IBM PL/I for VSE/ESA

Compile-Time Messages and Codes

Release 1

SC26-8059-00

IBM PL/I for VSE/ESA

Compile-Time Messages and Codes

Release 1

SC26-8059-00

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page iv.

First Edition (April 1995)

This edition applies to Version 1 Release 1 of IBM PL/I for VSE/ESA, 5686-069, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department J58
P.O. Box 49023
San Jose, CA, 95161-9023
United States of America

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

Trademarks

About this book

This publication lists all compile-time messages and codes from the IBM* PL/I for VSE/ESA* (PL/I VSE) compiler. Most of these messages have information illustrating the detected condition and suggesting appropriate corrective action. Error numbers, restriction numbers, and compiler return codes are provided to further aid you in determining a problem.

Users of this book are those application programmers who code and compile PL/I programs. This book is intended to help you find a resolution for the message or code you received while performing one of the above tasks.

Using your documentation

The publications provided with PL/I VSE are designed to help you do PL/I programming under VSE. Each publication helps you perform a different task.

Where to look for more information

For information about the PL/I VSE library, see Table 1.

Table 1. How to use the publications you receive with PL/I VSE

To...	Use...
Evaluate the product	*Fact Sheet*
Understand warranty information	*Licensed Program Specifications*
Install the compiler	*Installation and Customization Guide*
Understand product changes and adapt programs to PL/I VSE	*Migration Guide*
Prepare and test your programs and get details on compiler options	*Programming Guide*
Get details on PL/I syntax and specifications of language elements	*Language Reference* *Reference Summary*
Diagnose compiler problems and report them to IBM	*Diagnosis Guide*
Get details on compile-time messages[1]	*Compile-Time Messages and Codes*

Note:

1. For details on run-time messages, see the LE/VSE library.

You might also require information about IBM* Language Environment* for VSE/ESA* (LE/VSE). For information about the LE/VSE library, see Table 2.

Table 2 (Page 1 of 2). How to use the publications you receive with LE/VSE

To...	Use...
Evaluate Language Environment	*Fact Sheet* *Concepts Guide*
Install LE/VSE	*Installation and Customization Guide*
Understand the LE/VSE program models and concepts	*Concepts Guide* *Programming Guide*

Table 2 (Page 2 of 2). How to use the publications you receive with LE/VSE

To...	Use...
Prepare your LE/VSE-conforming applications and find syntax for run-time options and callable services	*Programming Guide* *Reference Summary*
Debug your LE/VSE-conforming application and get details on run-time messages	*Debugging Guide and Run-Time Messages*
Diagnose problems that occur in your LE/VSE-conforming application	*Diagnosis Guide*
Understand warranty information	*Licensed Program Specifications*

For the complete titles and order numbers of these and other related publications, see the "Bibliography" on page 115.

What is new in PL/I VSE

This is a major new release of PL/I, containing many new features and facilities. It brings to VSE many of the functions of the MVS & VM version of PL/I (IBM SAA* AD/Cycle* PL/I MVS & VM), while retaining close source compatibility with the DOS PL/I Optimizing Compiler (DOS PL/I).

PL/I VSE enables you to integrate your PL/I applications into IBM Language Environment for VSE/ESA (LE/VSE). In addition to PL/I's already impressive features, you gain access to LE/VSE's rich set of library routines and enhanced interlanguage communication (ILC) with IBM COBOL for VSE/ESA (COBOL/VSE).

IBM Language Environment for VSE/ESA support

PL/I VSE provides the following functions in the LE/VSE area:

Interlanguage communication (ILC) support:

- Object code produced by PL/I VSE Release 1 can be linked with object code produced by other LE/VSE-conforming compilers (currently only COBOL/VSE).
- PL/I VSE programs can fetch COBOL/VSE phases.
- COBOL/VSE programs can fetch PL/I VSE phases.

Note: PL/I VSE does not support ILC with:

- FORTRAN
- RPG
- DOS/VS COBOL
- C/370*

Limited ILC support is provided for VS COBOL II at Release 3.2 or later.

Common support for multiple operating environments:

- Some of the restrictions on PL/I coding in the CICS* environment have been lifted.
- Procedure OPTIONS option FETCHABLE can be used to specify the procedure that gets control within a fetched phase.
- CEETDLI is supported in addition to PLITDLI and EXEC DLI.

- LE/VSE services provide storage management and condition handling support, as well as PLIDUMP and MSGFILE support for PL/I messages and other output.
- By default, only user-generated output is written to SYSLST. All run-time generated messages are written to MSGFILE.
- ERROR conditions now get control of all system abends. The PL/I message is issued only if there is no ERROR on-unit or if the ERROR on-unit does not recover from the condition via a GOTO.
- Selected items from PL/I Package/2 (the PL/I product for OS/2*) are implemented to allow better coexistence.
  - Limited support of OPTIONS(BYVALUE and BYADDR)
  - Limited support of EXTERNAL(environment-name) allowing alternate external names
  - Limited support of OPTIONAL arguments/parameters
  - Support for %PROCESS statement
  - NOT and OR compiler options

Product packaging:

- All PL/I VSE resident library routines are now packaged with LE/VSE, and are loaded at run time rather than link-edited with the application program. Changes to the resident library no longer require PL/I programs to be re-linked.
- At link-edit time, you have the option of getting math results that are compatible with LE/VSE or with DOS PL/I.
- Installation enhancements are provided to ease product installation and migration.

For migration considerations, see the *PL/I VSE Migration Guide.*

Usability enhancements

These enhancements expand the PL/I language statements and options, PL/I data types, and compiler options, to make the language easier to use.

Enhanced double-byte character set (DBCS) support: This support introduces many enhancements that facilitate processing of GRAPHIC and mixed-character data and allows the source of the PL/I program to be in DBCS and/or the single-byte character set (SBCS), rather than only in SBCS.

Hexadecimal data constants: Constants for bit and character data can now be defined in hexadecimal notation, such that each *character* (0-9 and A-F) represents 4 bits.

Interface improvements for all (sub)systems: A new compiler option, SYSTEM, lets the programmer specify the target operating environment (of the generated object code), and the format of the parameters for the MAIN procedure.

Specification of compile-time options: You can specify compile-time options on the *PROCESS statement, a new %PROCESS statement, and in the PARM option of the EXEC IEL1AA JCL statement.

Linking after errors: The COMPILE compile-time option has been enhanced to allow linking to proceed after a severe error.

Run-time options: You can specify program run-time options in the PARM option of the EXEC JCL statement. PL/I VSE and LE/VSE will use these to control the execution of PL/I programs.

