure through an entry point with POINTER, AREA or data attributes, any RETURN statement encountered must return a value of type POINTER or AREA or of a data type compatible with the data attributes of the entry point.

System Action: The ERROR condition is raised

E IEM1754I THE EXPRESSION USED IN THE RETURN STATEMENT IN STATEMENT NUMBER xxx IS INVALID

Explanation: The only permitted arguments are data types STRING, POINTER, and AREA.

System Action: Raise ERROR condition on execution of the statement.

W IEM1790 DATA CONVERSIONS WILL BE DONE BY SUBROUTINE CALL IN THE FOLLOWING STATEMENTS yyyy

User Response: Check to see if the conversion can be avoided or performed in line

S IEM1793I ILLEGAL ASSIGNMENT OR CONVERSION IN STATEMENT NUMBER xxx. EXECUTION WILL RAISE THE ERROR CONDITION.

Explanation: Illegal assignment or conversion in source statement, e.g. label to arithmetic.

System Action: An instruction is compiled which will cause execution to abort if the statement is executed

T IEM1794I COMPILER ERROR IN STATEMENT NUMBER xxx PHASE OE.

Explanation: Compiler error caused by input text in bad format

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM1795I INVALID ITEM IN FREE STATEMENT NUMBER xxx

Explanation: Variable in FREE statement is either not CONTROLLED or not at level 1

System Action: Error condition and message given at object time

E IEM1796I ASSIGNMENT OF AN ILLEGAL LABEL CONSTANT IN STATEMENT NUMBER xxx.

Explanation: The label constant does not appear in the value list in the DECLARE statement for the label variable.

System Action: Accepts label constant as if in value list and continues compilation.

W IEM1797I CONVERSION OF NULL VALUES IN POINTER/OFFSET ASSIGNMENTS IS INVALID. NULLO HAS BEEN REPLACED BY NULL, OR NULL BY NULLO, IN STATEMENT NUMBER xxx

Explanation: A NULLO offset type constant has been assigned to a pointer, or a NULL pointer type constant to an offset. Conversion of null values is not allowed. The constant type has been corrected.

System Action: The assignment is unaffected

S IEM1800I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FLOATING-POINT. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

System Action: Truncates result.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1801I AN ERROR HAS OCCURRED IN THE CONVERSION TO FLOATING-POINT OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1802I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FIXED BINARY. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1803I AN ERROR HAS OCCURRED IN THE CONVERSION TO FIXED BINARY OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1804I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO FIXED DECIMAL. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

System Action: Truncates result.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1805I AN ERROR HAS OCCURRED IN THE CONVERSION TO FIXED DECIMAL OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USE OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1806I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO DECIMAL NUMERIC FIELD. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1807I AN ERROR HAS OCCURRED IN THE CONVERSION TO DECIMAL NUMERIC FIELD OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Esponse: Change the constant and check its use in the given statement and elsewhere.

S IEM1808I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO STERLING NUMERIC FIELD. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1809I AN ERROR HAS OCCURRED IN THE CONVERSION TO STERLING NUMERIC FIELD OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1810I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO BIT STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1811I AN ERROR HAS OCCURRED IN THE CONVERSION TO BIT STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1812I AN ERROR HAS OCCURRED WHEN CONVERTING THE CONSTANT yyyy TO CHARACTER STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1813I AN ERROR HAS OCCURRED IN THE CONVERSION TO CHARACTER STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1814I AN ERROR HAS OCCURRED IN THE CONVERSION OF THE CONSTANT yyyy TO PICTURED CHARACTER STRING. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1815I AN ERROR HAS OCCURRED IN THE CONVERSION TO PICTURED CHARACTER STRING OF THE STERLING CONSTANT WHICH HAS DECIMAL PENCE FORM yyyy. THE ERROR WAS DETECTED IN STATEMENT NUMBER xxx BUT CHECK ALL SIMILAR USES OF THIS CONSTANT.

User Response: Change the constant and check its use in the given statement and elsewhere.

S IEM1816I zzzz USED IN FILE OPTION IN STATEMENT NUMBER xxx IS NOT A FILE. OPTION HAS BEEN IGNORED. EXECUTION ERROR WILL RESULT

Explanation: Dictionary reference of file triple was not file constant or file parameter code.

System Action: Ignores option, but continues to scan statement.

E IEM1817I INVALID KEYTO OPTION zzzz IGNORED IN STATEMENT NUMBER xxx

Explanation: KEYTO option must be scalar character string variable.

S IEM1818I zzzz USED IN KEY/KEYFROM OPTION IN STATEMENT NUMBER xxx IS NOT A SCALAR. OPTION IGNORED.

System Action: Ignores option but continues scan of statement

S IEM1819I zzzz USED IN THE IGNORE OPTION IN STATEMENT NUMBER xxx IS NOT A SCALAR. OPTION IGNORED.

System Action: Ignores option but continues scan of statement

User Response: Correct IGNORE variable

T IEM1823I COMPILER ERROR DETECTED IN PHASE NJ/NK.

Explanation: NJ/NK found some unexpected input. Register 9 in dump will indicate cause of error.

System Action: Terminates compilation

User Response: Save relevant data. Call your local IBM representative.

S IEM1824I OPTIONS IN OPEN STATEMENT NUMBER xxx ARE IN CONFLICT WITH PAGESIZE AND/OR LINESIZE.

E IEM1825I INVALID REPLY OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1826I INVALID MESSAGE IN DISPLAY STATEMENT NUMBER xxx. STATEMENT IGNORED.

S IEM1827I INVALID ARGUMENT TO DELAY STATEMENT NUMBER xxx. STATEMENT IGNORED.

T IEM1828I COMPILER ERROR. INCORRECT NUMBER OF TMPDS FOLLOWING ZERO OPERAND IN STATEMENT NUMBER xxx

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM1829I INVALID SCALAR EXPRESSION OPTION IN WAIT STATEMENT NUMBER xxx. MAXIMUM EVENT COUNT GIVEN.

Explanation: The optional scalar expression in the WAIT statement cannot be converted to an integer.

System Action: The number of event names in the list is assumed as the event count.

T IEM1830I COMPILER ERROR. INCORRECT INPUT TO PHASE NG IN WAIT STATEMENT NUMBER xxx.

Explanation: The compiler has encountered incorrect input to phase NG and cannot continue.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM1832I INVALID PAGE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1833I INVALID LINE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1834I MULTIPLE COPY OPTIONS SPECIFIED IN STATEMENT NUMBER xxx. THE FIRST ONE IS USED.

System Action: The first option only is used

E IEM1835I INVALID FILE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1836I INVALID STRING OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1837I NO FILE OR STRING SPECIFIED IN STATEMENT NUMBER xxx. STATEMENT IGNORED.

Explanation: No FILE or STRING given in GET/PUT statement

E IEM1838I INVALID TITLE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1839I INVALID IDENT OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1840I INVALID LINESIZE OPTION IGNORED IN STATEMENT NUMBER xxx

E IEM1841I INVALID PAGESIZE OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1843I NO FILE SPECIFIED IN OPEN/CLOSE STATEMENT NUMBER xxx. ANY OPTIONS ARE IGNORED.

T IEM1844I COMPILER ERROR. INCORRECT NUMBER OF TMPDS FOLLOWING ZERO OPERAND IN STATEMENT NUMBER xxx

System Action: Compilation is aborted

User Response: Save relevant data. Call your local IBM representative.

E IEM1845I MULTIPLE DATA SPECIFICATIONS IGNORED IN STATEMENT NUMBER xxx

E IEM1846I INVALID SKIP OPTION IGNORED IN STATEMENT NUMBER xxx

S IEM1847I NO DATA SPECIFICATIONS GIVEN FOR GET STATEMENT NUMBER xxx. STATEMENT DELETED.

S IEM1848I NO DATA SPECIFICATIONS OR PRINT OPTIONS GIVEN FOR PUT STATEMENT NUMBER xxx. STATEMENT DELETED.

W IEM1849I THE USE OF THE BUILT-IN FUNCTION NULL IN STATEMENT NUMBER xxx IS INVALID; NULLO HAS BEEN SUBSTITUTED. CHECK ALL SIMILAR USES OF NULL.

System Action: Substitute NULLO

W IEM1850I THE USE OF THE BUILT-IN FUNCTION NULLO IN STATEMENT NUMBER xxx IS INVALID; NULL HAS BEEN SUBSTITUTED. CHECK ALL SIMILAR USES OF NULLO.

System Action: Substitute NULL

S IEM1860I THE ILLEGAL ITEM zzzz HAS BEEN DELETED FROM THE I/O DATA LIST IN STATEMENT NUMBER xxx

S IEM1861I AN ILLEGAL TEMPORARY RESULT OR SUBSCRIPTED ELEMENT HAS BEEN DELETED FROM THE I/O DATA LIST IN STATEMENT NUMBER xxx

S IEM1862I AN EXPRESSION OR FUNCTION INVOCATION IS AN ILLEGAL DATA ITEM AND HAS BEEN DELETED FROM THE DATA-DIRECTED I/O STATEMENT NUMBER xxx.

E IEM1870I THE FORMAT LIST IN STATEMENT NUMBER xxx CONTAINS NO DATA FORMAT ITEMS AND WILL BE EXECUTED ONCE IF THE STATEMENT IS INVOKED.

System Action: At execution time, on finding no data format items, control passes out of the statement at the end of the format list.

S IEM1871I IN STATEMENT NUMBER xxx THE FORMAT LIST CONTAINS AN E OR F FORMAT ITEM WITH AN ILLEGAL SPECIFICATION. THE FORMAT ITEM HAS BEEN DELETED.

W IEM1872I IN STATEMENT NUMBER xxx AN E FORMAT ITEM HAS A FIELD WIDTH WHICH WOULD NOT PERMIT PRINTING OF A MINUS SIGN.

S IEM1873I IMPLEMENTATION RESTRICTION. IN STATEMENT NUMBER xxx AN A, B OR CONTROL FORMAT ITEM SPECIFIES AN EXCESSIVE LENGTH WHICH HAS BEEN REPLACED BY THE MAXIMUM OF 32,767.

E IEM1874I IN STATEMENT NUMBER xxx AN INPUT STATEMENT CONTAINS A FORMAT ITEM WHICH MAY BE USED ONLY IN OUTPUT STATEMENTS.

Explanation: PAGE, SKIP, LINE, COLUMN, and format items A and B with no width specification, may be used only for output.

System Action: Invalid format item deleted

W IEM1875I IN STATEMENT NUMBER xxx AN E FORMAT ITEM HAS AN ILLEGAL SPECIFICATION IF USED FOR AN OUTPUT DATA ITEM.

Explanation: The specification violates the restriction that the field width w must be greater than s+n+2.

System Action: There will be an error at execution time.

User Response: Correct specification

T IEM2304I COMPILER ERROR. DICTIONARY ENTRY zzzz UNRECOGNIZED IN STATIC CHAIN.

Explanation: Due to a compiler error, a dictionary entry with an unrecognized code byte has been found in the static chain.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM2305I COMPILER ERROR. DOPE VECTOR REQUESTED BY NON-STRING, NON-STRUCTURE MEMBER zzzz

Explanation: Due to a compiler error, the allocation of a dope vector has been requested for an item which should never require one.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM2352I THE AUTOMATIC VARIABLES IN THE BLOCK HEADED BY STATEMENT NUMBER xxx ARE MUTUALLY DEPENDENT. STORAGE CANNOT BE ALLOCATED.

Explanation: This message is generated when a number of automatic variables are mutually dependent. It is not then possible to allocate storage in order of dependency.

System Action: Compilation is terminated

User Response: Rewrite statement, eliminating mutual dependency.

T IEM2700I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. SPECIAL ASSIGNED REGISTER IN FORMAT/DATA LIST CODE CANNOT BE FOUND.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM2701I COMPILER ERROR. PHASE IEMRF, STATEMENT NUMBER xxx. PSTOR GREATER THAN 32K.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM2702I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. BCT WITHOUT DICTIONARY REFERENCE AS DESTINATION.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM2703I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. LINK REGISTER IN BALR IS NOT ASSIGNED.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM2704I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. 'USNG' ITEM DOES NOT HAVE ASSIGNED REGISTER.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM2705I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. DROPPED REGISTER NOT ACTIVE.

Explanation: Register number in field in DROP item is not in register table nor in storage.

System Action: Continues compilation, ignoring DROP. Execution is inhibited.

User Response: Save relevant data. Call your local IBM representative.

S IEM2706I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. NOT ALL REGISTERS IN 'DRPL' ITEM CAN BE FOUND.

System Action: Ignores DRPL item and continues

User Response: Save relevant data. Call your local IBM representative.

S IEM2707I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. NOT ALL SYMBOLIC REGISTERS DROPPED AT END OF PROCEDURE OR BEGIN BLOCK.

Explanation: One or more symbolic registers have been used in the PROCEDURE or BEGIN block, but no corresponding DROP has occurred.

System Action: Inserts in listing at end of block: the register number, the offset from register 9 at which the register is stored, and the words 'ERROR STOP'.

User Response: Save relevant data. Call your local IBM representative.

S IEM2708I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. ASSIGNED REGISTER USED IN SOURCE FIELD IS NOT INITIALIZED.

Explanation: The assigned register should have a previous value (e.g. X in AR X,Y or L Y,10(X) etc.), but none can be found.

System Action: Register 13 is used instead of the correct number, and compilation is continued.

User Response: Save relevant data. Call your local IBM representative.

S IEM2709I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. SYMBOLIC REGISTER SHOULD HAVE PREVIOUS VALUE, BUT HAS NOT.

Explanation: Register X in an instruction such as AR X,Y, or L Y,10(X), has not been set up previously.

System Action: Inserts register 12 and continues compilation.

User Response: Save relevant data. Call your local IBM representative.

T IEM2710I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. MORE THAN ONE REGISTER PAIR REQUIRED IN AN INSTRUCTION.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM2711I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. ASSIGNED REGISTER IS STILL IN USE AT THE START OF A PROCEDURE.

Explanation: Assigned register status should be zero at the start of each procedure.

System Action: Drops the assigned register and continues compilation

User Response: Save relevant data. Call your local IBM representative.

S IEM2712I COMPILER ERROR IN INPUT TO PHASE IEMRF, STATEMENT NUMBER xxx. IPRM/IPRM' OR EPRM/EPRM' PAIRS ARE NOT MATCHED IN PREVIOUS STATEMENT.

System Action: Compilation is continued

User Response: Save relevant data. Call your local IBM representative.

S IEM2816I ILLEGAL ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

System Action: Remainder of environment attributes ignored.

S IEM2817I COMPILER ERROR. INVALID ATTRIBUTE CODE IN STATEMENT NUMBER xxx

Explanation: An invalid attribute marker has been found in the dictionary entry corresponding to the file attributes in the statement specified

System Action: Ignores the rest of the entry

User Response: Save relevant data. Call your local IBM representative.

S IEM2818I CONFLICTING ATTRIBUTE IN STATEMENT NUMBER xxx IGNORED.

Explanation: An attribute other than 'ENVIRONMENT' clashes with previously declared attributes in the specified statement

System Action: Ignores this attribute.

S IEM2819I ERRONEOUS USE OF PARENTHESIS IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Misplaced parenthesis in ENVIRONMENT attribute

System Action: Remainder of ENVIRONMENT attribute ignored

S IEM2820I ERRONEOUS USE OF COMMA IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Misplaced comma in ENVIRONMENT attribute

System Action: Remainder of ENVIRONMENT attribute ignored

S IEM2821I ILLEGAL CHARACTER IN KEYWORD IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Invalid keyword in ENVIRONMENT attribute

System Action: Remainder of ENVIRONMENT attribute ignored

S IEM2822I FIELD TOO LARGE IN ENVIRONMENT OPTION IN STATEMENT NUMBER xxx

Explanation: Field in item in ENVIRONMENT attribute too large

System Action: Remainder of ENVIRONMENT attribute ignored

S IEM2823I ERROR IN FORMAT OF ENVIRONMENT ATTRIBUTE IN STATEMENT NUMBER xxx

Explanation: Format of item in ENVIRONMENT attribute incorrect

System Action: Remainder of ENVIRONMENT attribute ignored

S IEM2824I CONFLICT BETWEEN ENVIRONMENT ATTRIBUTE AND OTHER ATTRIBUTES IN STATEMENT NUMBER xxx

Explanation: An option in the ENVIRONMENT attribute clashes with either another ENVIRONMENT option or with a declared attribute.

System Action: Remainder of ENVIRONMENT attribute ignored

S IEM2825I CONFLICTING OPTIONS IN ENVIRONMENT ATTRIBUTE IN STATEMENT NUMBER xxx. REST OF ENVIRONMENT IGNORED.

System Action: DECLARE control block is constructed from attributes which have already been processed. The rest are ignored.

User Response: Correct ENVIRONMENT option

S IEM2826I IMPLEMENTATION RESTRICTION. DIRECT FILE zzzz DECLARED IN STATEMENT NUMBER xxx MUST HAVE AN ORGANIZATION SUBFIELD IN THE ENVIRONMENT ATTRIBUTE.

System Action: No compile-time action, but execution will fail.

User Response: Provide ENVIRONMENT attribute

W IEM2827I A D COMPILER OPTION HAS BEEN DECLARED IN THE ENVIRONMENT LIST IN STATEMENT NUMBER xxx. IT HAS BEEN IGNORED.

W IEM2828I ENVIRONMENT OPTIONS CLTASA AND CLT360 HAVE BOTH BEEN DECLARED IN STATEMENT NUMBER xxx. THE SECOND ONE LISTED WILL BE IGNORED.

W IEM2829I IN STATEMENT NUMBER xxx THE PARAMETER SPECIFIED IN THE INDEXAREA OPTION IS GREATER THAN 32767 AND HAS BEEN IGNORED.