Passing parameters to the MAIN procedure: VSE JCL can also be used to pass a parameter to the MAIN PL/I procedure. A slash (/) separates the run-time options from the program parameter.

OPEN statement enhancements:

- There are new parameters on the PL/I OPEN statement that allow additional file attributes to be specified at file open time. These attributes are added to those in the file declaration.
- A vendor exit on the PL/I OPEN statement can be used to change the system logical unit number of the PL/I spill file.
- Data set name sharing for VSAM files, using the DSN option of the ENVIRONMENT attribute.

Date and time enhancements: A new built-in function, DATETIME, returns consistent date and time, including the four-digit year.

PL/I statement numbering options: A new compiler option, NUMBER, specifies that PL/I statement numbers will be derived from the sequence numbers in the program source deck, instead of being allocated sequentially.

Dynamic loading of external procedures: PL/I now supports the FETCH and RELEASE statements, to load external procedures into main storage at run time instead of having them link-edited with the MAIN procedure. (If these external procedures are PL/I, they must be compiled with PL/I VSE.)

New I/O facilities: PL/I VSE provides the following new I/O facilities:

- Support for REGIONAL(2) files
- Support for V and VS formats on REGIONAL(3) files
- Support for DELETE statement on REGIONAL files
- Support for multitrack search on REGIONAL files, using the LIMCT option of the ENVIRONMENT attribute
- Support for VSAM variable-length relative-record data sets (VRDS)
- Support for V format on consecutive unbuffered files
- Support for VS and VBS formats on consecutive buffered files

System programmer functions: A number of significant new features enhance PL/I as a system programming language:

- Support of additional program execution environments.

  PL/I can now be used for some system exit routines, such as the LE/VSE initialization exit.
- Additional support for pointers.

PL/I built-in functions are now available to perform extended operations on pointers, including pointer arithmetic.

- Additional support for entry variables.

  A new built-in function and pseudovariable, ENTRYADDR, allows programmers to manipulate entry point addresses of procedures.

Extended addressing enhancements

These enhancements exploit the large amounts of storage available in the VSE/ESA environment, making programming easier.

Addressing mode: PL/I VSE programs can be link-edited with AMODE(31) and RMODE(ANY).

Location of variables: PL/I variables can now be located above the 16-megabyte line.

Fullword array subscripts: Array bounds can now be in the range -2^{31} (-2,147,483,648) through $+2^{31}-1$ (+2,147,483,647). The associated built-in functions (such as LBOUND and HBOUND) now return FIXED BINARY(31) values.

AREA and aggregate sizes: An AREA can now have a maximum size of 2,147,483,647 ($2^{31}-1$) bytes.

An aggregate can now have a maximum size of 2,147,483,647 ($2^{31}-1$) bytes. For unaligned BIT arrays and aggregates that contain any unaligned BIT data (arrays or non-arrays), the maximum size is 268,435,455 ($2^{28}-1$) bytes.

These numbers include any control information bytes that might be needed.

Syntax notation

Special notation used in this book is as follows:

<	Shift-out character
>	Shift-in character

Chapter 1. Compile-time and macro preprocessor messages

The messages and codes in this chapter are the ones produced while your program is compiling. These messages are the compiler control messages, preprocessor messages, and compiler messages.

In this chapter, the messages are listed in numeric order.

- ***Compiler control*** messages (numbers 0002 through 0048) are mainly concerned with errors detected in the specification of compiler options in the PROCESS statement.
- ***Preprocessor*** messages (number 0001, numbers 0050 through 0229, number 573, and numbers 2233 through 2277) follow any listed output from the preprocessor, and, if compilation follows immediately, before any listed output from the compilation.
- ***Compiler*** messages (numbers 0230 through 0999) follow the source program and any other listings produced by the compiler.

Format of messages

Each message has a number of the form IELnnnnI. "IEL" indicates that the message is a preprocessor or compile-time message and "nnnn" is the number of the message. The final "I" indicates that no system operator action is required. After "nnnnI," most messages are followed by a severity code, either "I," "W," "E," "S," or "U."

These codes indicate the following:

VSE return code	Severity code	Description
0	I	An ***informational*** message calls attention to some aspect of the source program that might assist the programmer.
4	W	A ***warning*** message calls attention to a possible program error or to a potential failure to achieve full optimization. It does not imply a syntactical error in the source program. In addition to alerting the programmer, warning messages can help make the program more efficient.
8	E	An ***error*** message describes an error that the compiler corrected. The correction is likely to be successful.
12	S	A ***severe*** error message describes an error that the compiler attempts to correct, but might not do so successfully. Frequently, the correction consists of ignoring the incorrect section of the statement.
16	U	An ***unrecoverable*** error message describes an error that cannot be corrected by the compiler. Such errors, when discovered, normally force termination of the compilation. They are usually caused by a compiler, system, or setup error rather than by an error in the source program.

Except for compiler control messages, the compiler prints the messages in groups according to these severity levels.

Compiler control messages relating to compile-time options (for example, specified in PROCESS statements) do not have severity codes. These messages are produced by Phase AE during compile-time options processing; they appear after the compile-time options string, not at the end of the listing with the other messages.

The FLAG compiler option suppresses the listing of messages in the compiler listing. You can find a description of the FLAG option in the *PL/I VSE Programming Guide.*

Symbols in messages

Many of the messages reproduced in this publication contain symbols indicating where the compiler inserts information when it prints the message. The symbols used are:

D	An identifier used in the program
N	A decimal integer
P	Compiler phase
T	Text: up to 20 characters derived from the source program
T[1]	Text: up to 20 characters derived from the source program, being the first text insert in the message
T[2]	Text: up to 20 characters derived from the source program, being the second text insert in the message

You might also see these symbols:

<	Shift-out character
>	Shift-in character

Alternative forms of messages

Most messages are shown with their short form followed by their long form. In cases where the long and short messages are identical, the message appears only once. The format of compiler messages is controlled by the LMESSAGE and SMESSAGE compiler options.

The compiler might produce some messages with optional phrases. If a message has an optional phrase, it is listed in this publication with the phrases enclosed in square brackets. For example, message IEL0399I can print as:

```
SEMICOLON ASSUMED.
```

or as:

```
SEMICOLON ASSUMED AFTER T.
```

This message appears in this publication as:

```
IEL0399I E SEMICOLON ASSUMED [AFTER T].
```

A message can have other phrases included, such as:

```
PROLOGUE CODE
STATEMENT IGNORED
RESULTS OF PROLOGUE UNDEFINED
TO D
```

The term PROLOGUE refers to the instructions generated by the compiler for a PROCEDURE or BEGIN statement. These instructions perform the housekeeping that is required on entry to a procedure or begin block. Messages with references to the prologue indicate that the compiler detected the condition resulting in the message while generating the prologue code.

Conditions detected while generating the prologue code can include items such as the misuse of the INITIAL attribute or of parameters. Consequently, the presence of a reference to the prologue indicates that the error is not contained in the PROCEDURE or BEGIN statement itself, but in some other statement, such as a DECLARE statement, that follows the indicated statement.

Before calling IBM . . .

Before you call IBM for programming support for a compile-time error, recompile the program to get the following:

1. A listing of the source program
2. The job stream (source program and job control statements) in machine readable form

The *PL/I VSE Diagnosis Guide* gives the requirements for problem determination and APAR submission.

Messages IEL0001-IEL0995

IEL0001I U PROCESSOR ERROR NUMBER N DURING PHASE P.

Explanation: An error has occurred during preprocessing. Processing has been terminated. This error is due to a fault in the preprocessor, not the source program. A detailed explanation of error number N is given in "Error and restriction numbers (0 to 946) for IEL0001I, IEL0230I, and IEL0970I" on page 104.

Programmer Response: Rerun the job, and if the problem recurs, call IBM for programming support. Before calling IBM, refer to the introduction to this part of the publication for details of information that IBM will need in order to diagnose the problem.

IEL0002I U END-OF-FILE ENCOUNTERED ON INPUT FILE DURING COMPILER INITIALIZATION.

Example:

```
// EXEC IEL1AA
*PROCESS;
/*
```

Explanation: The compiler has encountered the end of file for the source program before reading a complete PL/I statement.

Programmer Response: Ensure that the source program immediately follows the EXEC IEL1AA statement. If a PL/I comment is the first statement in the source program, ensure that the "/*" is not in the first two positions of the record (columns 1 and 2) and are thereby assumed to be the job control end-of file delimiter. If the first statement in the source program is a PROCESS statement, ensure that the terminating semicolon is not in positions 73-80 of the first record.

IEL0003I THE FOLLOWING STRING IS NOT RECOGNIZED AS A VALID OPTION KEYWORD AND IS IGNORED - T.

Example:

```
* PROCESS ATRIBUTES;
          |________|
              T
```

Explanation: A character string in the PROCESS statement cannot be recognized as a valid keyword. In the above example, ATTRIBUTES is misspelled.

IEL0004I RIGHT PARENTHESIS MISSING IN SPECIFICATION OF FOLLOWING OPTION, BUT OPTION IS ACCEPTED - T.

Example:

```
*PROCESS FLAG(I;
```

IEL0005I THE SPECIFICATION OF THE FOLLOWING OPTION CONTAINS INVALID SYNTAX, DEFAULT ASSUMED FOR T.

Example:

```
* PROCESS SIZE)80K)...;
          |___|
            T
```

IEL0006I THE FOLLOWING OPTION IS DELETED, DEFAULT ASSUMED FOR T.

Explanation: The compiler, while processing the PROCESS statement, has encountered an option keyword that was deleted from the compiler at installation time. The default assumed for the option is the default specified for the option at installation time.

Programmer Response: If the option is essential, arrange to have the option restored to the compiler when the system is next generated or use the CONTROL option to restore the option temporarily.

IEL0011I SOURCE OR SEQUENCE MARGINS INCORRECTLY SPECIFIED. DEFAULTS ASSUMED FOR 'MARGINS' OR 'SEQUENCE'.

Example:

```
* PROCESS MARGINS(72,2,1);
```

Explanation: The left-hand margin position is to the right of the right-hand margin position. The default values assumed will be 2 and 72. The carriage control character position, if specified, is ignored.

IEL0012I CARRIAGE CONTROL CHARACTER OVERLAPS SEQUENCE FIELD OR SOURCE MARGINS. CONTROL CHARACTER IGNORED.

Example:

```
1. *PROCESS MAR(5,72,73) SEQ(73,80);
2. *PROCESS MAR(5,72,10);
```

Explanation: The carriage control character position, if used, must be outside the margins or sequence limits. The values of 5 and 72 are used for the margins, and the carriage control character position is ignored.

IEL0013I ARGUMENT NOT WITHIN PERMITTED RANGE. DEFAULT ASSUMED FOR OPTIONS -- MARGINS DEFAULT ASSUMED FOR OPTIONS -- LINECOUNT

Example:

```
1. *PROCESS MARGINS(2,103,1)...;
2. *PROCESS LINECOUNT(0)...;
```

IEL0014I UNMATCHED LEFT PARENTHESIS IN COMPILER OPTIONS SPECIFICATION. SUBSEQUENT OPTIONS IGNORED.

Example:

```
*PROCESS LINECOUNT(50;
```

IEL0015I SPECIFIED 'SIZE' OPTION IS LESS THAN MINIMUM REQUIRED BY COMPILER. DEFAULT ASSUMED.

Example:

```
*PROCESS SIZE(40K)...;
```

Explanation: The compiler requires at least 256K bytes of main storage.

IEL0016I SIZE SPECIFICATION TOO BIG. SIZE(MAX) ASSUMED.

Example:

```
*PROCESS SIZE(100000K);
```

Explanation: The value specified in the compiler option SIZE exceeded your storage resources. The maximum amount of storage available to you will be used for the compilation.

IEL0018I NAME FIELD TOO LONG. 'NAME' OPTION IGNORED.

Explanation: The total number of nonblank characters appearing in the name field of the specified NAME option is too large. Correct the specification and resubmit the job.

IEL0019I 'SIZE' OPTION IGNORED. VALUE IN FIRST MEMBER OF BATCH ASSUMED.

Explanation: It is not possible to alter the amount of main storage to be used by the compiler for the compilation of the second or subsequent external procedures in a batched compilation.

IEL0023I W NON-BLANK CHARACTERS FOLLOWING SEMICOLON IGNORED.

Example:

```
* PROCESS A,X; P;PROC OPTIONS(MAIN);
```

Explanation: Nonblank characters have been detected following the semicolon in the options list. Any comments and the first statement in the external procedure must follow a PROCESS statement on the following card (or line).

IEL0024I U SPILL FILE NEEDED BUT JCL STATEMENT INCORRECT. COMPILATION TERMINATED.

Explanation: If the spill file cannot be opened, message IEL0026I or message IEL0031I will be produced. If the spill file is needed, message IEL0024I is produced. The compilation can be completed without needing a spill file.

IEL0025I INVALID SYNTAX IN LAST OPTION OF 'PARM' FIELD. PROCESSING OF 'PARM' OPTIONS TERMINATED.