Explanation: If the parameter is not specified or is outside the permitted range, data management uses as much main storage as is required for the master index.

S IEM2833I COMPILER ERROR. OPERAND OF CL OR SL NOT LABEL.

Explanation: The dictionary entry referenced after a compiler label or statement label marker in the text is not in fact a label

System Action: The label definition is ignored.

User Response: Save relevant data. Call your local IBM representative.

T IEM2834I COMPILER ERROR. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM2835I COMPILER ERROR. SUBSCRIPTED LABEL CHAIN ERROR.

Explanation: Subscripted labels in the source program result in the creation of chains of dictionary entries. An error in the chaining causes this message to appear.

System Action: The label definition is ignored

User Response: Save relevant data. Call your local IBM representative.

T IEM2836I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE.

Explanation: Not enough scratch core is available for the generated label number table created by this phase. The condition arises when a large number of such labels have been used, and this in turn is related to the size of the program.

System Action: The compilation is terminated

User Response: Break down the program into smaller modules

T IEM2837I COMPILER ERROR. MULTIPLY DEFINED LABEL OR INVALID LABEL NUMBER.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM2852I COMPILER ERROR. NON-ZERO OFFSET IN PSEUDO-REGISTER REFERENCE.

Explanation: Use of a pseudo-register accompanied by literal offset has been called for by compiled code. This cannot be assembled owing to the manner in which pseudo-register relocation is performed.

System Action: The literal offset is ignored.

User Response: Save relevant data. Call your local IBM representative.

S IEM2853I COMPILER ERROR. REFERENCE TO INVALID DICTIONARY ENTRY.

Explanation: A dictionary reference in the input text does not correspond to a legal dictionary entry.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data. Call your local IBM representative.

S IEM2854I COMPILER ERROR. INVALID DICTIONARY REFERENCE OFFSET.

Explanation: A dictionary reference in the input text corresponds to a valid dictionary entry, but the dictionary reference offset is not valid.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data. Call your local IBM representative.

S IEM2855I COMPILER ERROR. REQUESTED OFFSET NOT ASSIGNED.

Explanation: Although implied by a dictionary reference in the text, storage has not been allocated

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data. Call your local IBM representative.

S IEM2865I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM CONTAINS TOO MANY BLOCKS AND/OR CONTROLLED VARIABLES.

Explanation: The compiler allocates a pseudo-register entry for each block and CONTROLLED variable in the source program. The maximum number of such entries is 1,024 .

System Action: No pseudo-registers are allocated for items after the limit has been reached.

User Response: Reduce number of blocks, or CONTROLLED variables, in program to less than 1,025.

W IEM2866I THIS PL/I COMPILATION HAS GENERATED EXTERNAL NAMES IN WHICH THE FIRST LEADING CHARACTER OF THE EXTERNAL PROCEDURE NAME HAS BEEN REPLACED BY A SPECIAL CHARACTER.

Explanation: The external procedure name with its first character changed is being used as a base for generating names for External Symbol Dictionary entries. If the same thing happens in another compilation, and the two are then joined by the Linkage Editor, two External Symbol Dictionary entries may have the same name.

System Action: None

E IEM2867I IMPLEMENTATION RESTRICTION. EXTERNAL NAME zzzz HAS BEEN TRUNCATED TO 7 CHARACTERS.

Explanation: External identifiers are restricted to 7 characters

System Action: Name of ESD entry truncated by taking first 4 and last 3 characters; phase then carries on normally.

User Response: Shorten the name.

W IEM2868I THIS PL/I COMPILATION HAS GENERATED EXTERNAL NAMES IN WHICH THE SECOND LEADING CHARACTER OF THE EXTERNAL PROCEDURE NAME HAS BEEN REPLACED BY A SPECIAL CHARACTER.

Explanation: The external procedure with its second character changed is being used as a base for generating names for External Symbol Dictionary entries. If the same thing happens in another compilation, and the two are then joined by the Linkage Editor, two External Symbol Dictionary entries may have the same name.

System Action: None

T IEM2881I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM2882I COMPILER ERROR IN STATEMENT NUMBER xxx. OPERAND OF DC CODE INVALID.

Explanation: The operand of a DCA4 pseudo-code item is not valid - the operand should always be relocatable.

System Action: An offset of zero is assembled into the text.

User Response: Save relevant data. Call your local IBM representative.

S IEM2883I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID REQUEST FOR RELOCATABLE TEXT.

Explanation: The operand of a branch instruction has been found to require relocation.

System Action: An offset of zero is assembled into the text.

User Response: Save relevant data. Call your local IBM representative.

S IEM2884I COMPILER ERROR IN STATEMENT NUMBER xxx. NON-ZERO OFFSET IN PSEUDO-REGISTER REFERENCE.

Explanation: Use of a pseudo-register accompanied by a literal offset has been called for by compiled code. This cannot be assembled owing to the manner in which pseudo-register relocation is performed.

System Action: The literal offset is ignored

User Response: Save relevant data. Call your local IBM representative.

S IEM2885I COMPILER ERROR IN STATEMENT NUMBER xxx. REFERENCE TO INVALID DICTIONARY ENTRY.

Explanation: A dictionary reference in the input text does not correspond to a legal dictionary entry

System Action: An offset of zero is assembled into the output text

User Response: Save relevant data. Call your local IBM representative.

S IEM2886I COMPILER ERROR IN STATEMENT NUMBER xxx. INVALID DICTIONARY REFERENCE OFFSET.

Explanation: A dictionary reference in the text corresponds to a valid dictionary entry, but the dictionary reference offset is not valid.

System Action: An offset of zero is assembled into the output text

User Response: Save relevant data. Call your local IBM representative.

S IEM2887I COMPILER ERROR IN STATEMENT NUMBER xxx. REQUESTED OFFSET NOT ASSIGNED.

Explanation: Although implied by a dictionary reference in the input text, storage has not been allocated.

System Action: An offset of zero is assembled into the output text.

User Response: Save relevant data. Call your local IBM representative.

T IEM2888I COMPILER ERROR IN STATEMENT NUMBER xxx. UNDEFINED LABEL.

Explanation: No offset has been assigned to a label generated by the compiler.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM2897I IMPLEMENTATION RESTRICTION. QUALIFIED NAME zzzz LONGER THAN 256 CHARACTERS.

Explanation: The fully qualified name of the variable indicated will not fit into its Symbol Table entry

System Action: Leaves Symbol Table entry incomplete and carries on with the initialization of the Static Internal control section.

User Response: Shorten the qualified name

S IEM2898I DATA-DIRECTED GET STATEMENT IN PROGRAM WITH NO LIST BUT PROGRAM HAS NO DATA VARIABLES.

System Action: Zeros are inserted in the argument list for the call to the Library routine to 'GET DATA', and compilation continues

User Response: Correct GET statement

W IEM2899I INITIALIZATION SPECIFIED FOR TOO FEW ELEMENTS IN STATIC ARRAY zzzz

System Action: Initialization terminated when end of initial string is found.

W IEM2900I INITIALIZATION SPECIFIED FOR TOO MANY ELEMENTS IN STATIC ARRAY zzzz

System Action: Initialization is terminated when every element has been initialized.

T IEM2913I COMPILER ERROR. INVALID PSEUDO-CODE OPERATION.

Explanation: The input text contains a marker which is not valid

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM3088I -3213I THE CONFLICTING ATTRIBUTE aaaa HAS BEEN IGNORED IN THE DECLAR-

ATION OF IDENTIFIER yyyy IN STATEMENT NUMBER xxx

Explanation: The attribute given in the message conflicts with another attribute declared for the same identifier, or is invalid for that identifier.

System Action: The attribute given in the message is ignored.

E IEM3584I AN UNBALANCED NUMBER OF PARENTHESES HAS BEEN DETECTED WITHIN A STATEMENT AT OR NEAR STATEMENT NUMBER xxx

Explanation: An occurrence of a comma immediately followed by a period at or near the given statement has been taken as a statement delimiter. The statement contains an unbalanced number of parentheses.

T IEM3841I I/O ERROR ON SEARCHING DIRECTORY.

Explanation: This message is written directly on SYSPRINT. A permanent I/O error was detected when an attempt was made to search the directory of the library containing the compiler.

System Action: Compilation is terminated

User Response: Check the directory and re-attempt compilation. If error persists, have the computing system checked.

T IEM3842I COMPILER ERROR. ALL TEXT BLOCKS IN CORE ARE BUSY. REFERENCED BLOCK CANNOT BE BROUGHT INTO CORE.

Explanation: All blocks in core have become busy. Compiler cannot continue since an external block cannot be read in.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3843I COMPILER ERROR. ZDABRF USED WITH BLOCK NOT IN CORE.

Explanation: Referenced block is not in core

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3844I COMPILER ERROR. DICTIONARY ENTRY IS TOO LONG FOR THIS ENVIRONMENT.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3845I COMPILER ERROR. REFERENCED TEXT BLOCK NOT IN CORE.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3846I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE. ALL TEXT BLOCKS FULL.

Explanation: There is no more space for text in this environment

System Action: Compilation is terminated

User Response: Subdivide program and recompile

T IEM3847I COMPILER ERROR. REQUEST FOR SCRATCH CORE IS GREATER THAN 4K.

Explanation: Request for scratch core exceeds 4096 bytes

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3848I COMPILER ERROR. AN ATTEMPT WAS MADE TO RELEASE UNALLOCATED CORE.

Explanation: Attempt to release unallocated scratch core

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3849I COMPILER ERROR. PHASE yy IN RELEASE LIST IS NOT IN PHASE DIRECTORY.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3850I COMPILER ERROR. PHASE yy IN LOAD LIST IS NOT IN PHASE DIRECTORY.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3851I COMPILER ERROR. PHASE yy NOT MARKED. IT IS LOADED.

Explanation: An unmarked phase is loaded

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3852I COMPILER ERROR. REFERENCED BLOCK IS NOT IN USE. COMPILER CANNOT CONTINUE.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3853I IMPLEMENTATION RESTRICTION. SOURCE PROGRAM TOO LARGE. DICTIONARY IS FULL.

Explanation: This message is written directly on SYSPRINT.

System Action: Compilation is terminated

User Response: Subdivide into more than one program and recompile

T IEM3855I ERROR IN PHASE yy.

Explanation: This message is written directly on SYSPRINT. A compiler error has been discovered during the printing of compile-time diagnostic messages (if phase quoted in message is BM) or of source-program diagnostic messages (if phase is XA).

System Action: Compilation is terminated. Note that only the diagnostic message output is incomplete. All other output files have been generated satisfactorily.

User Response: Save relevant data. Call your local IBM representative.

T IEM3856I COMPILER ERROR. PROGRAM CHECK TYPE nnnn HAS OCCURRED IN PHASE yy AT OR NEAR STATEMENT NUMBER xxx

Explanation: A program check has occurred during compilation. This is due to a compiler failure which may have been exposed by an error in the source code.

System Action: Compilation is terminated

User Response: Check source code carefully. If an error is found, correcting it may enable compilation to be completed successfully. Whether or not an error is found, please save relevant data and contact your your local IBM representative.

T IEM3857I COMPILER ERROR. ATTEMPT TO PASS CONTROL TO AN UNNAMED PHASE. AN UNMARKED PHASE HAS BEEN ENCOUNTERED.

Explanation: An unmarked phase has been encountered. Compiler cannot continue

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3858I COMPILER ERROR. REQUESTED OR UNWANTED PHASE NOT IN PHASE DIRECTORY.

Explanation: Request to mark a phase which is not in phase

directory. Compiler cannot continue.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3859I INSUFFICIENT CORE IS AVAILABLE TO CONTINUE THIS COMPILATION.

Explanation: An attempt is being made to expand the number of text blocks in core. The GETMAIN routine has failed to get the core. This will only occur where less than 45,056 bytes are available to the compiler, or when the SIZE option has been given too large a value.

System Action: Compilation is terminated

User Response: Check the SIZE option and check that the required core is available in the system on which the compilation is being run

E IEM3860I I/O ERROR ON SYSIN. RECORD ACCEPTED AS INPUT.

Explanation: The error may be a machine error, or, if SYSIN is a card reader, there may be a hole pattern which does not represent a valid System/360 character (validity check).

System Action: The error message number is printed in the source listing before the record in error. The record is accepted as input.

User Response: If SYSIN is a card reader, check that every column of the indicated card contains a valid code. If the I/O error persists, have the computing system checked.

T IEM3861I I/O ERROR ON SYSLIN. GENERATION OF LOAD FILE IS TERMINATED.

User Response: Check DD card and recompile. If I/O error persists, have computing system checked.

T IEM3862I I/O ERROR ON SYSPRINT

Explanation: This is an on-line message. It is written to operator. There is an I/O error on SYSPRINT. The compiler cannot continue.

System Action: Compilation is terminated

User Response: Check DD card and recompile. If I/O error persists, have computing system checked.

T IEM3863I I/O ERROR ON SYSPUNCH. GENERATION OF OBJECT DECK IS TERMINATED.

User Response: Check DD card and recompile. If I/O error persists, have computing system checked.

T IEM3864I I/O ERROR ON SYSUT1

Explanation: This message is written directly on SYSPRINT. There is an I/O error on SYSUT1. The compiler cannot continue.

System Action: Compilation is terminated

User Response: Check DD card and recompile. If I/O error persists, have computing system checked.

T IEM3865I ERROR IN COMPILER ABORT

Explanation: This message is written directly on SYSPRINT. The compiler has tried twice to abort and cannot do so. Compilation will therefore terminate without the production of any further diagnostic messages.

System Action: Compilation is terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM3872I I/O ERROR ON SYSUT3

Explanation: This message is written directly on SYSPRINT. There is an I/O error on SYSUT3. The compiler cannot continue.

System Action: Compilation is terminated

User Response: Check DD card and recompile. If I/O error

persists, have computing system checked.

E IEM3873I I/O ERROR ON SYSUT3. RECORD ACCEPTED AS INPUT.

System Action: Compilation continues

User Response: Check DD card and recompile. If I/O error persists, have computing system checked.

T IEM3874I UNABLE TO OPEN SYSIN

Explanation: This message is written directly on SYSPRINT. Unable to open SYSIN. The compiler cannot continue.

System Action: Compilation is terminated

User Response: Check SYSIN DD card and recompile

T IEM3875I UNABLE TO OPEN SYSLIN. LOAD FILE NOT GENERATED.

User Response: Check SYSLIN DD card and recompile. If error persists, have computing system checked.

T IEM3876I UNABLE TO OPEN SYSPRINT

Explanation: This is an online message. It is written to operator. Unable to open SYSPRINT. The compiler cannot continue

System Action: Compilation is terminated

User Response: Check SYSPRINT DD card and recompile

T IEM3877I UNABLE TO OPEN SYSPUNCH. OBJECT DECK NOT GENERATED.

User Response: Check SYSPUNCH DD card and recompile

T IEM3878I UNABLE TO OPEN SYSUT1. COMPILATION CANNOT CONTINUE.

System Action: Compilation is terminated

User Response: Check SYSUT1 DD card and recompile

T IEM3880I UNABLE TO OPEN SYSUT3

Explanation: This message is written directly on SYSPRINT. Unable to open SYSUT3. The compiler cannot continue.

System Action: Compilation is terminated

User Response: Check SYSUT3 DD card and recompile

S IEM3888I SYSPUNCH BLOCKSIZE NOT A MULTIPLE OF 80. THE DECK AND MACDCK OPTIONS HAVE BEEN DELETED.

Explanation: On opening SYSPUNCH, the blocksize definition, either on the DD card or in the data-set label, was not a multiple of 80.

System Action: The DECK and MACDCK options are deleted

User Response: Correct the blocksize definition and recompile

S IEM3889I SYSLIN BLOCKSIZE NOT A MULTIPLE OF 80. THE LOAD OPTION HAS BEEN DELETED.

Explanation: On opening SYSLIN, the blocksize definition, either on the DD card or in the data-set label, was not a multiple of 80.

System Action: The LOAD option is deleted

User Response: Correct the blocksize definition and recompile

W IEM3890I NO RECFM GIVEN FOR SYSIN. U TYPE RECORDS ARE ASSUMED

Explanation: No RECFM definition has been found in the DCB parameter of the SYSIN DD card or the data-set label.

System Action: Compilation proceeds assuming U type records. U format will be specified in the data-set label when SYSIN is closed.

User Response: Check RECFM definition on SYSIN DD card and rerun if necessary.

T IEM3891I SYSIN BLOCKSIZE IS TOO LARGE

Explanation: On opening SYSIN for unblocked records, a blocksize of greater than 100 has been specified either in the

DCB parameter of the SYSIN DD card or in the data-set label.

System Action: Compilation is terminated

User Response: Change the invalid blocksize definition and recompile

T IEM3893I SYSIN BLOCKSIZE NOT A MULTIPLE OF RECORD LENGTH.

Explanation: On opening SYSIN for FB format records, it has been noted that the blocksize definition, either on the DD card or in the data set label, is not an exact multiple of the record length.

System Action: Compilation is terminated.

User Response: Correct the blocksize definition and recompile

T IEM3894I SYSIN BLOCKSIZE NOT EQUAL TO RECORD LENGTH.

Explanation: On opening SYSIN for F format records, the block size and record-length definitions, either on the DD card or in the data-set label, were found to be unequal.

System Action: Compilation is terminated.

User Response: Change the incorrect definition and recompile

T IEM3895I SYSIN RECORD LENGTH TOO LARGE

Explanation: On opening SYSIN for F format records, a record length definition, either on the DD card or in the data-set label, was found to be greater than 100.

System Action: Compilation is terminated.