Example:

```
EXEC PGM = IELOAA,PARM = 'TA(KQ'
```

Explanation: In the above example, the right parenthesis has been omitted.

IEL0026I THE COMPILER SPILL FILE IS NOT DIRECT ACCESS. COMPILATION WILL TERMINATE IF SPILL FILE NEEDED.

Explanation: Compilation will be terminated if the spill file is needed and it is not on a direct access storage device. Compilation will *not* be terminated if the spill file is not needed.

IEL0027I U INCORRECT SPECIFICATION OF THE 'CONTROL' OPTION. COMPILATION TERMINATED.

Explanation: Either the CONTROL option has been specified syntactically incorrect or the wrong password has been supplied.

IEL0028I DELIMITER AT START OF STRING 'T' IS INVALID AND IS IGNORED.

Example:

```
* PROCESS 'FLAG(S)';
```

the quote (') characters are invalid.

Example:

```
* PROCESS (FLAG(S));
```

the first left parenthesis and the last right parenthesis are invalid.

IEL0030I U THE COMPILER INPUT FILE CANNOT BE OPENED.

Explanation: The compiler input file IJSYSIN cannot be opened, possibly because no JCL statement for the file has been provided. Compilation is terminated.

IEL0031I THE COMPILER SPILL FILE CANNOT BE OPENED. COMPILATION WILL TERMINATE IF SPILL FILE NEEDED.

Explanation: The compiler spill file IJSYS01 cannot be opened, possibly because no JCL statement has been provided. Compilation will *not* be terminated if the spill file is not needed.

IEL0032I S THE COMPILER PUNCH FILE CANNOT BE OPENED.

Explanation: The DECK or MDECK option has been requested but IJSYSPH cannot be opened, possibly because no JCL statement has been provided. Compilation continues with no punched output.

IEL0033I S THE COMPILER LINK FILE CANNOT BE OPENED.

Explanation: The OBJECT option has been specified but SYSLIN cannot be opened, possibly because no JCL statement has been provided. The NOOBJECT option is assumed and compilation continues.

IEL0034I U INSUFFICIENT MAIN STORAGE AVAILABLE. COMPILATION TERMINATED.

Explanation: The compiler has insufficient main storage to complete initialization. The partition is below the minimum required, or the buffers allocated to the compiler input/print/load/punch files might be too big.

Programmer Response: Retry with a larger partition, or increase the partition GETVIS storage available to the program.

IEL0035I 'NUMBER' OPTION BUT NO 'SEQUENCE'. DEFAULT SEQUENCE ASSUMED.

Example:

```
*PROCESS NUM NSEQ;
```

Explanation: The NUMBER option derives a line number from the sequence number in the position specified in the SEQUENCE option. If this position is not specified, the following position is assumed:

F-format records: last eight columns
U-format records: first eight columns
V-format records: first eight columns

IEL0036I THE FOLLOWING OPTION IS NOT SUPPORTED AND IS IGNORED - T.

Explanation: A valid PL/I option keyword has been specified, but is not supported by this compiler.

IEL0040I 'NOT' AND 'OR' OPTIONS CONFLICT. BOTH OPTIONS ARE IGNORED.

Explanation: The strings specified in the subfields of the NOT compiler option and the OR compiler option cannot contain any of the same characters.

Programmer Response: Change either the NOT option subfield, or the OR option subfield, or both, so that none of the characters in the subfield strings are the same. If these options are not explicitly specified by your source program, use the OPTIONS compiler option to see which characters are in conflict.

IEL0041I SEQUENCE FIELD OVERLAPS SOURCE MARGINS. DEFAULT SEQUENCE ASSUMED.

Example:

```
*PROCESS MAR(10,72) SEQ(10,18);
```

Explanation: The source margins need not overlap the position of the sequence number. If they do, the following position for the sequence number is assumed:

F-format records: last eight columns

IEL0042I SOURCE MARGINS OVERLAP SEQUENCE FIELD. 'SEQUENCE' AND 'NUMBER' OPTIONS IGNORED.

Example:

```
*PROCESS MAR(2,80) SEQ(1,8);
```

Explanation: The assumed position of the sequence number, as described in the explanation for message IEL0041I, has failed to prevent overlapping of the sequence number by the source margins. The SEQUENCE option is ignored. The NUMBER and GONUMBER options will be replaced by the STMT and GOSTMT options if these are specified.

IEL0043I 'COUNT' OPTION USED WITH 'NOGOSTMT' OR 'NOGONUMBER' OPTION. 'COUNT' OPTION IGNORED.

Example:

```
*PROCESS CT NUM NGN;
```

Explanation: Statement frequency counting is performed by recording the numbers of statements involved in all branches. With the exception of points of interrupt, all statements that might be involved in branches can be recognized at compile-time.

When a statement number table is not available at run time (because NOGOSTMT or NOGONUMBER are in effect) it is impossible to determine the statement number at a point of interrupt. If return is not made to the point of interrupt, the count values will be incorrect.

If NOGOSTMT or NOGONUMBER are not specified explicitly, GOSTMT or GONUMBER (depending on whether STMT or NUMBER have been specified) will be implied by COUNT.

IEL0045I U I/O ERROR ON T.

Example:

```
IEL0045I U I/O ERROR ON SPILL FILE:
END OF FILE CONDITION ENCOUNTERED.
```

Explanation: I/O error occurred on the indicated file. Where possible, a more detailed description of the error is included in T.

IEL0046I INVALID OPTION SUBFIELD SPECIFIED. SUBFIELD IGNORED IN OPTION T.

Example:

```
*PROCESS XREF(LONG);
```

Explanation: In the example, LONG is not a valid suboption for the XREF option.

IEL0047I U COMPILER INITIALIZATION ERROR. COMPILATION TERMINATED.

Explanation: An error has occurred in the compiler initialization phase.

IEL0048I PRINT FILE CANNOT BE OPENED.

Explanation: The attempt to OPEN the print file has failed, possibly because no JCL statement has been provided. Compilation is terminated.

IEL0050I E IDENTIFIER BEGINNING T EXCEEDS N CHARACTERS.

PREPROCESSOR RESTRICTION. IDENTIFIER BEGINNING T IS TOO LONG. TRUNCATED TO FIRST N CHARACTERS.

Example:

```
%INCLUDE DECLARATIONS;
%INCLUDE X(DECLARATIONS);
```

Explanation: The maximum possible length for an identifier in a %INCLUDE statement is 8 characters. Therefore in the above example, the identifier DECLARATIONS is truncated to DECLARAT.

IEL0051I S NESTING LEVEL FOR '%INCLUDE' STATEMENT EXCEEDS N.