User Response: Correct the record-length definition and recompile

T IEM3896I SYSPRINT BLOCKSIZE IS NOT OF FORM 4+N*125

Explanation: On opening SYSPRINT, the blocksize definition, either on the DD card or in the data-set label, was not of the form 4+N*125.

System Action: Compilation is terminated

User Response: Correct the blocksize definition and recompile

T IEM3897I SYSIN DEFINITION IS INVALID

Explanation: On opening SYSIN, the record-format definition, either on the DD card or in the data-set label, was varying. This is invalid.

System Action: Compilation is terminated

User Response: Correct the definition of SYSIN and recompile

W IEM3898I COMPILER CORE REQUIREMENT EXCEEDED SIZE GIVEN. AUXILIARY STORAGE USED.

System Action: SPILL file opened

W IEM3899I A BLOCK FOR OVERFLOW DICTIONARY ENTRY OFFSETS WAS CREATED DURING COMPILER PHASE YY

Explanation: This message occurs only in compilations run with the extended dictionary option. An entry offset table in a dictionary block became full before the entries filled the block.

System Action: The block is created to hold the entry offsets overflowing from any entry offset tables during this compilation.

E IEM3900I ERROR IN PROCESS STATEMENT.

Explanation: This message is written directly on SYSPRINT. The syntax of the PROCESS statement is incorrect.

System Action: An attempt is made to interpret the statement correctly. Actual results will depend on the nature of the syntax error.

User Response: Check that the options required have been correctly applied. If not, and recompilation is necessary,

correct the syntax of the PROCESS statement.

E IEM3901I ERROR IN PROCESS STATEMENT. DEFAULT OPTIONS ASSUMED.

Explanation: This message is written directly on SYSPRINT. Invalid syntax in the PROCESS statement has rendered the options unrecognizable.

System Action: The installation defaults are assumed for all options.

User Response: If the use of installation default options is unsatisfactory, correct the syntax of the PROCESS statement and recompile.

E IEM3902I OBJNM FIELD TOO LARGE. FIRST EIGHT CHARACTERS OF NAME HAVE BEEN USED.

Explanation: The name specified in the OBJNM option may not have more than eight characters.

System Action: First eight characters of name used.

User Response: Either amend object module name as required, or alter other references to object module to correspond with truncated name.

W IEM3903I CARRIAGE CONTROL POSITION LIES WITHIN THE SOURCE MARGIN. IT HAS BEEN IGNORED.

User Response: Recompile with carriage control position outside source margin.

E IEM3904I THE FOLLOWING STRING NOT IDENTIFIED AS A KEYWORD - yyyy

Explanation: This message is written directly on SYSPRINT. The compiler was processing the option list passed to it as an invocation parameter, when it found a character string that it could not identify as a keyword.

System Action: The offending character string is ignored.

User Response: Correct the erroneous parameter, and recompile.

E IEM3905I THE FOLLOWING KEYWORD DELETED, DEFAULT USED FOR - yyyy

Explanation: This message is written directly on SYSPRINT. The compiler was processing the option list passed to it as an invocation parameter, when it found an option keyword that had been deleted at system generation.

System Action: The keyword passed at invocation time is ignored. The default interpretation for the option, as set at system generation, is used.

User Response: None, unless it is required to reinstate the deleted keyword, in which case it is necessary to generate the required version of the compiler with a system generation run.

E IEM3906I OPTION SPECIFICATION CONTAINS INVALID SYNTAX, DEFAULT USED FOR - yyyy

Explanation: This message is written directly on SYSPRINT. The compiler was processing the option list passed to it as an invocation parameter, when it found that a sub-parameter, associated with the keyword given in the diagnostic message, was incorrectly specified.

System Action: The keyword passed at invocation time is ignored. The default interpretation for the option, as set at system generation, is used.

User Response: Correct the erroneous parameter, and recompile.

E IEM3907I THE FOLLOWING NAME IGNORED AS IT DOES NOT APPEAR IN THE PHASE DIRECTORY - yy

Explanation: This message is written directly on SYSPRINT. The two characters given in the message were used as parameters to the DUMP option. This usage is incorrect since the characters do not represent the name of a compiler phase.

System Action: The processing of the DUMP option continues, unless the two characters were used to indicate the first

phase of an inclusive phase dump, in which case the scan of the DUMP option is terminated.

User Response: Correct the erroneous parameter, and recompile.

S IEM3908I SYNTAX ERROR IN DUMP OPTION SPECIFICATION

Explanation: This message is written directly on SYSPRINT. Incorrect use of delimiters in the specification of the DUMP option parameters.

System Action: Processing of DUMP option is terminated

User Response: Correct the erroneous specification, and recompile.

T IEM3909I EXTENDED DICTIONARY CAPACITY EXCEEDED. COMPILATION TERMINATED.

Explanation: This message occurs only in compilations run with the extended dictionary option. The block created to hold overflow dictionary entry offsets is full.

System Action: Compilation is terminated

User Response: Subdivide program and recompile

T IEM3910I SYSPRINT BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION

Explanation: The size specified allows a limited buffer area which is smaller than that required by the specified blocksize.

System Action: Compilation is terminated

User Response: Use smaller blocksize or larger SIZE option

W IEM3911I SIZE AVAILABLE FOUND TO BE yyyyyy BYTES. SIZE=44K ASSUMED. COMPILATION CONTINUES.

Explanation: SIZE is found to be less than 44K.

T IEM3912I SYSIN BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION

Explanation: The size specified allows a limited buffer area which is smaller than that required by the buffers for SYSIN, or for SYSIN and SYSPRINT together.

System Action: Compilation is terminated

User Response: Ensure that SIZE option allows room for both the SYSIN and the SYSPRINT buffers.

S IEM3913I SYSPUNCH BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION. THE DECK AND MACDCK OPTIONS HAVE BEEN DELETED.

Explanation: The SIZE specified allows a limited buffer area which is smaller than that required by the specified SYSPUNCH blocksize.

System Action: The DECK and MACDCK options are deleted

User Response: Ensure that the SIZE option allows room for the SYSPUNCH buffers needed, and recompile.

S IEM3914I SYSLIN BLOCKSIZE IS TOO LARGE WITH THIS SIZE OPTION. THE LOAD OPTION HAS BEEN DELETED.

Explanation: The SIZE specified allows a limited buffer area which is smaller than that required by the specified SYSLIN blocksize.

System Action: The LOAD option is deleted.

User Response: Ensure that the SIZE option allows room for the SYSLIN buffers needed and recompile.

COMPILE-TIME PROCESSING DIAGNOSTIC MESSAGES

The details given under the heading "Source Program Diagnostic Messages" apply equally to compile-time processing messages, with one exception: all compile-time processing messages are listed in a group following the SOURCE2 input listing and preceding the source program listing.

The line number in the messages refers to the line in which the error was found. The incorrect statement may have commenced on an earlier line.

S IEM4106I UNEXPECTED END-OF-FILE IN STRING AT OR BEYOND LINE NUMBER xxx. A STRING DELIMITER HAS BEEN INSERTED.

Explanation: End-of-file encountered while scanning for closing quote of a string constant.

System Action: Closing quote inserted before end-of-file.

T IEM4109I REPLACEMENT VALUE IN LINE NUMBER xxx CONTAINS UNDELIMITED STRING. PROCESSING TERMINATED.

Explanation: End-of-string delimiter cannot be found in a replacement value.

E IEM4112I ILLEGAL CHARACTER IN APPARENT BIT STRING IN LINE NUMBER xxx. STRING TREATED AS A CHARACTER STRING.

S IEM4115I UNEXPECTED END-OF-FILE IN COMMENT AT OR BEYOND LINE NUMBER xxx. A COMMENT DELIMITER HAS BEEN INSERTED.

Explanation: End-of-file encountered while scanning for end-of-comment delimiter.

T IEM4118I REPLACEMENT VALUE IN LINE NUMBER xxx CONTAINS UNDELIMITED COMMENT. PROCESSING TERMINATED.

Explanation: End-of-comment delimiter cannot be found in a replacement value.

E IEM4121I INVALID CHARACTER HAS BEEN REPLACED BY BLANK IN OR FOLLOWING LINE NUMBER xxx

Explanation: Invalid character found in source text

T IEM4124I COMPILER ERROR. PUSH DOWN STACK OUT OF PHASE

System Action: Processing terminated

User Response: Save relevant data. Call your local IBM representative.

T IEM4130I UNDECLARED IDENTIFIER zzzz REFERENCED AT LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: An attempt is made to execute a statement which references an identifier for which a DECLARE statement has not been executed.

E IEM4133I % ENCOUNTERED IN LABELLIST OF STATEMENT IN LINE NUMBER xxx. IT HAS BEEN IGNORED.

User Response: Remove % from label list

E IEM4134I UNEXPECTED COLON WITHOUT PRECEDING LABEL IN LINE NUMBER xxx. COLON HAS BEEN IGNORED.

S IEM4136I STATEMENT TYPE NOT RECOGNIZABLE IN LINE NUMBER xxx. STATEMENT DELETED.

E IEM4139I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH USE AS LABEL IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless an attempt is made to execute a statement which references the ill-defined identifier.

E IEM4142I LABEL zzzz IN LINE NUMBER xxx MULTIPLY DEFINED. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement which references the multiply defined label is executed.

W IEM4143I LABELS BEFORE DECLARE STATEMENT IN IINE NUMBER xxx ARE IGNORED.

E IEM4148I IDENTIFIER zzzz IN LINE NUMBER xxx USED WITH CONFLICTING ATTRIBUTES. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: Usage of identifier conflicts with a previous usage or declaration. If the line number refers to a procedure END statement, the error occurred within the procedure.

System Action: No action unless a statement is executed which references the identifier in error.

E IEM4150I FORMAL PARAMETER zzzz WAS NOT DECLARED IN PROCEDURE ENDING IN LINE NUMBER xxx. TYPE CHARACTER HAS BEEN FORCED.

E IEM4151I LABEL zzzz IS NOT DEFINED. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the undefined label.

E IEM4152I END OF FILE OCCURS BEFORE END FOR CURRENT PROCEDURE OR DO. END HAS BEEN INSERTED AT LINE NUMBER xxx.

E IEM4153I LABEL zzzz IS UNDEFINED IN THE PROCEDURE ENDING IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: Label may have been defined outside of procedure, but transfers out of procedures are not allowed.

System Action: Any reference to the label in the procedure will terminate processing.

E IEM4154I SEMICOLON TERMINATES IF EXPRESSION IN LINE NUMBER xxx. SEMICOLON HAS BEEN IGNORED.

S IEM4157I NEITHER % NOR THEN FOLLOWS IF EXPRESSION IN LINE NUMBER xxx. IF STATEMENT DELETED.

E IEM4160I % MISSING BEFORE THEN OF IF STATEMENT IN LINE NUMBER xxx. % HAS BEEN INSERTED.

E IEM4163I THEN MISSING FOLLOWING % IN IF STATEMENT IN LINE NUMBER xxx. A THEN HAS BEEN INSERTED.

E IEM4166I COMPILE TIME STATEMENT MUST FOLLOW THEN OR ELSE IN LINE NUMBER xxx. A % HAS BEEN INSERTED IN FRONT OF STATEMENT.

Explanation: % does not precede the first statement in the THEN or ELSE clause of an IF statement.

User Response: If the statement in question is meant to be a non-compile time statement, it should be put inside of a "% DO" group.

E IEM4169I THEN MISSING FROM IF STATEMENT AT LINE NUMBER xxx IN A COMPILE TIME PROCEDURE. A THEN HAS BEEN INSERTED.

E IEM4172I THE % IN LINE NUMBER xxx IS NOT ALLOWED IN COMPILE TIME PROCEDURES. IT HAS BEEN IGNORED.

W IEM4175I LABELS BEFORE ELSE IN LINE NUMBER xxx HAVE BEEN IGNORED.

Explanation: Label(s) found preceding an ELSE statement.

S IEM4176I NO STATEMENT FOLLOWS THEN OR ELSE IN LINE NUMBER xxx. A NULL STATEMENT HAS BEEN INSERTED.

E IEM4178I ELSE WITHOUT PRECEDING IF IN LINE NUMBER xxx HAS BEEN IGNORED.

S IEM4184I ASSIGNMENT STATEMENT IN LINE NUMBER xxx MUST END WITH SEMICOLON. TEXT DELETED TILL SEMICOLON IS FOUND.

E IEM4187I LABEL MISSING FROM PROCEDURE STATEMENT IN LINE NUMBER xxx. A DUMMY LABEL HAS BEEN INSERTED.

T IEM4188I IMPLEMENTATION RESTRICTION. NO MORE THAN 254 COMPILE-TIME PROCEDURES MAY BE DEFINED IN A COMPILATION. PROCESSING TERMINATED.

User Response: Delete excess procedures

E IEM4190I LABEL zzzz ON PROCEDURE IN LINE NUMBEER xxx IS PREVIOUSLY DEFINED. ANY REFERENCE TO IT WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the multiply defined label.

E IEM4193I ILLEGAL USE OF FUNCTION NAME zzzz ON LEFT HAND SIDE OF EQUALS SYMBOL. ANY REFERENCE WILL TERMINATE PROCESSING.

E IEM4196I PREVIOUS USE OF IDENTIFIER zzzz CONFLICTS WITH USE AS ENTRY NAME IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the erroneous identifier.

S IEM4199I FORMAL PARAMETER zzzz IS REPEATED IN PARAMETER LIST IN LINE NUMBER xxx. THE SECOND OCCURRENCE HAS BEEN REPLACED BY A DUMMY PARAMETER.

S IEM4202I IMPLEMENTATION RESTRICTION: MORE THAN 15 PARAMETERS OCCUR

IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: Processing is terminated if an attempt is made to execute a statement which references the procedure that has more than 15 parameters.

E IEM4205I FORMAL PARAMETER MISSING IN LINE NUMBER xxx. A DUMMY HAS BEEN INSERTED.

E IEM4208I UNRECOGNIZABLE PARAMETER yyyy IN LINE NUMBER xxx. IT HAS BEEN REPLACED BY A DUMMY PARAMETER.

S IEM4211I PARAMETER IN LINE NUMBER xxx NOT FOLLOWED BY COMMA OR PARENTHESIS. TEXT DELETED TO NEXT COMMA OR END OF STATEMENT.

S IEM4212I UNEXPECTED END OF PROCEDURE STATEMENT IN LINE NUMBER xxx. RIGHT PARENTHESIS INSERTED.

Explanation: A semicolon was encountered during scan of an apparent parameter list.

System Action: A right parenthesis is inserted before the semicolon and processing continues.

E IEM4214I ILLEGAL FORM FOR ATTRIBUTE FOR RETURNED VALUE IN LINE NUMBER xxx. TEXT DELETED TO SEMICOLON.

Explanation: Returned values may only be FIXED or CHARACTER.

System Action: CHARACTER attribute is assigned

E IEM4217I NO ATTRIBUTE FOR RETURNED VALUE IN LINE NUMBER xxx. CHARACTER ATTRIBUTE IS USED.

System Action: CHARACTER attribute is assigned

S IEM4220I SEMICOLON NOT FOUND WHERE EXPECTED IN PROCEDURE STATEMENT IN LINE NUMBER xxx. TEXT DELETED UP TO NEXT SEMICOLON.

E IEM4223I ENTRY ATTRIBUTE AND PROCEDURE STATEMENT FOR ENTRY zzzz DISAGREE ON THE NUMBER OF PARAMETERS. THE LATTER IS USED.

System Action: The number of parameters specified in the PROCEDURE statement is used.

E IEM4226I RETURNS ATTRIBUTE AND PROCEDURE STATEMENT FOR ENTRY zzzz DISAGREE ON ATTRIBUTE OF RETURNED VALUE.

System Action: The returned value will first be converted to the type on the procedure statement and will then be converted to the type given in the RETURNS attribute. A third conversion can occur if the type given in the returns attribute does not agree with the type required where the result is used.

S IEM4229I PROCEDURE STATEMENT AT LINE NUMBER xxx MAY NOT BE USED WITHIN A PROCEDURE. PROCEDURE HAS BEEN DELETED.

Explanation: Compile-time procedures may not be nested.

System Action: Text is deleted up to and including the first % END following the erroneous PROCEDURE statement.

S IEM4232I PROCEDURE STATEMENT AT LINE NUMBER xxx MAY NOT FOLLOW THEN OR ELSE. PROCEDURE HAS BEEN REPLACED BY A NULL STATEMENT.

Explanation: A PROCEDURE statement may appear in a THEN or ELSE clause only if it is inside a compile-time DO group.

S IEM4235I RETURN STATEMENT IN LINE NUMBER xxx IS NOT ALLOWED OUTSIDE OF COMPILE-TIME PROCEDURE. STATEMENT DELETED.

E IEM4238I RETURNED VALUE MUST BE PARENTHESIZED IN LINE NUMBER xxx. PARENTHESIS INSERTED.

E IEM4241I RETURNS EXPRESSION IN LINE NUMBER xxx DOES NOT END RETURN STATEMENT. REMAINDER OF STATEMENT HAS BEEN IGNORED.

S IEM4244I GOTO IN LINE NUMBER xxx IS NOT FOLLOWED BY LABEL. STATEMENT DELETED.

E IEM4247I PREVIOUS USE OF IDENTIFIER zzzz CONFLICTS WITH USE AS OBJECT OF GOTO IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

System Action: No action unless a statement is executed which references the erroneous identifier.

S IEM4248I SEMICOLON NOT FOUND WHERE EXPECTED IN GOTO STATEMENT IN LINE NUMBER xxx. TEXT DELETED UP TO NEXT SEMICOLON.

T IEM4250I GOTO zzzz IN LINE NUMBER xxx TRANSFERS CONTROL INTO ITERATIVE DO OR ENCLOSED INCLUDED TEXT. PROCESSING TERMINATED.

S IEM4253I ACTIVATE OR DEACTIVATE IN LINE NUMBER xxx NOT ALLOWED IN A COMPILE-TIME PROCEDURE. STATEMENT DELETED.

E IEM4254I EMPTY ACTIVATE OR DEACTIVATE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

E IEM4256I SURPLUS COMMA IN ACTIVATE OR DEACTIVATE IN LINE NUMBER xxx. THE COMMA HAS BEEN DELETED.

S IEM4259I UNRECOGNIZABLE FIELD IN ACTIVATE OR DEACTIVATE STATEMENT IN LINE NUMBER xxx. THE FIELD HAS BEEN DELETED.

S IEM4262I ONLY PROCEDURES OR VARIABLES MAY HAVE ACTIVITY CHANGED. IDENTIFIER zzzz IN LINE NUMBER xxx HAS BEEN DELETED FROM STATEMENT.

S IEM4265I COMMA MUST SEPARATE FIELDS OF ACTIVATE AND DEACTIVATE STATEMENTS. IN LINE NUMBER xxx TEXT AFTER IDENTIFIER yyyy HAS BEEN DELETED UP TO NEXT COMMA.

S IEM4271I INVALID SYNTAX IN DO STATEMENT IN LINE NUMBER xxx. IT HAS BEEN CONVERTED TO A GROUPING DO.

W IEM4277I NO MAXIMUM VALUE WAS SPECIFIED IN ITERATIVE DO IN LINE NUMBER xxx. PROGRAM WILL LOOP UNLESS ALTERNATE EXIT IS PROVIDED.

E IEM4280I UNEXPECTED % IN LINE NUMBER xxx TREATED AS HAVING BEEN PRECEDED BY SEMICOLON.

E IEM4283I MULTIPLE TO'S HAVE OCCURRED IN DO STATEMENT IN LINE NUMBER xxx. SECOND 'TO' HAS BEEN CHANGED TO 'BY'.

E IEM4286I MULTIPLE BY'S HAVE OCCURRED IN DO STATEMENT IN LINE NUMBER xxx. SECOND 'BY' HAS BEEN CHANGED TO 'TO'.

E IEM4289I DO STATEMENT IN LINE NUMBER xxx SHOULD END WITH SEMICOLON. TEXT TO SEMICOLON DELETED.

E IEM4292I END STATEMENT AT LINE NUMBER xxx MAY NOT FOLLOW THEN OR ELSE. A NULL STATEMENT HAS BEEN INSERTED BEFORE THE END STATEMENT.

E IEM4295I SEMICOLON NOT FOUND WHERE EXPECTED IN END STATEMENT IN LINE NUMBER xxx. TEXT DELETED UP TO SEMICOLON.

E IEM4296I END STATEMENT IN LINE NUMBER xxx NOT PRECEDED BY DO OR PROCEDURE STATEMENT. END HAS BEEN DELETED.

Explanation: An END statement has been encountered which is not preceded by a DO or PROCEDURE statement that has not already been terminated.

E IEM4298I LABEL REFERENCED ON END STATEMENT IN LINE NUMBER xxx CANNOT BE FOUND. END TREATED AS HAVING NO OPERAND.

Explanation: The label cannot be found on a DO or PROCEDURE statement that has not already been terminated.

E IEM4299I END STATEMENT ENDING PROCEDURE IN LINE NUMBER xxx DID NOT HAVE A PRECEDING PERCENT. A PERCENT IS INSERTED.

Explanation: The END statement referred to in this message is the logical end of the procedure.

E IEM4301I IDENTIFIER zzzz ON END STATEMENT IN LINE NUMBER xxx IS NOT A LABEL. END TREATED AS HAVING NO OPERAND.

E IEM4304I PROCEDURE zzzz DID NOT INCLUDE A RETURN STATEMENT.

Explanation: Language syntax requires use of RETURN statement in a procedure.

System Action: A null value is returned if the procedure is invoked.

S IEM4307I INCLUDE STATEMENT AT LINE NUMBER xxx IS NOT ALLOWED IN COMPILE-TIME PROCEDURES. STATEMENT DELETED.

E IEM4310I IMPLEMENTATION RESTRICTION. DDNAME IN LINE NUMBER xxx HAS BEEN TRUNCATED TO 8 CHARACTERS.

Explanation: The first of a pair of data set identifiers in an INCLUDE statement is a ddname and as such is limited to a maximum of 8 characters.

S IEM4313I UNRECOGNIZABLE FIELD IN INCLUDE STATEMENT AT LINE NUMBER xxx. FIELD HAS BEEN DELETED.

System Action: Text is deleted up to next comma or semicolon.

S IEM4319I EMPTY INCLUDE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

Explanation: At least one identifier must appear in an INCLUDE statement i.e., the data set member name.

E IEM4322I IMPLEMENTATION RESTRICTION. MEMBER NAME IN LINE NUMBER xxx HAS BEEN TRUNCATED TO 8 CHARACTERS.

System Action: First 8 characters of member name have been used.

User Response: Correct data set member name in INCLUDE statement.

E IEM4325I RIGHT PARENTHESIS INSERTED AFTER MEMBER NAME IN LINE NUMBER xxx.

T IEM4328I COMPILER ERROR. DICTIONARY INFORMATION INCORRECT.

Explanation: A name containing an invalid character is found in the dictionary.

System Action: Processing is terminated

User Response: Save relevant data. Call your local IBM representative.

S IEM4331I DECLARE STATEMENT IN LINE NUMBER xxx IS ILLEGAL AFTER THEN OR ELSE. STATEMENT DELETED.

User Response: Correct program. A DECLARE statement can appear in the THEN or ELSE clause of an IF statement if it is inside a DO group.

E IEM4332I EMPTY DECLARE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

S IEM4334I IMPLEMENTATION RESTRICTION. FACTORING IN DECLARE STATEMENT IN LINE NUMBER xxx EXCEEDS 3 LEVELS. REMAINDER OF STATEMENT DELETED.

User Response: Reduce level of factoring in DECLARE statement.

E IEM4337I SURPLUS COMMA HAS BEEN FOUND IN DECLARE STATEMENT IN LINE NUMBER xxx. THIS COMMA HAS BEEN DELETED.

E IEM4340I IDENTIFIER MISSING WHERE EXPECTED IN LINE NUMBER xxx. A DUMMY IDENTIFIER HAS BEEN INSERTED.

E IEM4343I IDENTIFIER zzzz IN LINE NUMBER xxx HAS MULTIPLE DECLARATIONS. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: An identifier may be declared only once.

System Action: No action unless a statement is executed which references the multiply declared identifier.

S IEM4346I UNRECOGNIZABLE SYNTAX IN DECLARE STATEMENT IN LINE NUMBER xxx. STATEMENT DELETED.

E IEM4349I LABEL zzzz CANNOT BE DECLARED IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

Explanation: An attempt has been made to declare an identifier which has already been used as a label.

System Action: No action unless a statement is executed which references the declared label.

E IEM4352I EXTRA PARENTHESIS DELETED IN LINE NUMBER xxx.

E IEM4355I ILLEGAL ATTRIBUTE yyyy IN LINE NUMBER xxx. ATTRIBUTE HAS BEEN DELETED.

Explanation: Legal attributes are FIXED, CHARACTER, ENTRY and RETURNS.

System Action: The illegal attribute is deleted.

E IEM4358I CLOSING RIGHT PARENTHESIS INSERTED IN LINE NUMBER xxx.

E IEM4361I RETURNS ATTRIBUTE OCCURRED WITHOUT ENTRY ATTRIBUTE FOR PROCEDURE zzzz IN DECLARE STATEMENT AT OR BEFORE LINE NUMBER xxx.

Explanation: Both ENTRY and RETURNS attributes must be declared for a compile-time procedure name.

System Action: The identifier is treated as an ENTRY name. If it is referenced, the arguments will be converted to the types declared for the procedure parameters.

E IEM4364I NO ATTRIBUTES WERE DECLARED FOR IDENTIFIER zzzz IN DECLARE STATEMENT AT OR BEFORE LINE NUMBER xxx. CHARACTER HAS BEEN ASSIGNED.

E IEM4367I RETURNS ATTRIBUTE NOT GIVEN FOR ENTRY NAME zzzz IN DECLARE STATEMENT AT OR BEFORE LINE NUMBER xxx.

Explanation: Both ENTRY and RETURNS attributes must be declared for a compile-time procedure name.

System Action: The attribute of the returned value is determined by the relevant PROCEDURE statement.

E IEM4370I ENTRY ATTRIBUTE DISAGREES WITH DECLARATION FOR FORMAL PARAMETER zzzz. THE LATTER HAS BEEN USED.

Explanation: An ENTRY attribute in a DECLARE statement does not agree with the parameter attributes declared in the procedure.

System Action: If the relevant procedure is referenced, the argument will be converted to the type declared for the formal parameter.

E IEM4373I RETURNS ATTRIBUTE IN LINE NUMBER xxx MUST BE PARENTHESIZED. PARENTHESIS INSERTED.

E IEM4376I ONLY FIXED OR CHARACTER ARE ALLOWED IN RETURNS ATTRIBUTE IN LINE NUMBER xxx. ATTRIBUTE IGNORED.

Explanation: An illegal attribute was found.

System Action: The attribute of the returned value is determined by the relevant PROCEDURE statement.

E IEM4379I ATTRIBUTE yyyy IS ILLEGAL IN ENTRY ATTRIBUTE IN LINE NUMBER xxx. NO CONVERSION WILL BE DONE.

Explanation: An invalid attribute was found.

System Action: No conversion to an ENTRY attribute will be carried out. However, if the relevant procedure is referenced, arguments will be converted to the types declared for the procedure parameters.

E IEM4382I ATTRIBUTE CONFLICTS WITH PREVIOUS ATTRIBUTE FOR IDENTIFIER zzzz IN LINE NUMBER xxx. ATTRIBUTE IGNORED.

E IEM4383I PREVIOUS USAGE OF IDENFIFIER zzzz CONFLICTS WITH ATTRIBUTE IN LINE NUMBER xxx. ANY REFERENCE WILL TERMINATE PROCESSING.

E IEM4391I OPERAND MISSING IN LINE NUMBER xxx. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

S IEM4394I ILLEGAL OPERATOR yyyy IN LINE NUMBER xxx. IT HAS BEEN REPLACED BY A PLUS.

W IEM4397I A LETTER IMMEDIATELY FOLLOWS CONSTANT yyyy IN LINE NUMBER xxx. AN INTERVENING BLANK HAS BEEN ASSUMED.

E IEM4400I OPERATOR .NOT. IN LINE NUMBER xxx USED AS AN INFIX OPERATOR. IT HAS BEEN REPLACED BY .NE.

T IEM4403I COMPILER ERROR. EXPRESSION SCAN OUT OF PHASE.

System Action: Processing is terminated.

User Response: Save relevant data. Call your local IBM representative.

E IEM4406I PREVIOUS USAGE OF IDENTIFIER zzzz CONFLICTS WITH USE IN EXPRESSION IN LINE NUMBER xxx.

System Action: Processing is terminated if an attempt is made to execute a statement which references the identifier in question.

S IEM4407I UNDECIPHERABLE KEYWORD. nnn IDENTIFIERS HAVE BEEN DELETED BEFORE yyyy IN LINE NUMBER xxx.

Explanation: The processor has found a mis-match while scanning a keyword consisting of more than one identifier.

System Action: The identifiers preceding the non-matching identifier are deleted.

S IEM4409I OPERATOR MISSING IN LINE NUMBER xxx. A PLUS HAS BEEN INSERTED.

S IEM4412I NO EXPRESSION WHERE ONE IS EXPECTED IN LINE NUMBER xxx. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

S IEM4415I ILLEGAL OPERAND yyyy IN LINE NUMBER xxx HAS BEEN REPLACED BY A FIXED DECIMAL ZERO.

E IEM4421I MISSING LEFT PARENTHESIS INSERTED AT BEGINNING OF EXPRESSION IN LINE NUMBER xxx.

T IEM4433I REFERENCE IN LINE NUMBER xxx TO STATEMENT OR IDENTIFIER WHICH IS IN ERROR. PROCESSING TERMINATED.

S IEM4436I EXCESS ARGUMENTS TO FUNCTION zzzz IN LINE NUMBER xxx. EXTRA ARGUMENTS HAVE BEEN DELETED.

Explanation: Too many arguments appear in a procedure reference.

W IEM4439I TOO FEW ARGUMENTS TO FUNCTION zzzz IN LINE NUMBER xxx. MISSING ARGUMENTS HAVE BEEN REPLACED BY FIXED DECIMAL ZEROS.

Explanation: Too few arguments appear in a procedure reference.

E IEM4448I NO ENTRY DECLARATION FOR PROCEDURE zzzz REFERENCED IN LINE NUMBER xxx. ATTRIBUTES TAKEN FROM PROCEDURE.

Explanation: All procedure names must be declared with ENTRY and RETURNS attributes before the procedure is referenced.

T IEM4451I PROCEDURE zzzz REFERENCED IN LINE NUMBER xxx CANNOT BE FOUND. PROCESSING TERMINATED.

T IEM4452I RECURSIVE USE OF PROCEDURE zzzz IN LINE NUMBER xxx IS DISALLOWED. PROCESSING TERMINATED.

E IEM4454I TOO FEW ARGUMENTS HAVE BEEN SPECIFIED FOR THE BUILTIN FUNCTION SUBSTR IN LINE NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

E IEM4457I TOO MANY ARGUMENTS HAVE BEEN SPECIFIED FOR THE BUILTIN FUNCTION SUBSTR IN LINE NUMBER xxx. EXTRA ARGUMENTS HAVE BEEN IGNORED.

E IEM4460I FIXED OVERFLOW HAS OCCURRED IN LINE NUMBER xxx. RESULT TRUNCATED.

System Action: Truncation occurs on left to 5 decimal digits.

E IEM4463I ZERO DIVIDE HAS OCCURRED AT LINE NUMBER xxx. RESULT SET TO ONE.

S IEM4469I END-OF-FILE FOUND IMBEDDED IN STATEMENT IN LINE NUMBER xxx. EXECUTION OF STATEMENT WILL CAUSE TERMINATION.

E IEM4472I IDENTIFIER BEGINNING zzzz IN STATEMENT AT LINE NUMBER xxx IS TOO LONG AND HAS BEEN TRUNCATED.

Explanation: Identifiers may not exceed 31 characters in length.

System Action: The identifier is truncated to the first 31 characters.

S IEM4473I CONSTANT yyyy IN LINE NUMBER xxx HAS PRECISION GREATER THAN 5. A FIXED DECIMAL ZERO HAS BEEN INSERTED.

Explanation: Implementation restriction. Precision of fixed decimal numbers is limited to 5 digits.

System Action: A value of zero is assigned.

E IEM4475I QUESTION MARK IN LINE NUMBER xxx HAS NO SIGNIFICANCE. IT HAS BEEN IGNORED

Explanation: Question mark, although a recognizable character in PL/I, has no syntactical meaning.

T IEM4478I STRING IN LINE NUMBER xxx CONVERTS TO A FIXED DECIMAL NUMBER WITH PRECISION GREATER THAN 5. PROCESSING TERMINATED.

Explanation: Implementation restriction. Precision of fixed decimal numbers is limited to 5 digits.

System Action: Processing is terminated

T IEM4481I CHARACTER STRING IN LINE NUMBER xxx CONTAINS CHARACTER OTHER THAN 1 OR 0 AND CANNOT BE CONVERTED TO A BIT STRING. PROCESSING TERMINATED.

T IEM4484I STRING IN LINE NUMBER xxx OR IN PROCEDURE REFERENCED IN SAID LINE NUMBER CANNOT BE CONVERTED TO A FIXED DECIMAL CONSTANT. PROCESSING TERMINATED.

T IEM4499I A % STATEMENT IS FOUND IN A REPLACEMENT VALUE IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: A replacement value may not contain a compile-time statement.

T IEM4502I AN IDENTIFIER zzzz WITH CONFLICTING USAGE OR MULTIPLE DEFINITIONS IS REFERENCED IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: An attempt is made to execute a statement which references an identifier that was not properly defined.

User Response: Correct program

S IEM4504I VARIABLE zzzz IS USED IN LINE NUMBER xxx BEFORE IT IS INITIALIZED. IT HAS BEEN GIVEN NULL STRING OR ZERO VALUE.

Explanation: A value must be assigned to variables before they are referenced after being declared.

T IEM4505I DD STATEMENT FOR INCLUDE zzzz MISSING IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: A DD statement must be present, in the Job Control cards for the compilation, with a name in the name field that corresponds to the ddname identifier in the INCLUDE statement. If no ddname is specified in the INCLUDE statement, a SYSLIB DD statement is required.

User Response: Insert appropriate DD statement and recompile.

T IEM4508I UNRECOVERABLE I/O ERROR WHILE SEARCHING FOR MEMBER OF INCLUDE zzzz IN LINE NUMBER xxx. PROCESSING TERMINATED.

User Response: Check DD statement and reattempt compilation. If error persists, check computing system.

T IEM4511I ILLEGAL RECORD FORMAT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: Included records must be a fixed length of not more than 100 characters with a maximum blocking factor of 5. Blocksize must be a multiple of the record length.

T IEM4514I MEMBER OF INCLUDE zzzz IN LINE NUMBER xxx NOT FOUND ON DATA SET. PROCESSING TERMINATED.

User Response: Check INCLUDE statement, DD statement and data file.

W IEM4517I RECORD LENGTH NOT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. RECORD LENGTH EQUAL TO BLOCKSIZE HAS BEEN ASSUMED.

User Response: Correct record length specification in DD statement, if necessary.