MORE THAN N LEVELS OF NESTING FOR '%INCLUDE' STATEMENT. STATEMENT IGNORED. RERUN WITH 'MACRO' OPTION.

Example:

```
      %INCLUDE A;
in A: %INCLUDE B;
in B: %INCLUDE C;
```

(and so on, to a depth greater than 8).

Explanation: %INCLUDE statements cannot be nested with more than eight levels when using the INCLUDE compile-time option.

Programmer Response: The preprocessor, which has no limits on the depth of nesting, should be used by specifying the MACRO compiler option instead of the INCLUDE compile-time option.

IEL0052I S '%INCLUDE' MEMBER T NOT FOUND.

'%INCLUDE' MEMBER T NOT FOUND. MEMBER IGNORED.

Example:

```
%INCLUDE X,Y;
```

Explanation: If member X.P cannot be found in the library search chain, the member is ignored and processing continues with Y.

IEL0053I S I/O ERROR READING MEMBER T.

I/O ERROR READING MEMBER T. PROCESSING OF MEMBER TERMINATED.

Example:

```
%INCLUDE X,Y;
```

Explanation: If an I/O error is encountered while including member X.P, processing of X.P is terminated and an attempt is made to include Y.P.

Programmer Response: Rerun the job. If the error recurs, call IBM for programming support. Before calling IBM, refer to the introduction to this part of the publication for details of information that IBM will need in order to diagnose the problem.

IEL0054I S INVALID TEXT BEGINNING T IGNORED.

INVALID TEXT BEGINNING T IN '%INCLUDE' STATEMENT. STATEMENT IGNORED.

Example:

```
%INCLUDE A*B;
```

Explanation: The syntax of the %INCLUDE statement is incorrect. In the example shown, an identifier is expected.

IEL0056I W INVALID CARRIAGE CONTROL POSITION IGNORED FOR '%INCLUDE' MEMBER D.

CARRIAGE CONTROL POSITION FOR '%INCLUDE' MEMBER D IS WITHIN SOURCE MARGINS OR SEQUENCE FIELD. IT IS IGNORED.

Explanation: The carriage control position specified in the MARGINS options must lie outside the margins and outside any sequence field.

IEL0057I S SEQUENCE AND MARGINS OVERLAP FOR D. T ASSUMED.

SEQUENCE AND MARGINS FIELDS OVERLAP FOR '%INCLUDE' FILE D. T ASSUMED.

Explanation: The MARGINS compile-time option is modified if it overlaps the sequence field.

Programmer Response: If the fix-up is unsatisfactory, either or both of the following compile-time options will need to be modified: MARGINS, SEQUENCE.

IEL0059I S I/O ERROR SEARCHING FOR MEMBER T.

I/O ERROR SEARCHING FOR MEMBER T. MEMBER IGNORED.

Example:

```
%INCLUDE X;
```

Explanation: In the example shown, an I/O error has occurred during an attempt to find member X.P.

Programmer Response: If the error persists, call IBM for programming support.

IEL0065I E 'RETURNS' ATTRIBUTE ON D IGNORED.

'RETURNS' ATTRIBUTE ON BUILTIN FUNCTION D IGNORED.

Example:

```
%DECLARE SUBSTR BUILTIN RETURNS(CHAR);
```

Explanation: Data type returned by a built-in function is determined by the language rules.

IEL0066I E 'ENTRY' ATTRIBUTE ON D IGNORED.

'ENTRY' ATTRIBUTE ON BUILTIN FUNCTION D IGNORED.

Example:

```
%DECLARE INDEX BUILTIN ENTRY;
```

Explanation: The BUILTIN attribute implies the ENTRY attribute.

IEL0067I S D INVALID BUILTIN FUNCTION NAME.

D IS NOT VALID BUILTIN FUNCTION NAME. REFERENCE WILL END PROCESSING.

Example:

```
%DECLARE HARRIET BUILTIN;
```

Explanation: Specify only allowed built-in function names for the preprocessor.

IEL0068I E DESCRIPTOR LIST AFTER 'ENTRY' IGNORED.

PARAMETER DESCRIPTOR LIST ON 'ENTRY' ATTRIBUTE IGNORED.

Example:

```
%DECLARE P ENTRY(CHAR,FIXED);
```

should be

```
%DECLARE P ENTRY;
```

Explanation: The arguments are always converted to the types specified by the PROCEDURE statement.

IEL0069I E 'RETURNS' ATTRIBUTE IGNORED.

'RETURNS' ATTRIBUTE IN 'DECLARE' STATEMENT IGNORED.

Example:

```
%DECLARE P ENTRY RETURNS(FIXED);
```

should be

```
%DECLARE P ENTRY;
```

Explanation: The attribute of the value returned by a compile-time procedure is determined by the procedure statement.

IEL0070I S END OF SOURCE TEXT IN STRING.

END OF SOURCE TEXT IN STRING. QUOTE ASSUMED BEFORE END OF SOURCE TEXT.

Explanation: End of source text found while scanning for a closing quotation mark for a character or graphic string. Check that all quotation marks are paired.

IEL0071I S NO DELIMITER ON REPLACEMENT VALUE STRING.

REPLACEMENT VALUE CONTAINS NO END OF STRING DELIMITER. DELIMITER ASSUMED AT END OF STRING.

Explanation: An end-of-string delimiter has not been found in a replacement value.

IEL0072I E INVALID CHARACTER IN BIT STRING.

INVALID CHARACTER IN BIT STRING. PROCESSED AS CHARACTER STRING.

IEL0073I S END OF SOURCE TEXT IN COMMENT.

END OF SOURCE TEXT IN COMMENT. COMMENT DELIMITER ASSUMED AT END OF SOURCE TEXT.

Explanation: The end of the source text has been encountered while scanning for an end-of-comment delimiter.

IEL0074I E NO COMMENT DELIMITER IN REPLACEMENT VALUE.

REPLACEMENT VALUE CONTAINS NO END OF COMMENT DELIMITER. COMMENT DELIMITER ASSUMED AT END OF REPLACEMENT VALUE.

Explanation: An end-of-comment delimiter cannot be found in a replacement value.

IEL0075I E INVALID CHARACTER REPLACED BY BLANK.

Explanation: An invalid character has been found in the source text.

IEL0077I E CONFLICTING USE OF D.

USE OF D IN PROCEDURE ENDING AT THIS LINE CONFLICTS WITH PREVIOUS USE. REFERENCE WILL END PROCESSING.