W IEM4520I BLOCKSIZE NOT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. BLOCKSIZE EQUAL TO RECORD LENGTH HAS BEEN ASSUMED.

User Response: Correct blocksize specification in DD statement, if necessary.

W IEM4523I RECORD LENGTH AND BLOCKSIZE NOT SPECIFIED FOR INCLUDE zzzz IN LINE NUMBER xxx. RECORD LENGTH OF 80 AND BLOCKSIZE OF 400 HAVE BEEN ASSUMED.

User Response: Correct record length and block size specifications in DD statement, if necessary.

T IEM4526I I/O ERROR WHILE READING TEXT INCLUDED FROM zzzz AT LINE NUM-

BER xxx. PROCESSING TERMINATED.

User Response: Check DD statement and reattempt compilation. If error persists, check computing system.

T IEM4529I IMPLEMENTATION RESTRICTION. EXCESSIVE LEVEL OF NESTING OR REPLACEMENT AT LINE NUMBER xxx. PROCESSING TERMINATED.

Explanation: Level of nesting in this case is calculated by summing the number of current unbalanced left parentheses, the number of current nested DO's, the number of current nested IF's, and the number of current nested replacements. A level of 50 is always acceptable.

T IEM4532I INPUT RECORD AT LINE NUMBER xxx IS TOO LONG. PROCESSING TERMINATED.

Explanation: Input record contains more than 100 characters.

T IEM4535I INPUT RECORD CONTAINS FEWER CHARACTERS THAN SORMGIN REQUIRES. PROCESSING TERMINATED.

Explanation: The length of the input record is less than the left margin of the SORMGIN specification.

User Response: Check SORMGIN option on EXEC control card.

T IEM4538I COMPILER ERROR. SUBSTR MUST BE A SINGLE WORD KEYWORD.

Explanation: The keyword SUBSTR has been replaced, in the compiler tables, with a multiple keyword. The Compile-Time Processor cannot handle this situation.

System Action: Processing is terminated

User Response: Replace the multiple keyword with a single keyword and recompile the compiler.

T IEM4547I COMPILER ERROR. INSUFFICIENT SPACE FOR TABLES.

System Action: Processing is terminated

User Response: Save relevant data. Call your local IBM representative.

E IEM4550I RIGHT PARENTHESIS INSERTED IN LINE NUMBER xxx TO END ARGUMENT LIST FOR PROCEDURE zzzz.

Explanation: The argument list referred to is in a source program reference to a compile-time procedure.

T IEM4553I IN LINE NUMBER xxx ARGUMENT LIST FOR PROCEDURE zzzz CONTAINS COMPILE TIME CODE. PROCESSING TERMINATED.

Explanation: Compile-time code may not be embedded in argument list of compile-time procedure reference.

E IEM4559I LEFT PARENTHESIS BEGINNING ARGUMENT LIST OF PROCEDURE zzzz WAS NOT FOUND. PROCEDURE WAS INVOKED AT LINE NUMBER xxx WITHOUT ARGUMENTS.

Explanation: The argument list referred to is in a source program reference to a compile-time procedure.

E IEM4562I IDENTIFIER IN LINE NUMBER xxx EXCEEDS 31 CHARACTERS. REPLACEMENT WAS DONE ON TRUNCATED FORM zzzz.

Explanation: A non-compile-time source text identifier consists of more than 31 characters.

E IEM4570I THE THIRD ARGUMENT OF BUILT-IN FUNCTION SUBSTR IS NEGATIVE, IN LINE NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

E IEM4572I THE THIRD ARGUMENT OF BUILT-IN FUNCTION SUBSTR EXCEEDS THE STRING LENGTH, IN LINE NUMBER xxx. THE SUBSTRING HAS BEEN TRUNCATED AT THE END OF THE ORIGINAL STRING.

E IEM4574I THE COMBINED SECOND AND THIRD ARGUMENTS OF BUILT-IN FUNCTION SUBSTR EXCEED THE STRING LENGTH, IN LINE NUMBER xxx. THE SUBSTRING HAS BEEN TRUNCATED AT THE END OF THE ORIGINAL STRING.

E IEM4576I THE SECOND ARGUMENT OF BUILT-IN FUNCTION SUBSTR IS LESS THAN ONE, IN LINE NUMBER xxx. ITS VALUE HAS BEEN RESET TO ONE.

E IEM4578I THE SECOND ARGUMENT OF BUILT-IN FUNCTION SUBSTR EXCEEDS THE STRING LENGTH, IN LINE NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

S IEM4580I AN UNINITIALISED VARIABLE HAS BEEN FOUND IN A BUILT-IN FUNCTION ARGUMENT LIST, IN STATEMENT NUMBER xxx. A NULL STRING HAS BEEN RETURNED.

User Response: Initialise the variable before invoking the built-in function.

OBJECT-TIME DIAGNOSTIC MESSAGES

The messages in the following text may be printed on the output data set specified for SYSPRINT, as the result of an exceptional or error condition occurring during the execution of a PL/I program. If the SYSPRINT DD statement is absent, then the object-time messages appear on the operator's console, except for the ON CHECK system action messages and the COPY option output, which will not be produced at all in this case.

Each message number is of the form IHEnnnI, where the code IHE indicates a PL/I library message, and nnn the number of the message. The final character I indicates the informative nature of the message.

Diagnostic messages are printed at execution time for two main reasons:

1. An error occurs for which no specific ON-condition exists in PL/I. A diagnostic message is printed, and the ERROR ON-condition is raised.
2. An ON-condition is raised, by compiled code or by the Library, and the action required is system action, for which the language specifies COMMENT as part of the necessary action.

Object time diagnostic messages will take one of the following forms:

1. IHEnnnI FILE name - text AT location message
2. IHEnnnI rtname - text AT location message
3. IHEnnnI text AT location message

where 'name' is the name of the file associated with the error (given only in I/O diagnostic messages)

'rtname' is the name of the Library routine in which the error occurred (given only for computational subroutines).

'location message' is either

OFFSET ± hhhhh FROM ENTRY POINT E1

or

OFFSET ± hhhhh FROM ENTRY POINT OF cccc ON-UNIT

Note: If it is a Model 91 message resulting from an imprecise interrupt, "AT OFFSET..." is replaced by "NEAR OFFSET..." since the instruction causing the interrupt cannot be precisely identified.

If the statement number compiler option has been specified, each message will also contain IN STATEMENT nnnnn prior to AT location message. nnnnn gives the number of the statement in which the condition occurred.

The diagnostic messages for other than ON-type errors are mainly self-explanatory. Explanations in the following lists are given only when the message is not self-explanatory.

IHE003I SOURCE PROGRAM ERROR IN STATEMENT nnnnn

This message will always contain a statement number whether or not the compiler option is specified.

IHE004I INTERRUPT IN ERROR HANDLER - PROGRAM TERMINATED

Explanation: When an unexpected program interrupt occurs during the handling of another program interrupt, it indicates that the program has a disastrous error in it, such as DSA chain out of order, instructions overwritten, or such. The program is abnormally terminated, and the above message is printed out at the console. A dump is produced with a User Completion Code of 4000.

IHE005I PSEUDO-REGISTER VECTOR TOO LONG - PROGRAM NOT EXECUTED

Explanation: This error arises when the sum of the number of procedures, the number of files, and the number of controlled variables exceeds 1000.

It causes return to the Supervisor from IHESAP; PL/I program is not entered. The message always appears at the console. A return code of 4004 is generated.

IHE006I NO MAIN PROCEDURE. PROGRAM TERMINATED.

Explanation: No external procedure in the program has been given the option MAIN. This message appears at the console. A return code of 4008 is generated.

IHE009I IHEDUM*. NO PLIDUMP DD CARD. EXECUTION TERMINATED.

Explanation: Execution has been abnormally terminated with a dump and a completion code of (3000 + Return Code (if set))

IHE010I PROGRAM ENDED BY OS360. RETURN CODE = hhh (a hexadecimal number).

Explanation: The major task has been terminated abnormally by the operating system. The above message appears on the console.

IHE011I KEY ERROR WHEN CLOSING FILE AT END OF TASK

Explanation: An unresolved key error exists for which no condition can now be raised. The above message appears on the console.

I/O Errors

IHE018I FILE name - FILE TYPE NOT SUPPORTED

IHE020I FILE name - ATTEMPT TO READ OUTPUT FILE

IHE021I FILE name - ATTEMPT TO WRITE INPUT FILE

IHE022I GET/PUT STRING EXCEEDS STRING SIZE

Explanation:

For input: programmer has requested more than exists on the input string.

For output: programmer is trying to write more than his output string will hold.

IHE023I FILE name - OUTPUT TRANSMIT ERROR NOT ACCEPTABLE

Explanation: The ERROR is raised, (i) upon return from a TRANSMIT ON-unit, if the device in error is other than a printer, or (ii) if access to a file by RECORD I/O has been attempted after the TRANSMIT condition has been raised for output.

IHE024I FILE name - PRINT OPTION/FORMAT ITEM FOR NON-PRINT FILE

Explanation: Attempt to use PAGE, LINE or SKIP ≤ 0 for a non-print file.

IHE025I DISPLAY - MESSAGE OR REPLY AREA LENGTH ZERO

Explanation: This message appears only if the REPLY option is exercised.

IHE026I FILE name - DATA DIRECTED INPUT - INVALID ARRAY DATUM

Explanation: Number of subscripts on external medium does not correspond to number of declared subscripts.

IHE027I GET STRING - UNRECOGNIZABLE DATA NAME

Explanation:

1. GET DATA - name of data item found in string is not known at the time of the GET statement, or
2. GET DATA data list - name of data item found in string is not specified in the list.

IHE029I FILE name - UNSUPPORTED FILE OPERATION

Explanation: Programmer has executed an I/O statement with an option or verb not applicable to the specified file.

For example:

I/O Option or Verb	File Attribute
READ SET| LOCATE	DIRECT| (SEQUENTIAL UNBUFFERED)
REWRITE (without FROM)| EXCLUSIVE| UNLOCK| (READ NOLOCK)	(SEQUENTIAL INPUT|OUTPUT| UPDATE)| (DIRECT INPUT| OUTPUT)
KEYTO	REGIONAL DIRECT
LINESIZE| PAGESIZE	STREAM (INPUT| UPDATE)

IHE030I FILE name - REWRITE/DELETE NOT IMMEDIATELY PRECEDED BY READ

IHE031I FILE name - INEXPLICABLE I/O ERROR

Explanation: Data Management has detected some error in the current I/O operation, but has provided no further details.

IHE032I FILE name - OUTSTANDING READ FOR UPDATE EXISTS

Explanation: When a record is read from an INDEXED file which contains blocked records and which is open for DIRECT UPDATE, the record must be rewritten. Between the READ statement and the associated REWRITE statement, no other operation may be performed on the file.

IHE033I FILE name - NO COMPLETED READ EXISTS (INCORRECT NCP VALUE)

Explanation: This message may be issued because the correct NCP value has not been specified or it may be due to incorrect source code.

IHE034I FILE name - TOO MANY INCOMPLETE I/O OPERATIONS

Explanation: The number of incomplete I/O operations equals the NCP value.

IHE035I FILE name - EVENT VARIABLE ALREADY IN USE

IHE036I FILE name - IMPLICIT OPEN FAILURE, CANNOT PROCEED

Explanation: There has been a failure in an implicit OPEN operation.

IHE037I FILE name - ATTEMPT TO REWRITE OUT OF SEQUENCE

Explanation: An intervening I/O statement occurs between a READ statement and a REWRITE statement referring to the same record.

IHE038I FILE name - ENDFILE FOUND UNEXPECTEDLY IN MIDDLE OF DATA ITEM.

Explanation: The ERROR condition is raised when end-of-file is encountered before the delimiter when scanning list-directed or data-directed input, or if the field width in the format list of edit-directed input would take the scan beyond the end-of-file.

IHE039I FILE name - ATTEMPT TO CLOSE FILE NOT OPENED IN CURRENT TASK

I/O ON-conditions

All these conditions may be raised by the SIGNAL statement.

IHE100I FILE name - UNRECOGNIZABLE DATA NAME

Explanation:

Initiating ON-condition: NAME

1. GET DATA - name of data item found on external medium is not known at the time of the GET statement, or
2. GET DATA data list - name of data item found on external medium is not specified in the list.

IHE110I FILE name - RECORD CONDITION SIGNALED

IHE111I FILE name - RECORD VARIABLE SMALLER THAN RECORD SIZE

Explanation: The variable specified in the READ statement INTO option allows fewer characters than exist in the record.

F format records:
a WRITE statement attempts to

put a record smaller than the record size.

All formats:
a REWRITE attempts to replace a record with one of smaller size. (Note: This condition cannot be detected for U-format records read for UNBUFFERED or DIRECT files.)

IHE112I FILE name - RECORD VARIABLE LARGER THAN RECORD SIZE

Explanation: The variable specified in the READ statement INTO option requires more characters than exist in the record; or a WRITE statement attempts to put out a record greater than the available record size; or a REWRITE statement attempts to replace a record with one of greater size.

IHE113I ATTEMPT TO WRITE/LOCATE ZERO LENGTH RECORD

Explanation: A WRITE or REWRITE statement attempts to put out a record of zero length, or a LOCATE statement attempts to get buffer space for a record of zero length; such records are used for end-of-file markers for direct access storage devices.

IHE114I FILE name - ZERO LENGTH RECORD READ

Explanation: A record of zero length has been read from a REGIONAL data set accessed in the DIRECT mode. This should not occur, unless the data set was created by another processor. A zero length record, on a direct access device, is an end-of-file signal.

IHE120I FILE name - PERMANENT INPUT ERROR

Explanation:

Initiating ON-CONDITION:
TRANSMIT

IHE121I FILE name - PERMANENT OUTPUT ERROR

Explanation:

Initiating ON-condition:
TRANSMIT

IHE122I FILE name - TRANSMIT CONDITION SIGNALED

IHE130I FILE name - KEY CONDITION SIGNALED

IHE131I FILE name - KEYED RECORD NOT FOUND

Explanation: READ, REWRITE, or DELETE statement specified record key which does not match with records of data set. If REGIONAL (2) or (3) data sets are employed, and the DD statement parameter LIMCT is used, then the record does not exist within the number of records or tracks searched, but may exist elsewhere.

IHE132I FILE name - ATTEMPT TO ADD DUPLICATE KEY

Explanation: WRITE statement specified a key value which already exists within data set.

1. INDEXED data sets: detected for both SEQUENTIAL and DIRECT access.

2. REGIONAL data sets: detected only for REGIONAL (1) and (2) SEQUENTIAL output.

IHE133I FILE name - KEY SEQUENCE ERROR

Explanation: WRITE statement specified, during creation of data set (OUTPUT SEQUENTIAL), a key which for:

1. INDEXED data sets is lower in binary collating sequence than prior key

2. REGIONAL data sets the relative record/track value is lower than that of prior key.

IHE134I FILE name - KEY CONVERSION ERROR

Explanation: WRITE, READ, REWRITE, or DELETE statement for REGIONAL data set specified character string key value whose relative record/track partition contains characters other than blank or the digits 0 through 9, or which contains only the character blank.

IHE135I FILE name - KEY SPECIFICATION ERROR

Explanation:

1. INDEXED: the KEYFROM or KEY expression may be the NULL string. Alternatively, RKP does not equal zero and the embedded key is not identical with that specified by the KEYFROM option (or the KEY option in the case of a rewrite statement). A third possibility is that an attempt has been made during SEQUENTIAL UPDATE to replace a record by one whose embedded key does not match that of the original record.

2. REGIONAL: as for INDEXED, or initial character of KEY or KEYFROM expression value is the value (8)'1'B.

IHE136I FILE name - KEYED RELATIVE RECORD/TRACK OUTSIDE DATA SET LIMIT

Explanation: WRITE, READ, REWRITE, or DELETE statement for REGIONAL data set specified a key whose relative record/track value exceeds the number of records or tracks assigned to the data set.

IHE137I FILE name - NO SPACE AVAILABLE TO ADD KEYED RECORD

Explanation: WRITE statement attempted to add record, but data set was full. If REGIONAL (2) or (3) data set, condition is raised if space within optional limits (DD parameter LIMCT) is unavailable.

IHE140I FILE name - END OF FILE ENCOUNTERED

Explanation:

Initiating ON-condition:
ENDFILE

IHE150I FILE name - CANNOT BE OPENED, NO DD CARD

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

IHE151I FILE name - CONFLICTING DECLARE AND OPEN ATTRIBUTES

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

There is a conflict between the declared PL/I file attributes. For example:

Attribute	Conflicting Attributes
PRINT	INPUT, UPDATE, RECORD, DIRECT, SEQUENTIAL, BACKWARDS, BUFFERED, UNBUFFERED, EXCLUSIVE, KEYED
STREAM	UPDATE, RECORD, DIRECT, SEQUENTIAL, BACKWARDS, BUFFERED, UNBUFFERED, EXCLUSIVE, KEYED
EXCLUSIVE	INPUT, OUTPUT, SEQUENTIAL, BACKWARDS, BUFFERED, UNBUFFERED
DIRECT	SEQUENTIAL, BACKWARDS, BUFFERED, UNBUFFERED
UPDATE	INPUT, OUTPUT, BACKWARDS
OUTPUT	INPUT, BACKWARDS
BUFFERED	UNBUFFERED

Some attributes may have been supplied when a file is opened implicitly. Example of attributes implied by I/O statements are:

I/O Statement	Implied Attributes
DELETE	RECORD, DIRECT, UPDATE
GET	INPUT
LOCATE	RECORD, OUTPUT, SEQUENTIAL, BUFFERED
PUT	OUTPUT
READ	RECORD, INPUT
REWRITE	RECORD, UPDATE
UNLOCK	RECORD, DIRECT, UPDATE, EXCLUSIVE
WRITE	RECORD, OUTPUT

In turn, certain attributes may imply other attributes:

Attribute	Implied Attributes
BACKWARDS	RECORD, SEQUENTIAL, INPUT
BUFFERED	RECORD, SEQUENTIAL
DIRECT	RECORD, KEYED
EXCLUSIVE	RECORD, KEYED, DIRECT, UPDATE
KEYED	RECORD
PRINT	OUTPUT, STREAM
SEQUENTIAL	RECORD
UNBUFFERED	RECORD, SEQUENTIAL
UPDATE	RECORD

Finally, a group of alternate attributes has one of the group as a default. The default is implied if none of the group is specified explicitly or is implied by other attributes or by the opening I/O statement. The groups of alternates are:

Group	Default
STREAM|RECORD	STREAM
INPUT|OUTPUT|UPDATE	INPUT
SEQUENTIAL|DIRECT (RECORD files)	SEQUENTIAL
BUFFERED|UNBUFFERED (SEQUENTIAL files)	BUFFERED

IHE152I FILE name - FILE TYPE NOT SUPPORTED

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

The user has attempted to associate a paper-tape device with a file that does not have the INPUT attribute.