Example:

```
%DCL E ENTRY;
%P: PROCEDURE RETURNS(CHAR);
    E = 3;
%END;
```

Explanation: An identifier has been used but not declared in a compiler-time procedure. The use conflicts with a use or declaration outside the procedure.

IEL0078I E '%' IN LABEL LIST IGNORED.

Example:

```
% LABEL4: % IF C1 = C2 etc.
```

Explanation: In the above statement the second "%" is ignored.

IEL0079I E NO LABEL BEFORE COLON.

NO LABEL BEFORE COLON. COLON IGNORED.

Example:

```
%: A = B;
```

Programmer Response: Insert label or remove colon.

IEL0080I S INVALID TEXT IGNORED FROM T TO SEMICOLON.

Example:

```
% GOTOLABEL 2;  should be  % GOTO LABEL2;
```

IEL0081I E CONFLICTING USE OF D.

CONFLICTING USE OF IDENTIFIER D AS LABEL. REFERENCE WILL END PROCESSING.

Example:

```
%DCL (A,B,C) CHAR;
%A: B = C;
```

Explanation: No system action is taken unless a statement which references the identifier is detected.

IEL0082I E MULTIPLE USE OF D AS LABEL.

D USED AS LABEL MORE THAN ONCE. REFERENCE WILL END PROCESSING.

Example:

```
%L;A = 1;
%L;A = 2;
```

Explanation: No system action is taken unless a statement which references the multiply-defined label is detected.

IEL0083I W LABELS ON DECLARE STATEMENT.

LABELS ON 'DECLARE' STATEMENT IGNORED.

Example:

```
% LABEL1: DECLARE J FIXED;
```

IEL0084I S CONFLICTING USE OF D.

USE OF D CONFLICTS WITH PREVIOUS USE AS LABEL.

Example:

```
%L:;
%L = 2;
```

IEL0085I E NO ATTRIBUTE DECLARED FOR D.

NO ATTRIBUTE DECLARED FOR PARAMETER D IN PROCEDURE ENDING AT THIS LINE. CHARACTER ASSUMED.

Example:

```
%PROC1: PROC (P1,P2,P3) RETURNS (CHAR);
DCL (P1, P2) FIXED;
%END PROC1;
```

IEL0086I E LABEL D IS UNDEFINED.

LABEL D IS UNDEFINED. REFERENCE WILL END PROCESSING.

Explanation: No system action is taken unless a %GOTO statement that references the undefined label is executed. Check all references to the label, or define it.

IEL0087I E END OF SOURCE TEXT IN PROGRAM.

END OF SOURCE TEXT BEFORE LOGICAL END OF PROGRAM. '%END' STATEMENT ASSUMED.

Explanation: Check that each %PROCEDURE and %DO statement is matched with a %END statement.

IEL0088I E D IS UNDEFINED IN PROCEDURE.

LABEL D IS UNDEFINED IN PROCEDURE ENDING AT THIS LINE. REFERENCE WILL END PROCESSING.

Explanation: A label must be defined within the procedure as transfers out of procedures are not allowed.

IEL0089I E SEMICOLON AFTER 'IF' EXPRESSION.

SEMICOLON TERMINATES 'IF' EXPRESSION. SEMICOLON IGNORED.

Example:

```
%IF P1 = P2;
%THEN C1 = C2;
```

IEL0090I S 'IF' STATEMENT IGNORED.

'IF' EXPRESSION NOT FOLLOWED BY '%' OR 'THEN'. 'IF' STATEMENT IGNORED.

Example:

```
%IF C1 = C2 GOTO L1;
```

IEL0091I E NO '%' BEFORE 'THEN'.

MISSING '%' ASSUMED BEFORE 'THEN' IN '%IF' STATEMENT.

Example:

```
% IF C1 = C2 THEN;
```

IEL0092I E NO 'THEN' AFTER '%'.

MISSING 'THEN' ASSUMED AFTER '%' IN '%IF' STATEMENT.

Example:

```
%IF C1 = C2
%C2 = C3;
```

IEL0093I E INVALID STATEMENT AFTER '%THEN' or '%ELSE'.

STATEMENT AFTER '%THEN' OR '%ELSE' NOT A PREPROCESSOR STATEMENT. '%' ASSUMED BEFORE IT.

Example:

```
% IF C1 = C2 % THEN C1 = C3;
```

is incorrect.

Explanation: If the statement in question is not a preprocessor statement, it should be inside a preprocessor do-group.

IEL0094I E MISSING 'THEN' ASSUMED.

MISSING 'THEN' ASSUMED IN 'IF' STATEMENT.

Example:

```
%P: PROC RETURNS (FIXED);
    IF I = 1 GOTO L;

⋮
%END;
```

IEL0095I E INVALID '%' IGNORED.

INVALID '%' IN PREPROCESSOR PROCEDURE IGNORED.

Example:

```
% PROC1: PROCEDURE RETURNS(CHARACTER);
% DCL A FIXED;
%END;
```

Explanation: Statements within preprocessor procedures cannot be preceded by "%."

IEL0096I W LABELS ON 'ELSE' IGNORED.

Example:

```
%IF A = B %THEN %;
% LABEL3: ELSE %;
```

IEL0097I E NULL STATEMENT ASSUMED.

NO STATEMENT AFTER 'THEN' OR 'ELSE'. NULL STATEMENT ASSUMED.

Example:

```
% IF I = 1 % THEN % ELSE%;
```

IEL0098I E NO 'IF' BEFORE 'ELSE'

NO 'IF' BEFORE 'ELSE'. 'ELSE' IGNORED.

Explanation: An ELSE clause has been found which is not part of an IF statement.

IEL0100I E DUMMY LABEL ASSUMED ON STATEMENT.

NO LABEL ON '%PROCEDURE' STATEMENT. DUMMY LABEL ASSUMED.

Example:

```
%PROC RETURNS(CHAR);
```

Explanation: A %PROCEDURE statement should have a label.

Programmer Response: Insert a label on the PROCEDURE statement.

IEL0101I U MORE THAN N PROCEDURES.

PREPROCESSOR RESTRICTION. MORE THAN N PREPROCESSOR PROCEDURES DEFINED IN A COMPILATION. PROCESSING TERMINATED.

Programmer Response: Reduce the number of preprocessor procedures to within the limit given by N.

IEL0102I E D PREVIOUSLY DEFINED.

ENTRY NAME D PREVIOUSLY DEFINED. REFERENCE WILL END PROCESSING.

Example:

```
%E: PROC RETURNS(CHAR)
   :
%E: PROC RETURNS(CHAR);
```

Explanation: No action is taken unless the multiply-defined label is referenced by a statement that is executed.

Programmer Response: Change the label on one of the %PROCEDURE statements, or remove one of the procedures.

IEL0103I E INVALID USE OF D.

INVALID USE OF FUNCTION D ON LEFT OF EQUALS SYMBOL. REFERENCE WILL END PROCESSING.

Example:

```
%DCL E ENTRY RETURNS(CHAR);
%E = 'ABC';
```

Explanation: Entry names and built-in function names cannot appear on the left-hand side of an assignment statement. Execution of such a statement will terminate processing.

IEL0104I E CONFLICTING USE OF D.

CONFLICTING USE OF IDENTIFIER D AS ENTRY NAME. REFERENCE WILL END PROCESSING.

Example:

```
%DCL C CHAR;
%C = C(I);
```

Explanation: An identifier followed by a parenthesis in a preprocessor expression is considered to be an entry name. Execution of such a statement will terminate processing.

IEL0105I E MULTIPLE USE OF D IN PARAMETER LIST.

PARAMETER D APPEARS MORE THAN ONCE IN PARAMETER LIST. AN ARGUMENT CORRESPONDING TO SECOND USE OF PARAMETER WILL NOT BE USED WITHIN PROCEDURE.

Example:

```
%E: PROC(P,P) RETURNS(CHAR);
```

Explanation: The number of parameters to the procedure is not changed, but, within the procedure, references to the multiply-defined parameter will apply to its first use.

IEL0106I S MORE THAN N PARAMETERS USED.

PREPROCESSOR RESTRICTION. MORE THAN N PARAMETERS USED. REFERENCE WILL END PROCESSING.

Explanation: Processing is ended if a procedure having more than fifteen parameters is referenced by a statement that is executed.

IEL0107I E MISSING PARAMETER.

MISSING PARAMETER. A CORRESPONDING ARGUMENT WILL NOT BE USED WITHIN PROCEDURE.

Example:

```
%PROCL: PROCEDURE (P1,P2,,P4)
RETURNS(CHAR);
```

Explanation: The assumption is made that the omission of the parameter is intentional.

IEL0108I E PARAMETER T INVALID.

PARAMETER T INVALID. AN ARGUMENT CORRESPONDING TO THE PARAMETER WILL NOT BE USED WITHIN THE PROCEDURE.

Example:

```
%P: PROC(8) RETURNS(CHAR);
```

Explanation: The expected parameter is not an identifier. The parameter is assumed to exist but is not identified within the procedure.

IEL0109I S T TO NEXT COMMA OR SEMICOLON IGNORED.

INVALID BLANK OR MISSING COMMA IN PARAMETER LIST. TEXT IGNORED FROM T TO NEXT COMMA OR SEMICOLON.

Example:

```
%PROC1: PROC (P1,P2,P3 P4)
RETURNS(CHAR);
```

IEL0110I S RIGHT PARENTHESIS ASSUMED FOR SEMICOLON.

SEMICOLON FOUND IN PARAMETER LIST. RIGHT PARENTHESIS ASSUMED.

Example:

```
%E: PROC (P ;
```

Explanation: A semicolon has been encountered during the scan of an apparent parameter list. A right parenthesis has been inserted before the semicolon.

IEL0111I E INVALID RETURNED VALUE T REPLACED BY 'CHARACTER'.

RETURNED VALUE NOT 'FIXED' OR 'CHARACTER'. T REPLACED BY 'CHARACTER'.

Example:

```
%E: PROC RETURNS(BIT);
```

Explanation: Returned values can only be FIXED or CHARACTER. CHARACTER is the assumed attribute.

IEL0112I E 'RETURNS(CHAR)' ASSUMED FOR RETURNED VALUE.

NO ATTRIBUTE FOR RETURNED VALUE. 'RETURNS(CHAR)' ASSUMED.

Example:

```
%P: PROC;
```

IEL0113I W INVALID CONTINUATION OF GRAPHIC STRING.

INVALID CONTINUATION OF GRAPHIC STRING. LAST COLUMN ON LINE IGNORED.

Explanation: This message is issued by the preprocessor. Since each graphic requires 2 bytes, you must be sure that the graphic string follows the continuation rules described in the *PL/I VSE Language Reference.*

IEL0114I E T IS IGNORED.

T IS IGNORED IN '%DEACTIVATE' STATEMENT.

Example:

```
%DEACTIVATE A NORESCAN;
```

Explanation: RESCAN and NORESCAN options are only valid in a %ACTIVATE statement.

IEL0115I W CHARACTER ASSUMED FOR UNDECLARED D.

REFERENCE TO UNDECLARED IDENTIFIER D. CHARACTER ATTRIBUTE ASSUMED.

Example:

```
%DCL (A,B) CHAR C FIXED;
%D = A||B;
```

Explanation: D is given the attribute CHAR by default.

IEL0116I S '%PROCEDURE' STATEMENT INVALID.

'%PROCEDURE' STATEMENT IN PREPROCESSOR PROCEDURE. TEXT IGNORED TO NEXT PREPROCESSOR '%END' STATEMENT.

Example:

```
%PROC: PROC;
PROC6: PROC;
END PROC6;
 %END;
```

Explanation: Procedures cannot be nested in preprocessor procedures. Other messages might be generated by this error.

IEL0117I S '%PROCEDURE' STATEMENT REPLACED BY NULL.

'%PROCEDURE' STATEMENT IN '%THEN' OR '%ELSE' CLAUSE REPLACED BY NULL STATEMENT. TEXT IGNORED TO NEXT PREPROCESSOR '%END' STATEMENT.

Example:

```
%IF C1 = C2 %THEN %PROC2: PROCEDURE;
END PROC2;
%ELSE %PROC3: PROCEDURE; etc.
```

Explanation: %PROCEDURE statements are not allowed in preprocessor THEN or ELSE clauses. Other messages might be generated by this error.

IEL0118I S '%RETURN' STATEMENT INVALID. IGNORED.

'%RETURN' STATEMENT INVALID OUTSIDE PREPROCESSOR PROCEDURE. STATEMENT IGNORED.

Example:

```
%RETURN(0);
```

IEL0119I E MISSING PARENTHESIS ASSUMED.

MISSING PARENTHESIS ASSUMED FOR 'RETURN' EXPRESSION.

Example:

```
%P: PROC FIXED;
RETURN 6);
%END;
```

IEL0120I E INVALID TEXT. T TO SEMICOLON IGNORED.

INVALID TEXT AFTER EXPRESSION IN 'RETURN' STATEMENT. TEXT IGNORED FROM T TO SEMICOLON.

Example:

```
%P:  PROC RETURNS(CHAR);
RETURN ('1') IF A = B;
%END;
```

Explanation: The RETURN statement has been processed but scan finds text when it expects a semicolon.