IHE153I FILE name - BLOCKSIZE NOT SPECIFIED

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

Block size not specified on DD card, nor on environment. However, will never occur for PRINT file, because default block size is assumed.

IHE154I FILE name - UNDEFINEDFILE CONDITION SIGNALED

IHE155I FILE name - ERROR INITIALIZING REGIONAL DATA SET

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

A REGIONAL data set, opened for DIRECT OUTPUT, cannot be properly formatted during the open process.

IHE156I FILE name - CONFLICTING ATTRIBUTE AND ENVIRONMENT PARAMETERS

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

Examples of conflicting parameters NAME block size is assumed. are:

ENVIRONMENT Parameter	File Attribute
No file organization parameter	KEYED
INDEXED|REGIONAL	STREAM
CONSECUTIVE	DIRECT| EXCLUSIVE
INDEXED	DIRECT OUTPUT
INDEXED|REGIONAL	OUTPUT without KEYED
Blocked records	UNBUFFERED
V-format records	BACKWARDS

IHE157I FILE name - CONFLICTING ENVIRONMENT AND/OR DD PARAMETERS

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

One of the following conflicts exists:

1. F-format records have not been specified for an

INDEXED, REGIONAL(1), or REGIONAL(2) file.

2. Blocked records have been specified with a REGIONAL file.

IHE158I FILE name - KEYLENGTH NOT SPECIFIED

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

A keylength has not been specified for an INDEXED, REGIONAL(2), or REGIONAL(3) file that is being opened for OUTPUT.

IHE159I FILE name - INCORRECT BLOCKSIZE AND/OR LOGICAL RECORD SIZE IN STATEMENT NUMBER xxx

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

One of the following situations exists:

1. F-format records

   a. The specified block size is less than the logical record length.

   b. The specified block size is not a multiple of the logical record length.

2. V-format records

   a. The specified block size is less than the logical record length + 4.

   b. The logical record length is less than 14 for a RECORD file or 15 for a STREAM file.

IHE160I FILE name - LINESIZE GREATER THAN IMPLEMENTATION DEFINED MAXIMUM LENGTH

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

The implementation-defined maximum linesize is:

F-format records 32758

V-format records 32750

IHE161I FILE name - CONFLICTING ATTRIBUTE AND DD PARAMETERS

Explanation:

Initiating ON-condition:
UNDEFINEDFILE

The user has attempted to associate a file with the BACKWARDS attribute with a device that is not a magnetic tape device.

Computational Errors

IHE200I rtname - X LT 0 IN SQRT(X)

IHE202I rtname - X LT 0 IN LOG(X) OR LOG2(X) OR LOG10(X)

IHE203I rtname - ABS(X) GE (2**50)*K IN SIN(X) OR COS(X) (K=PI) OR SIND(X) OR COSD(X) (K=180)

IHE204I rtname - ABS(X) GE (2**50)*K IN TAN(X) (K=PI) OR TAND(X) (K=180)

IHE206I rtname - X=Y=0 IN ATAN(Y,X) AND ATAND(Y,X)

IHE208I rtname - ABS(X) GT 1 IN ATANH(X)

IHE209I rtname - X=0, Y LE 0 IN X**Y

IHE210I rtname - X=0, Y NOT POSITIVE REAL IN X**Y

IHE211I rtname - Z=+I OR -I IN ATAN(Z) OR Z=+1 OR -1 IN ATANH(Z)

IHE212I rtname - ABS(X) GE (2**18)*K IN SIN(X) OR COS(X) (K=PI) OR SIND(X) OR COSD(X) (K=180)

IHE213I rtname - ABS(X) GE (2**18)*K IN TAN(X) (K=PI) OR TAND(X) (K=180)

List of Routine Names

IHESQS	Short float square root
IHELNS	Short float logarithm
IHETNS	Short float tangent
IHEATS	Short float arctan
IHESNS	Short float sine and cosine
IHEHTS	Short float hyperbolic arctan
IHESQL	Long float square root
IHELNL	Long float logarithm
IHETNL	Long float tangent
IHEATL	Long float arctan
IHESNL	Long float sine and cosine
IHEHTL	Long float hyperbolic arctan
IHEXIS	Short float integer exponentiation
IHEXIL	Long float integer exponentiation
IHEXXS	Short float general exponentiation
IHEXXL	Long float general exponentiation
IHEXIW	Short float complex integer exponentiation
IHEXIZ	Long float complex integer exponentiation
IHEXXW	Short float complex general exponentiation
IHEXXZ	Long float complex general exponentiation
IHEATW	Short float complex arctan and hyperbolic arctan
IHEATZ	Long float complex arctan and hyperbolic arctan

Computational ON-Conditions

All these conditions may be raised by the SIGNAL statement.

IHE300I OVERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the exponent of a floating-point number exceeds the permitted maximum, as defined by implementation.

IHE310I SIZE

Explanation: This condition is raised, by Library routines or by compiled code, when assignment is attempted where the number to be assigned will not fit into the target field. This condition can be raised by allowing the fixed overflow interrupt to occur on account of SIZE. If associated with I/O, then "FILE name" will be inserted between the message number and the text.

IHE320I FIXEDOVERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the result of a fixed-point binary or decimal operation exceeds the maximum field width as defined by implementation.

IHE330I ZERODIVIDE

Explanation: This condition is raised, by Library routines or by compiled code, when an attempt is made to divide by zero, or when the quotient exceeds the precision allocated for the result of a division. The condition can be raised by hardware interrupt or by special coding.

IHE340I UNDERFLOW

Explanation: This condition is raised, by Library routines or by compiled code, when the exponent of a floating-point number is smaller than the implementation-defined mimimum. The condition does not occur when equal floating-point numbers are subtracted.

IHE350I STRINGRANGE

Explanation: This condition is raised by library routines when an invalid reference by the SUBSTR built-in function or pseudo-variable has been detected.

IHE360I AREA CONDITION RAISED IN ALLOCATE STATEMENT

Explanation: There is not enough room in the area in which to allocate the based variable.

IHE361I AREA CONDITION RAISED IN ASSIGNMENT STATEMENT

Explanation: There is not enough room in the area to which the based variable is being assigned.

IHE362I AREA SIGNALED

Structure and Array Errors

IHE380I IHESTR - STRUCTURE OR ARRAY LENGTH GE 16**6 BYTES

Explanation: During the mapping of a structure or array, the length of the structure or array has been found to be greater than or equal to 16**6 bytes.

IHE381I IHESTR - VIRTUAL ORIGIN OF ARRAY GE 16**6 OR LE -16**6

Explanation: During the mapping of a structure, the address of the element with zero subscripts in an array, whether it exists or not, has been computed to be outside the range (-16**6 to +16**6).

Control Program Restrictions

IHE400I DELAY STATEMENT EXECUTED - NO TIMER FUNCTION IN SYSTEM

IHE401I TIME STATEMENT EXECUTED - NO TIMER FUNCTION IN SYSTEM

Condition Type ON-Conditions

IHE500I SUBSCRIPTRANGE

Explanation: This condition is raised, by library routines or by compiled code, when a subscript is evaluated and found to lie outside its specified bounds, or by the SIGNAL statement.

IHE501I CONDITION

Explanation: This condition is raised by execution of a SIGNAL (identifier) statement, referencing a programmer-specified EXTERNAL identifier.

Errors Associated with Tasking

The following errors are associated with execution of the CALL, READ, or WRITE statement with TASK option; with the WAIT statement; with the use of TASK or EVENT variables; with the PRIORITY pseudo-variable or built-in function; or with the COMPLETION pseudo-variable.

IHE550I ATTEMPT TO WAIT ON AN INACTIVE AND INCOMPLETE EVENT

IHE551I TASK VARIABLE ALREADY ACTIVE

Explanation: Task variable is already associated with an active task.

IHE552I EVENT ALREADY BEING WAITED ON

Explanation: During the execution of a WAIT statement, in order to complete the required number of events, an event must not be waited on which is already being waited on in another task.

IHE553I WAIT ON MORE THAN 255 INCOMPLETE EVENTS

IHE554I ACTIVE EVENT VARIABLE AS ARGUMENT TO COMPLETION PSEUDO-VARIABLE

IHE555I INVALID TASK VARIABLE AS ARGUMENT TO PRIORITY PSEUDO-VARIABLE

Explanation: The task variable specified was active and not associated with the current task or one of its immediate subtasks.

IHE556I EVENT VARIABLE ACTIVE IN ASSIGNMENT STATEMENT

IHE557I EVENT VARIABLE ALREADY ACTIVE

Explanation: Event variable is already associated with an active task.

IHE558I ATTEMPT TO WAIT ON AN I/O EVENT IN WRONG TASK

Explanation: An I/O event can be waited on only in the same task as the statement which initiated the I/O operation with which the event is associated.

In the following group of messages, hhh is a hexadecimal number.

IHE571I TASK (name) TERMINATED. COMPLETION CODE= hhh.

IHE572I TASK (name) TERMINATED. COMPLETION CODE = hhh. EVENT VARIABLE OVERWRITTEN OR DESTROYED.

IHE573I TASK (name) TERMINATED. COMPLETION CODE = hhh. TASK VARIABLE OVERWRITTEN OR DESTROYED.

IHE574I TASK (NAME) TERMINATED. COMPLETION CODE = hhh. INVALID FREE STATEMENT.

Explanation: The FREE statement freed, or tried to free, storage to which it is not applicable.

IHE575I TASK (name) TERMINATED. COMPLETION CODE = hhh. DISPLAY STATEMENT. REPLY NOT WAITED FOR.

Explanation: The task terminated normally without waiting for a reply from a DISPLAY statement with the REPLY option.

IHE576I TASK (name) TERMINATED. COMPLETION CODE = hhh. TOO MUCH MAIN STORAGE REQUESTED.

IHE577I TASK (name) TERMINATED WHILE STILL ACTIVE -- END OF BLOCK REACHED IN ATTACHING TASK.

Explanation: The attaching task reached:

1. An EXIT statement, or
2. The end of the block in which the subtask was attached

while the subtask was still active.

IHE579I TASK (name) TERMINATED. COMPLETION CODE = hhh. ABNORMAL TERMINATION DURING PUT STATEMENT.

Conversion ON-Conditions

Conversion errors occur most often on input, either owing to an error in the input data, or because of an error in a format list. For example, in edit-directed input, if the field width of one of the items in the data list is incorrectly specified in the format list, the input stream will get out of step with the format list and a conversion error is likely to occur.

IHE600I CONVERSION CONDITION SIGNALED

IHE601I CONVERSION ERROR IN F-FORMAT INPUT

IHE602I CONVERSION ERROR IN E-FORMAT INPUT

IHE603I CONVERSION ERROR IN B-FORMAT INPUT

IHE604I ERROR IN CONVERSION FROM CHARACTER STRING TO ARITHMETIC

IHE605I ERROR IN CONVERSION FROM CHARACTER STRING TO BIT STRING

IHE606I ERROR IN CONVERSION FROM CHARACTER STRING TO PICTURED CHARACTER STRING

IHE607I CONVERSION ERROR IN P-FORMAT INPUT (DECIMAL)

IHE608I CONVERSION ERROR IN P-FORMAT INPUT (CHARACTER)

IHE609I CONVERSION ERROR IN P-FORMAT INPUT (STERLING)

Note: When condition was due to an I/O conversion, then "FILE name" will be inserted between the message number and the text. Also, when the I/O conversion error was due to a TRANSMIT error, the word (TRANSMIT) is inserted between the file name and the text.

Conversion Errors, Non-ON-Type

IHE700I INCORRECT E(W,D,S) SPECIFICATION

IHE701I F FORMAT W SPECIFICATION TOO SMALL

IHE702I A FORMAT W UNSPECIFIED AND LIST ITEM NOT TYPE STRING

IHE703I B FORMAT W UNSPECIFIED AND LIST ITEM NOT TYPE STRING

IHE704I A FORMAT W UNSPECIFIED ON INPUT

IHE705I B FORMAT W UNSPECIFIED ON INPUT

Explanations: Messages 700 to 705 reveal that an EDIT operation was incorrectly specified.

IHE706I UNABLE TO ASSIGN TO PICTURED CHARACTER STRING

Explanation: A source datum which is not a character string cannot be assigned to a pictured character string because of a mismatch with the PIC description of the target.

IHE798I ONSOURCE OR ONCHAR PSEUDOVARIABLE USED OUT OF CONTEXT

Explanation: This message is printed and the ERROR condition raised if an ONSOURCE or ONCHAR pseudo-variable is used outside an ON-unit, or in an ON-unit other than either a CONVERSION ON-unit or an ERROR or FINISH ON-unit following from system action for CONVERSION.

IHE799I RETURN ATTEMPTED FROM CONVERSION ON-UNIT BUT SOURCE FIELD NOT MODIFIED

Explanation: A CONVERSION ON-unit has been entered as a result of an invalid conversion, and an attempt has been made to return, and hence reattempt the conversion, without using one or other of the pseudo-variables ONSOURCE or ONCHAR to change the invalid character.

Non-Computational Program Interrupt Errors

Certain program interrupts may occur in a PL/I program because the source program has an error which is severe but which cannot be detected until execution time. An example is a call to an unknown procedure, which will result in an illegal operation program interrupt. Other program interrupts, such as addressing, specification, protection, and data interrupts, may arise if PL/I control blocks have been destroyed. This can occur if an assignment is made to an array element whose subscript is out of range, since, if SUBSCRIPTRANGE has not been enabled, the compiler does not check array subscripts; a program interrupt may occur at the time of the assignment or at a later stage in the program. Similarly, an attempt to use the value of an array element whose subscript is out of range may cause an interrupt.

Care must be taken when parameters are passed to a procedure. If the data attributes of the arguments of the calling statement do not agree with those of the invoked entry point, or if an argument is not passed at all, a program interrupt may occur.

The use of the value of a variable that has not been initialized, or has had no assignment made to it, or the use of CONTROLLED variables that have not been allocated, may also cause one of these interrupts.

IHE800I INVALID OPERATION

IHE801I PRIVILEGED OPERATION

IHE802I EXECUTE INSTRUCTION EXECUTED

IHE803I PROTECTION VIOLATION

IHE804I ADDRESSING INTERRUPT

IHE805I SPECIFICATION INTERRUPT

IHE806I DATA INTERRUPT

Explanation: This condition can be caused by an attempt to use the value of a FIXED DECIMAL variable when no prior assignment to, or initialization of, the variable has been performed.

Model 91 Object-Time Diagnostic Messages

After a multiple-exception imprecise interrupt on a Model 91, certain exceptions will remain unprocessed if the ERROR condition is raised before all the exceptions have been handled. If the program subsequently is terminated as a direct result of the ERROR condition being raised in these circumstances, one or more of the following messages will be printed out.

IHE810I PROTECTION EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

IHE811I ADDRESSING EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

IHE812I SPECIFICATION EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

IHE813I DATA EXCEPTION UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

IHE814I ZERODIVIDE UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

IHE815I OVERFLOW UNPROCESSED AFTER MULTIPLE-EXCEPTION IMPRECISE INTERRUPT

Storage Management Errors

The following errors are associated with the handling of storage and transfer of control out of blocks. In some cases, these errors are a result of program error, but it is possible that the messages may be printed because the save area chain, allocation chain, or pseudo-register vector have been overwritten.

IHE900I TOO MANY ACTIVE ON-UNITS AND ENTRY PARAMETER PROCEDURES

Explanation: There is an implementation limit to the number of ON-units and/or entry parameter procedures which can be active at any time. An entry parameter procedure is one that passes an entry name as parameter to a procedure it calls. The total permissible number of these ON-units/entry parameter procedures is 127.

IHE902I GOTO STATEMENT REFERENCES A LABEL IN AN INACTIVE BLOCK

Explanation: The label referred to cannot be found in any of the blocks currently active in the current task; blocks are not freed. The statement number and offset indicate the GO TO statement causing the error.

APPENDIX H: LANGUAGE FEATURES RESTRICTED OR NOT SUPPORTED IN THE FOURTH VERSION

This appendix details the PL/I language features which are not implemented by the fourth version of the PL/I (F) Compiler, or which are implemented with restrictions.

IDENT Option

The IDENT option on OPEN/CLOSE statements is not permitted.

EVENT Option in the DISPLAY Statement

If the EVENT option is specified in the DISPLAY statement, it must follow the REPLY option.

Example:

```
DISPLAY ('LEGAL') EVENT (E1) REPLY
(ANSWER); /*ILLEGAL*/

DISPLAY ('LEGAL') REPLY (ANSWER) EVENT
(E1); /*LEGAL*/
```

EVENT Option in the WRITE Statement

The EVENT option should not be used in the WRITE statement if V or U format records are being added to a REGIONAL(3) data set which is being accessed in a DIRECT UPDATE mode.

INITIAL Attribute

An INITIAL attribute given for CHARACTER or BIT data of STATIC storage class may not specify 'complex expressions' as initial values.

Example:

```
DCL C CHAR(10)
    STATIC INITIAL (3+4I);  /*ILLEGAL*/

DCL C CHAR(10)
    STATIC INITIAL ('3+4I');/*LEGAL*/
```

DEFINED Attribute

Overlay defining is not permitted if the base is either subscripted or is declared CONTROLLED. Correspondence defining is not permitted if the base is declared CONTROLLED.

Example:

```
DCL B CONTROLLED, C(10) AUTOMATIC;

DCL
A DEFINED B,       /*ILLEGAL*/
E DEFINED C(I),    /*ILLEGAL*/
F (10) DEFINED C; /*LEGAL*/
```

No correspondence defining may be used with arrays of structures.

Example:

```
DCL 1 A (10), 2 B, 2 C;

DCL 1 D (5) DEFINED A,2 E,2 F; /*ILLEGAL*/
DCL D (5) DEFINED B;           /*LEGAL*/
```

Structures and Arrays of Structures in Certain Special Contexts

No structure or array of structures may be passed as an argument to either a built-in function (except the STRING, ADDR, and ALLOCATION functions), or to a procedure declared with the attribute GENERIC.

Example:

```
DCL 1 A(10), 2 B;

A=SIN(A);                  /*ILLEGAL*/
B=SIN(B);                  /*LEGAL*/
```

No reference may be made to both a structure and an array of structures in the same expression or assignment.

Example:

```
DCL 1 A(10), 2 B, 2 C,
      1 F(10), 2 B, 2 C,
      1 P, 2 Q, 2 R;

A=P;                       /*ILLEGAL*/
A=F;                       /*LEGAL*/
A(I)=P;                    /*LEGAL*/
A=F, BY NAME;              /*LEGAL*/
A=F(I), BY NAME;           /*ILLEGAL*/
```

No references can be made to cross sections of arrays of structures; the whole of an array of structures may be referenced, or a single element of the array may be referenced, but not a cross section.

Example:

```
DCL 1 A(10,10), 2 B, 2 C(10,10);
DCL 1 X(10), 2 Y, 2 Z(10,10);

A(*,J)=X;                    /*ILLEGAL*/
A=A+1;                       /*LEGAL*/
A(I,J)=X(I);                 /*LEGAL*/
```

A cross-section of an EVENT array is not permitted to appear in a WAIT statement.

Example:

```
DCL EVT (10,10,2) EVENT;

WAIT (EVT) 200; /*LEGAL*/

WAIT (EVT(I,J,2))100; /*LEGAL*/

WAIT (EVT(1,*,1))10; /*ILLEGAL*/
```

VARYING Strings

The only form of varying strings permitted in the INTO or FROM options in RECORD I/O are unsubscripted level1 VARYING strings that are not members of arrays or structures.

Interleaved Arrays of Varying Strings Passed as Arguments

A varying string, passed as part of an argument to a procedure, cannot have its length changed within the invoked procedure if all the following circumstances apply:

1. the argument passed is in the form of a subscripted structure
2. in the invoked procedure, the varying string is regarded as a scalar
3. in the calling procedure, the varying string is one element of an interleaved array. (An interleaved array is one in which other elements of a structure are mapped in storage between elements of the array. In the example below, S is an interleaved array.)
4. a dummy argument is not generated.

Example:

```
DCL 1 P(10),
      2 Q(10),
        3 R FIXED(5,0),
        3 S CHAR(5) VAR,
      2 T FIXED(5,0);

CALL PROC1(P.Q(4,4));    /* LENGTH OF STRING
                            MAY NOT BE CHANGED
                            IN PROC1*/
CALL PROC1((P.Q(4,4)));  /* CHANGE PERMIT-
                            TED. DUMMY ARG-
                            UMENT CREATED */
CALL PROC2(P(4));        /* CHANGE PERMIT-
                            TED. AN ARRAY OF
                            STRINGS IS PAS-
                            SED */
```

LABEL Arrays

Qualified names may not be used as label prefixes. Reference to arrays of label variables initialized by means of subscripted labels must not require explicit structure qualification.

For example:

```
DCL Z(3) LABEL,
    1 S,
     2 Y(3) LABEL,
    1 S1,
     2 Y(3) LABEL,
    1 S2,
     2 X(3) LABEL;

 Z(1): ; /*LEGAL*/
S.Y(1): ; /*ILLEGAL*/
 X(1): ; /*LEGAL*/
```

The FLOAT Attribute

Floating-point precision specified as FLOAT (*) in a parameter description within the GENERIC attribute, is not permitted.

List Processing, Table Handling, and Locate-Mode Input/Output Facilities

The CELL attribute is not permitted.

The SECONDARY Attribute

The SECONDARY attribute will be recognized but ignored.

The NORMAL and ABNORMAL Attributes

The attributes NORMAL and ABNORMAL are not implemented. If declared, they will be ignored.

BASED Variables

The pointer variable explicitly or implicitly qualifying a based variable must be a non-based unsubscripted scalar pointer identifier.

The BASED attribute must be followed by a pointer identifier in parentheses.

The OFFSET attribute must be followed by an identifier in parentheses.

The variable named in the OFFSET attribute must be an unsubscripted level 1 based area.

Offset variables may not be used in any SET option or in any explicit or implicit based variable qualifier.

A based variable may not have the INITIAL attribute.

A based label array cannot be initialized by means of subscripted label prefixes.

A based structure can have either:

1. One adjustable array bound, or
2. One adjustable-length bit or character string.

Based structure with one adjustable array bound: This is permitted only when there are no adjustable strings in the structure. The bound must conform to the following rules:

1. It must be of the form:

   X REFER (Y)

   where X is an unsubscripted fixed binary variable of default precision, and

   Y is an unsubcripted fixed binary variable, of default precision, which is:

   (a) part of the structure, and
   (b) not associated with an explicit pointer qualifier.

2. It must be the upper bound of the leading dimension (including inherited dimensions) of the element with which it is declared.
3. The structure member with which the bound is declared must be, or must contain, the last base element in the structure.

For example:

```
DCL 1 PARTS_LIST BASED (P),
  2 FIRM_NAME CHAR (40),
  2 REF_NO FIXED BINARY,
  2 FIRM_ADDRESS,
    3 STREET_TOWN CHAR (50),
    3 COUNTRY CHAR (30),
  2 STOCK (20: N REFER(REF_NO),100:200),
    3 NUMBER,
      4 HERE FIXED (10,0),
      4 ORDERED,
        5 PROVISIONAL FIXED (10,0),
        5 CONFIRMED FIXED (10,0),
    3 COST FIXED (5,0);
```

Based structure with one adjustable-length bit or character string: This is permitted only when there are no adjustable array bounds in the structure. The string must conform to the following rules:

1. It must be scalar.
2. It must be the last element in the structure.
3. The length must be declared in the form:

   X REFER (Y)

where X and Y are as described above.

For example:

```
DCL 1 TYPE_OF_HOUSE BASED (P),
  2 NUMBER_OF_FLOORS FIXED (2,0),
  2 AREA FIXED BINARY,
  2 RATES_CODE CHAR (N REFER (AREA));
```

Note: Pointer and AREA data types must be ALIGNED.

OFFSET and POINTER Built-In Functions

The OFFSET and POINTER built-in functions cannot be specified explicitly. However, if the value of an offset variable is assigned to a pointer variable, or a pointer value to an offset variable, the necessary conversion is implicit in the assignment.

Based Variable Declaration

A pointer variable must be included in a based variable declaration. (This is the pointer that will be set in the absence of a SET option from a LOCATE or an ALLOCATE statement referring to the based variable.)

ONCOUNT Built-In Function

This function is supported only for Model 91 requirements.

APPENDIX I: MODEL 91

When using the (F) Compiler to produce object programs for execution on a System/360, Model 91 the option M91 must be specified.

In the following discussion, the terms exception and interrupt are used. An exception is a hardware occurrence (such as an overflow error) which can cause a program interrupt. An interrupt is a suspension of normal program activities. There are many possible causes of interrupts, but the following discussion is concerned only with interrupts resulting from hardware exceptions.

The Model 91 is a high-speed processing system in which more than one instruction is executed concurrently. As a result, an exception may be detected and an interrupt occur when the address of the instruction which caused the exception is no longer held in the central processing unit. Consequently, the instruction causing the interrupt cannot be precisely identified. Interrupts of this type are termed imprecise. When an exception occurs, the machine stops decoding further instructions and ensures that all instructions which were decoded prior to the exception are executed before honoring the exception. Execution of the remaining decoded instructions may result in further exceptions occurring. An imprecise interrupt in which more than one exception has occurred is known as a multiple-exception imprecise interrupt. When the M91 option is used, the compiler and library permit processing of imprecise interrupts.

When the M91 option is specified, the following occurs:

1. The compiler inserts 'no-operation' instructions at certain points in the program to localize imprecise interrupts to a particular segment of the program, thus ensuring that interrupt processing results in the action specified in the source program. (A 'non-operation' instruction is an Assembler Language 'flush' instruction of the form BCR x,0, where x is not equal to zero. This instruction is implemented in the Model 91 in such a way that its execution is delayed until all previously decoded instructions have been executed.) The situations in which the 'no-operation' instructions are generated are:

   a. Before an ON statement.

   b. Before a REVERT statement.

   c. Before compiled code to set the SIZE condition.

   d. Before compiled code to change prefix options.

   e. For a null statement. (This feature provides the programmer with source language control over the timing of program interrupts.)

   f. Before every statement of the STMT option (in the PARM field of the EXEC statement) is used. (This is an important debugging tool.)

2. An external symbol dictionary (ESD) entry is created for the PL/I library module IHEM91, which is required only when an object program is to be executed on a Model 91. The module is included when the object module is link-edited, and it is called when an imprecise interrupt is detected. Module IHEM91 provides the facilities for:

   a. Detecting multiple-exception imprecise interrupts.

   b. Setting the value that is returned by the ONCOUNT built-in function.

   c. Raising the appropriate PL/I conditions.

The order of processing the exceptions is as follows:

1. PL/I conditions in the order:

   UNDERFLOW

   FIXEDOVERFLOW or SIZE

   ERROR if system action is required for either FIXEDOVERFLOW or SIZE

   OVERFLOW

   ERROR if system action is required for OVERFLOW

   ZERODIVIDE

   ERROR if system action is required for ZERODIVIDE

   Note: The conditions FIXEDOVERFLOW and SIZE cannot occur together since the

same hardware condition raises both of them.

2. Hardware exceptions in the order:

   data

   specification

   addressing

   protection

Conditions and exceptions are raised in the above order until one of the following situations occurs:

1. A GO TO statement arising out of an ON-unit is executed. All other exceptions will then be lost.

2. The ERROR condition is raised. If the program is terminated as a result of this action (i.e., system action causing the ERROR condition to be raised, followed by the FINISH condition), messages will be printed to indicate the nature of the unprocessed exceptions. The exceptions themselves will not be processed.

   When an interrupt results from multiple exceptions, only one of the PL/I conditions is raised for each type of exception that has occurred.

When a multiple-exception imprecise interrupt occurs, the ONCOUNT built-in function provides a binary integer count of the number of exception types, including the current one, remaining to be processed. (The count does not include PL/I ON conditions.) If the ONCOUNT function is used when only a single exception has occurred, or if it is used outside an ON-unit, a count value of zero is indicated.

Programs compiled with the M91 option can be executed on other System/360 models supported by the System/360 Operating System.

* PROCESS statement 30

ABNORMAL attribute 273
ADV, format of 162
aggregate length table 35
ALIAS linkage editor statement 44
ALIGNED/UNALIGNED attributes 14,103
  See also structure mapping 164
alignment
  of FIXED and FLOAT BINARY variables 126
alignment attributes, use of 122
area data, format of 139
area dope vector 161
AREA ON-unit, exit from 112
AREA, size of 151
AREA storage allocation and release 174-175
arguments, method of presenting 161
array arithmetic - programming hints .. 111
array assignment 109
array dope vector, format of 162
array expressions - nesting levels 150
array or structure elements
  matching with EDIT-directed format lists 113
arrays
  listed attributes 33
  order of transmission 113
arrays bounds permitted 149
arrays, format of 140
ASA carriage control character 68
assembler subroutine
  variable-length argument list for ... 82
assignment of variables 107
assignment, efficient multiple assignments 117
assignment to VARYING string 111
assignments of variables
  programming hints 109,126
ATTACH macro 186
ATR/NOATR option 26
attribute and cross reference table 33
  example 34
attributes, declaration of 105
attributes, factoring of 149
AUTOMATIC variables, declaration 105

BACKWARDS attribute 51,70
based variables allocation 115
based variables using REFER
  forcing correct boundary alignment . 156
batched compilation 30
  programming example 31
batched compilation output data sets ... 24
BCD/EBCDIC option 27
BEGIN blocks, use of 115
binary data alignment 126
BIT data 137
BIT strings, efficient use of 117
BLKSIZE
  for blocked INDEXED data sets 71
BLKSIZE DCB subparameter 59
block initialization
  prologues and epilogues 159
block invocation
  recursive invocation count pseudo-register 36
block management at run-time 158-160
block nesting, limits 149
block size for records 61
block size, maximum 142
blocks in a compilation 149
bounds permitted for arrays 149
BUFFERS option 143-144
buffers, number of 65
BUFNO DCB subparameter 59

CALL macro 186
calling sequence between modules 160
card reader/punch mode 66
card reader/punch stacking 66
cataloged data sets 19
  definition of 17
cataloged procedures, use of 48
  PL1DFC 183,19,49
  PL1LFC 183,19,49
  PL1LFCL 183,19,49
  PL1LFCLG 184,19,50
  PL1LFLG 184,19,50
CELL attribute 273
chained scheduling 65
CHANGE linkage editor statement 44
changes at Fourth version, Release 17 .. 13
channel programs, number of 65
character code for object program 142
CHARACTER data 137
CHARACTER PICTURE data 137
CHAR60/CHAR48 OPTION 28
CHECK condition 154
  for DATA-directed I/O 153
CHECK lists, maximum number of items .. 153
checkpoint/restart 83-87
CM
  See external symbol dictionary
COBOL option 144
COBOL structures length of 35
COBOL-PL/I data interchange 103
CODE DCB subparameter for paper tape ... 65
coded arithmetic data 135
collating sequence 154
COLUMN, used for non-PRINT file 142
communication between object modules ... 81
COMP/NOCOMP option 29
compatibility
  between versions of the compiler 14
  expression evaluation - concatenation 14
  multitasking 14
  PACKED now removed from the language 14
  RECURSIVE procedures 14
  REENTRANT procedures 14
  structure mapping 14-15
compilation job control 30
compilation speed improvements 115
compile-time facilities 123
compile-time processor 147-148

compile-time processor, listing of input 32
  additional data in cols 73-80 of output ... 33
compiler
  compatibility between different versions ... 14
  compiler completion codes ... 30
  features of different versions ... 11-14
  summary of Release 17 changes ... 13
compiler ddnames ... 20
compiler interface with the supervisor . 79
compiler options ... 14,24
compiler options, listing of ... 32
compiler output ... 31-40
  aggregate length table ... 35
  attribute and cross reference table . 33
  external symbol dictionary ESD listing ... 36
  source program listing ... 32
  STATIC INTERNAL storage map ... 37
compiler phases, language features calling particular optional phases ... 115
compiler processing ... 20
compiler storage requirements ... 23
compiler-generated names, number of ... 150
compiler-input/output blocking ... 23
COMPLETION built-in function ... 153
COMPLETION pseudo-variable ... 153
completion codes
  for abnormal step termination ... 133
  for batched compilation ... 31
  for the compiler ... 30
COMPLEX FIXED BINARY data ... 136
COMPLEX FIXED DECIMAL data ... 136
COMPLEX FLOAT BINARY data ... 136
COMPLEX FLOAT DECIMAL data ... 136
compound source statement numbering ... 32
concatenation of data sets ... 155
concatenation, priority of ... 14
condition prefixes, scope of ... 127
conditional job step execution ... 83
CONSECUTIVE data set access ... 70
CONSECUTIVE data set creation ... 70
CONSECUTIVE data set language features . 52
CONSECUTIVE data sets ... 51
CONSECUTIVE organization ... 143
constants returned by procedures ... 150
constants, floating-point, size of ... 150
constants, precision and length ... 150
constants, sterling ... 150
constants, string, size of ... 150
control program options ... 78
control record for batched compilation . 31
control sections
  procedure invocation ... 160
  section definition for IHEMAIN ... 37
  section definition for IHENTRY ... 37
control variables in DO-groups ... 127
CONVERSION ON condition ... 154
CONVERSION ON-units ... 112
conversions
  hints for efficient execution .. 116-117
  programming hints ... 108
  supported by in-line code ... 118-120
CSECTS
  See control section
CTLASA option ... 68
CTLASA record I/O control characters .. 144
CTL360 option ... 68
CTL360 record I/O control characters .. 144
cyclinder overflow area ... 67
CYLOFL subparameter ... 67

data aggregates - programming hints ... 111
data attributes
  for efficient execution ... 116
data control block DCB parameter ... 46
DATA directed I/O, CHECK condition ... 153
data element descriptor DED ... 137
data interchange ... 103
data management access methods ... 78
data set creation and access ... 70
data set definition ... 58
data set, definition of ... 17
data set label, ordering sequence for DCB ... 141
data set names
  generation data groups ... 18
  indexes of ... 18
  qualified or unqualified ... 18
  restriction ... 17
data set naming ... 58
data set organization ... 67
  types of ... 143
data set positioning, magnetic tape 143-144
DATA-directed I/O
  maximum length of qualified name ... 142
  maximum number of elements in list . 150
DATAFIELD built-in functions
DATAFIELD string length ... 152
DATE function ... 117
DCB creation ... 141
DCB parameter ... 46,59
DCB subparameters ... 60
DCB, file opening order of completion . 141
DD statement ... 20
DD statement, ordering sequence for DCB 141
DD statements ... 19,184
  modified in cataloged procedures ... 185
ddnames
  for PL/I SORT ... 96
debugging facilities (testing) ... 129-134
DECK/NODECK option ... 27
declaration of external indentifiers .. 105
DED data element descriptor ... 137
DEFINED items
  alignment attributes ... 122
deleted records ... 66
  SEQUENTIAL UPDATE of INDEXED data sets ... 72
delimiter (/) statement ... 20
DEN DCB subparameter ... 65
device classes ... 21
devices
  execution device choice ... 46
diagnostic messages
  See messages
dictionary blocksize ... 29
dictionary, reducing the size of .. 120-122
dimensions, maximum number of ... 149
DIRECT access ... 51
  See also INDEXED and REGIONAL data sets
DIRECT UPDATE files
  use of NOWRITE option ... 122
DISP parameter ... 59

DISPLAY message length 141
DISPLAY statement 132
division
  FIXED division programming hints ... 109
DO groups - programming hints 110
DO groups, use of 127
dope vector for string data 136
dope vectors, creation of 140
DSA dynamic storage area 159
DSNAME parameter 58
DSORG subparameter 67
dummy records
  in INDEXED data sets 53
  in REGIONAL data sets 58
dumping storage in multitasking 89
dumps
  See storage dumps
dynamic invocation of PL/I compiler ... 186
dynamic storage area DSA 159

E format data, maximum size 142
EBCDIC/BCD option 27
EDIT-directed I/O 154
  maximum length of E format data 142
EMPTY built-in function, use of 112
END statement 32
entry names as arguments 154
entry parameter procedures, max number 154
entry points
  declaration of attributes 105
ENVIRONMENT attribute 142
ENVIRONMENT attribute, use of 124
epilogue subroutine
  See dynamic storage area
ER
  See external symbol dictionary
ERROR condition 128
error messages
  See messages
ESD (external symbol dictionary) 36
ESD listing 36
event data, format of 139
EVENT option 57,146
  in a DISPLAY statement 272
  in a multitasking environment 88
  in multitasking I/O 91
  with REGIONAL(3) data sets 57
EXEC statement 20
execution speed, improvements 116
EXIT statement
  in a subtask 93
exponentation 154
expression evaluation 108
  maximum number of temporary results 151
EXTDIC/NOEXTDIC option 29,148
extended search limit 67
external indentifiers
  restrictions 105
external symbol dictionary listing 36
EXTREF/NOEXTREF option 26

F-format records 143
factoring of attributes 149
FCB (file control block) 81
file and ddname relationship 141
file attributes
  use of UNBUFFERED v BUFFERED 125
file attributes, specified in DD card .. 59
file control block 81
file opening and closing space
 requirements 82
files
  for both input and output
   (programming hint) 112
files and data sets 51
FINISH condition 128
FIXED BINARY, maximum precision 152
FIXED built-in function, default 152
FIXED data scale factor for 152
FIXED DECIMAL data with even precision
FIXED DECIMAL variables 126
FIXED DECIMAL, maximum precision 152
FLAG option 28
FLOAT BINARY, maximum precision 152
FLOAT built-in function, default 152
FLOAT DECIMAL variables 126
FLOAT DECIMAL, maximum precision 152
floating-point constants, size of 150
floating-point variable, magnitude of . 152
FORTRAN-PL/I data interchange 103
function values returned 151

generation data groups 18,19
GENERIC attribute
  limit of family members and
   arguments 149
GENKEY option 145

hardware for PL/I (F) Compiler 78

IDENT option 272
identifiers, length of 153
IHEQERR
  See external symbol dictionary
IHEQINV
  See external symbol dictionary
IHEQTIC
  See external symbol dictionary
IHESADA
  See external symbol dictionary
IHESADB
  See external symbol dictionary
in-line code for conversions and string
 handling 118
INCLUDE STATEMENT
  data set requirements 147
  use of 123
INDEXAREA parameter 145
INDEXED data set access 71
INDEXED data set creation 71
INDEXED data set language features 53
INDEXED data set, DIRECT access 73
INDEXED data set, SEQUENTIAL access 72
INDEXED data sets 51
  KEY condition raised for lost record 113
  maintenance 125
  records deleted by SEQUENTIAL UPDATE 72
  secondary storage restriction 104
INDEXED organization 143
indexes for INDEXED data sets 52
INITIAL attribute 272-274
initialization of program variables ... 107
initialization of variables, hints 126
input/output 91
  I/O bound programs 118
INDEXED data sets 51

synchronization in multiprocessing 94
input/output - programming hint ... 112-115
input/output conventions 141-147
input/output error recovery 126
INSERT statement 43
instruction sets 78
interrupts
  M91 option 277
interrupts, object time handling .. 160,129
interval timing options 79
invocation of PL/I dynamically 186

job control for linkage editor 45
job control statements
  example of compile, link and execute
   procedure 46
job priority 90
JOB statement 20
job step
  definition of 17

KEY condition
  not detected using LOCATE statement 114
KEY option 56
key position, relative 67
KEYLEN DCB subparameter 66
keys, for INDEXED data sets 51
keys, for REGIONAL data sets 55

LABEL attribute, maximum number 152
label data, format of 138
LABEL parameter 59
LABEL variables in structures
  with INITIAL and LIKE attributes ... 146
language not supported 272
LD
  See external symbol dictionary
LEAVE option 143-144
length of compiled modules 81
length of identifiers 153
length of structures and arrays
  See aggregate length table
LET linkage editor option 44
library module linkage conventions 161
library opening and closing modules
 storage requirements 82
LIMCT DCB subparameter 67
LINECNT option 26
  for object program listings 38
LINECNT/NOLINECNT option 26
LINESIZE option 68
LINESIZE, minimum and maximum 141
LINESIZE, used for non-PRINT file 142
link editing into a private library 41
link editing, module name priority 41
linkage editor 18
  updating private procedure libraries 18
linkage editor control statements 44
linkage editor diagnostics 45,46
linkage editor options 44
linkage editor output 45
linkage editor overlay processing 42
linkage editor processing 40
linkage editor, additional libraries ... 41
linkage editor, ddnames 41
linkage editor, input and output 40
LIST linkage editor option 44
LIST/NOLIST option 26
load module 18
load module cross reference table 45
load module map 45
load module, maximum size 45
LOAD/NOLOAD option 27
locate mode I/O 76
LOCATE mode I/O ON conditions 155
locate mode I/O, use of 114
logical record length 62
LRECL
  for blocked INDEXED data sets 71
LRECL DCB subparameter 59

MACDCK data set 20
MACDCK/NOMACDCK option 30
MACRO option 123
macro processor
  See compile-time processor
MACRO-NOMACRO option 29
magnetic tape conversion 7-track 66
MAIN option 81,148
MAP linkage editor option 44
mapping of structures 164-175
master index specification 67
MAX function 152
messages
  compiler diagnostic messages 39,188-260
  execution error messages 260-271
  FLAG options 28
  linkage editor 45,46
  to and from the operator 48
MFT 79
MIN function 152
MOD function 152
MODE DCB subparameter 66
model 91 277
modifying cataloged procedures 184
move mode I/O 76
multiple assignments, limitations 151
multiple closure statement numbering ... 32
multiple WAIT option 79
multitasking 87-94
  I/O synchronization 92
multitasking, Release 16 changes 14
multiprocessing 94
multiprogramming, with a fixed number
 of tasks 79
multiprogramming, with a variable
 number of tasks 79
MVT 79
M91/NOM91 option 30

NAME linkage editor statement 44
names generated by the compiler 150
NCAL linkage editor option 43
NEST/NONEST option 30
NTM DCB subparameter 67
null (//) statement 20

object deck data set 20
object deck serialization 40
object module 18
object module data set 20
object module listing 38
object module output, contents of 40
object program initialization 160
object program organization 158-175
object program output

object time messages 48
operator messages 48
record oriented 47
stream oriented 47
object-time interface with the supervisor 79
OBJNM option 26
OFFSET function and attribute 274
offset variable, format of 138
ON conditions 111
LOCATE mode I/O restrictions 155
ON statements in recursive contexts ... 155
ON-codes 129
ON-unit entry with SIGNAL 111
ON-units
in multitasking 89
ON-units, limit to active number 154
ONCHAR pseudo-variable 154,112
ONCODE built-in function, use of 128
ONCOUNT function 275
ONKEY built-in function 125
ONSOURCE built-in function 154
ONSOURCE pseudo-variable 154,112
ONSOURCE string length 152
operating system requirements 78
operator console 78
OPLIST/NOOPLIST option 26
OPT option 27
OPTCD=C DCB subparameter 65
OPTCD=L DCB subparameter 66
OPTCD=M DCB subparameter 67
OPTCD=U DCB subparameter 66
OPTIONS attribute 148
overflow area, independent 67
OVERFLOW condition in exponentiation .. 154
overlay program
linkage editor cross reference table 45
overlay programs, creation of 42
overlay techniques - examples 42-44
overriding cataloged procedures 124
overriding EXEC statement parameters .. 184

PACKED attribute 14
PAGESIZE, maximum 141
paper tape CODE subparameter 65
parameter lists
declaration of attributes 105
parameters, maximum number of 149
parameters, method of passing 161
PARM parameter option list format 104
PARM parameter override 184
partitioned data set
definition 19
PCP 78
PDS (partitioned data set) 19
PGMNAME
See external symbol dictionary
PGMNAMEA
See external symbol dictionary
PICTURE attribute
impact on execution speed 116
PICTURE attribute, maximum size 152
picture checking (on assignment) 109
PICTURE declarations
with decimal points 105
pictured data 137
PL/I language not supported 272
PL/I library 83
PL/I modules combined with modules from other languages 82
PL/I preprocessor
See compile-time processor
PL/I SORT 95-103
PL/I SORT - multitasking 103
PL1DFC 183,19,49
PL1LFC 183,19,49
PL1LFCL 183,19,49
PL1LFCLG 184,19,50
PL1LFLG 184,19,50
POINTER function 275
pointer variable, format of 137
pointer variable, reuse of 115
POSITION attribute, maximum value 152
PR
See external symbol dictionary
precision, specified for efficient execution 116
primary control program 78
primary string array dope vector 164
PRINT files 68
printed listings from compiler 31
printer character sets for PL/I 78
printer spacing control 66
priorities in PL/I 90
PRIORITY pseudo-variable, function, or option 90
PROCESS card 30
program control section in ESD 36
program csects
See control sections
program size estimation 81
program testing 128-134
programming in a multitasking environment 88
programming techniques 104-127
common errors and pitfalls 104-115
efficient programming 115-127
prologue and epilogue subroutines 159
PRTSP DCB subparameter 66
PRTY parameter 90
pseudo-register vector (PRV) 95,158
PUT DATA statement limitations 113

REAL FIXED BINARY data 135
REAL FIXED DECIMAL data 135
REAL FLOAT BINARY data 135
REAL FLOAT DECIMAL data 135
REAL PICTURE data 137
RECFM DCB subparameter 59,62
record alignment 155
RECORD condition 125
record data sets for printing 68
record formats 61,64,129
RECORD-oriented input/output 125
hints for efficient performance 118
LOCATE mode 76
printer, punch control characters .. 144
record format constraints 64
record length 62,142
RECURSIVE attribute 14
recursive use of a procedure 105
REENTRANT attribute 14
REENTRANT option 148
REFER attribute
forcing correct boundary alignment . 156
REGIONAL data set access 74

DIRECT access 75
replacement of records 75
SEQUENTIAL access 74
REGIONAL data set creation 73
DIRECT creation 74
SEQUENTIAL creation 74
REGIONAL data set language features 54
REGIONAL data set organization 55
REGIONAL data sets
source key last 8 bytes 113
REGIONAL organization 143
REGIONAL(1) data sets 56
REGIONAL(2) data sets 56
REGIONAL(3) data sets 57
REGIONAL(3) files
in multitasking, with EVENT option .. 91
register usage 161
REPLACE linkage editor statement 44
REPLY message length 141
REPLY option 132
restarts
See checkpoint/restart
return code setting by PL/I 83
return codes
in multitasking 89
return values
declaration of attributes 105
REWIND option 143-144
REWRITE statement
in locate-mode I/O 114
RKP DCB subparameter 67
run-time block management 158-160

SADV 163
scale factor for FIXED DATA 152
scope of variables in multitasking 88
SD
See external symbol dictionary
SDV string dope vector 136
SDV, format of 161
SECONDARY attribute 273
sequential access I/O 51
SEQUENTIAL access of REGIONAL data sets 56
sequential data set
definition 19
SEQUENTIAL files
in multitasking I/O 91
SETS list, size of 152
SIGNAL statement 129
SIZE CONDITION
for even precision FIXED DECIMAL
data 110
SIZE disabled 108
size of load module 45
SIZE option 25
SIZE requirements 23
SKIP, used for non-PRINT file 142
SNAP option, use of 131
in multitasking 89
SNAP SYSTEM 132
SORMGIN option 27,135
SORT, PL/I SORT 95-103
source keys, REGIONAL data sets 55
source program listing 32
source statement numbering 32
source statement record format 135
SOURCE/NOSOURCE option 29
SOURCE2/NOSOURCE2 option 29
space allocation
for REGIONAL data sets 73
SPACE parameter 59
space requirements for compiler data
sets 22
spanned VS-format or VBS-format records 61
spanned records 125
spanned records, use of 64
STACK DCB subparameter 66
standard file names 113
SYSIN implicit use 113
statement numbering 32
listed in attribute table 33
listed in cross reference table 33
statement size 148-149
statements, number in a procedure 148
STATIC INTERNAL control section
name in ESD listing 36
STATIC INTERNAL storage map 37
STATIC variable in overlay segment 104
sterling constants 150
STERLING PICTURE data 137
STMT option 48
STMT/NOSTMT option 28
STOP statement
in a subtask 93
storage dumps
at execution time 132
storage dumps in multitasking 89
storage for file opening and closing ... 82
storage requirements
for PL/I SORT 95
storage requirements for object program 33
storage, hints for the use of 122-123
STREAM data set creation and access 70
STREAM-oriented I/O 125
record formats 63
access method 78
based variable restriction 115
string array dope vector 163
STRING built-in function 153
string contants, size of 150
string data 136
string dope vector SDV 136
string dope vector, format of 161
string functions
supported by in-line code 120-122
string handling
in multitasking 92
string handling - programming hints ... 111
string handling, for improved speed ... 117
strings
for GET LIST or DATA 113
strings, VARYING
intermediate result string length .. 151
structure dope vector, format of 163
structure expressions - nesting level . 150
structure mapping 164
PACKED and UNALIGNED compared 14-15
structures, format of 140
subscript variables 117
subscripted identifiers, length of 153
SUBSTR pseudo-variable 111
switches, efficient use of 117
SYSIN DD statement 20
SYSLIB DD statement 18,20
SYSLIB DD, use of 123
SYSPRINT 20

use in STREAM I/O in multitasking ... 91
SYSPRINT implicit use 113
SYSPUNCH/SYSLIN in batched compilation . 24
system requirements 78
SYSUT1 20
SYSUT3 20
SYS1.LINK modules 83

tab controls for STREAM I/O 69
task data, format of 138
TASK option 148
task priority 90
task termination 93
tasking
pseudo-register for 36
tasking
See multitasking
temporary results, maximum permitted .. 151
temporary workspace VDAs 159
testing programs 128-134
text block size 29
TIME option 79
time taken for compilation 40,79
timer feature 78
TITLE option 51,141,146
trace of active procedures 131
TRANSMIT condition 129
TRTCH DCB subparameter 66

U-format records 143
UCS printer, suppress TRANSMIT 66
UNALIGNED attribute 14
effect on structure mapping 166
See also structure mapping 164
UNDEFINEDFILE condition
for a stream file 113
possible causes 112
UNIT parameter 58
UNSPEC built-in function
length of bit-string returned 152
UNSPEC pseudo-variable - programming hint 111
UPDATE, INDEXED data sets 52
utility data sets 20

V-format variable-length records ... 61,143
validity check direct-access 66
variable data area VDA 159
variable-length V-format records 61
variable-length records
use of spanned records 125
variables, maximum number of 148
VARYING string, assignment to 111
VBS-format spanned-blocked records . 61,143
VDA
for temporary workspace 159
volume
definition of 17
VOLUME parameter 59
VS-format spanned records 61,143

WAIT statement 146
in multitasking 88-89
warning messages
See messages
WHILE clause, use of 127

XCAL linkage editor options 44
XREF linkage editor option 44
XREF/NOXREF option 26

48-character set
semicolon in DATA-directed I/O 113
48-character set, use of reserved words 142
60-character set 28

READER'S COMMENT FORM

Form C28-6594-4

IBM System/360
PL/I (F) Programmer's Guide

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

	Very Good	Good	Fair	Poor	Very Poor
As a reference source:	.....	.....	.....	.....	.....
As a text:	.....	.....	.....	.....	.....

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM CORPORATION

112 EAST POST ROAD,
WHITE PLAINS, N.Y. 10601.

Attention: Department 813 (HP)

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM System/360 Printed in U.S.A. C28-6594-4

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]