IEL0121I S 'GOTO' STATEMENT IGNORED.

NO OPERAND IN 'GOTO'. STATEMENT IGNORED.

Example:

```
%GOTO;
```

IEL0122I E CONFLICTING USE OF D.

USE OF IDENTIFIER D IN '%GOTO' STATEMENT CONFLICTS WITH PREVIOUS USE. REFERENCE WILL END PROCESSING.

Example:

```
%P = PROC RETURNS(FIXED);
  ⋮
%GOTO P;
```

IEL0123I S SEMICOLON MISSING. T TO NEXT SEMICOLON IGNORED.

SEMICOLON MISSING AFTER '%GOTO' STATEMENT. TEXT IGNORED FROM T TO NEXT SEMICOLON.

Example:

```
%GOTO LABEL4 C1 = C2;
```

IEL0124I U '%GOTO' IS AN INVALID BRANCH.

OPERAND OF '%GOTO' IS LABEL IN ITERATIVE 'DO' OR INCLUDED TEXT. PROCESSING TERMINATED.

Example:

```
% GOTO L1;
  ⋮
% DO I 1 TO N;
%L1:
%END;
```

IEL0125I S STATEMENT INVALID IN PROCEDURE.

INVALID '%ACTIVATE' OR '%DEACTIVATE' IN PREPROCESSOR PROCEDURE. STATEMENT IGNORED.

Explanation: ACTIVATE and DEACTIVATE statements cannot be used in preprocessor procedures.

IEL0126I E STATEMENT HAS NO OPERAND. IGNORED.

'%ACTIVATE' OR '%DEACTIVATE' HAS NO OPERAND. STATEMENT IGNORED.

Example:

```
%ACTIVATE;
```

IEL0127I E REDUNDANT COMMA IGNORED.

MISSING OPERAND OR REDUNDANT COMMA IN '%ACTIVATE' OR '%DEACTIVATE'. COMMA IGNORED.

Example:

```
%DEACTIVATE C5,, C6;
```

IEL0128I S INVALID FIELD T IGNORED.

INVALID FIELD T IN '%ACTIVATE' OR '%DEACTIVATE ' STATEMENT IS IGNORED.

Example:

```
%ACTIVATE 7TIMES;
```

IEL0129I S IDENTIFIER D IGNORED.

IDENTIFIER D NOT PROCEDURE OR VARIABLE. IT HAS BEEN IGNORED IN '%ACTIVATE' OR '%DEACTIVATE' STATEMENT.

Example:

```
%LABEL4: ;
% DEACTIVATE LABEL4;
```

(where LABEL4 is a statement label).

IEL0130I S T TO COMMA OR SEMICOLON IGNORED.

INVALID BLANK OR MISSING COMMA IN '%ACTIVATE' OR '%DEACTIVATE' STATEMENT. TEXT IGNORED FROM T TO COMMA OR SEMICOLON.

Example:

```
%DEACTIVATE C5, C6 C7;
%DEACTIVATE IDENTIFIER;
```

IEL0131I S NON-ITERATIVE 'DO' ASSUMED.

INVALID SYNTAX IN 'DO' STATEMENT. NON-ITERATIVE 'DO' ASSUMED.

Example:

```
%DO A: = 1 TO 10;
```

IEL0132I W NO MAXIMUM VALUE FOR 'DO' ITERATION.

NO MAXIMUM VALUE SPECIFIED FOR 'DO' ITERATION.

Example:

```
%DO I = 1 BY 1;
 %EXIT;
%END;
```

Explanation: This warning is given because the program might loop.

Programmer Response: If the program loops, provide an iteration limit or an alternative exit.

IEL0133I E SEMICOLON ASSUMED BEFORE '%'.

MISSING SEMICOLON ASSUMED BEFORE '%'.

Explanation: A percent found in the text has been assumed to signify the start of a new statement.

IEL0134I E SECOND 'TO' REPLACED BY 'BY'.

SECOND 'TO' FOUND IN ITERATION SPECIFICATION OF 'DO' STATEMENT. REPLACED BY 'BY'.

Example:

```
%DO I = 1 TO 10 TO 1;
%DO I = 1 TO 10 TO 1 BY 1;
```

(BY will have been ignored when this message occurs.)

IEL0135I E SECOND 'BY' REPLACED BY 'TO'.

SECOND 'BY' FOUND IN ITERATION SPECIFICATION OF 'DO' STATEMENT. REPLACED BY 'TO'.

Example:

```
%DO I = 1 BY 1 BY 10;
```

IEL0136I E SEMICOLON MISSING. T TO NEXT SEMICOLON IGNORED.

MISSING SEMICOLON IN 'DO' STATEMENT. TEXT FROM T TO NEXT SEMICOLON IGNORED.

Example:

```
%DO I = 1 TO 10 BY 1 BY 7;
                     ↑
                     T
```

IEL0137I E NULL STATEMENT ASSUMED BEFORE 'END'.

'END' STATEMENT MAY NOT FOLLOW 'THEN' OR 'ELSE'. NULL STATEMENT ASSUMED BEFORE 'END'.

Example:

```
%DO; %IF C1 = C2 %THEN %END;
```

IEL0138I E SEMICOLON MISSING. T TO NEXT SEMICOLON IGNORED.

MISSING SEMICOLON IN 'END' STATEMENT. TEXT FROM T TO NEXT SEMICOLON IGNORED.

Explanation: A %END statement must be followed by a semicolon or by a label and a semicolon.

IEL0139I E REDUNDANT '%END' STATEMENT IGNORED.

Explanation: A %END statement is not preceded by a %DO or %PROCEDURE statement that has not already been terminated.

IEL0140I E REFERENCE TO UNKNOWN LABEL IGNORED.

LABEL REFERENCED IN '%END' STATEMENT NOT FOUND. REFERENCE IGNORED.

Explanation: The operand of the %END statement cannot be matched with the label on a %PROCEDURE or %DO statement which does not already have a matching %END statement.

IEL0141I E '%' ASSUMED BEFORE 'END' STATEMENT.

'%' ASSUMED BEFORE 'END' STATEMENT OF PROCEDURE.

Explanation: The END statement is the logical end of the procedure and should be preceded by "%."

IEL0142I E T NOT A LABEL. IGNORED.

IDENTIFIER T ON '%END' STATEMENT NOT A LABEL. IDENTIFIER IGNORED.

Example:

```
%X = Y + A;
  ⋮
%END A;
```

IEL0143I E NO 'RETURN' STATEMENT IN PROCEDURE.

NO 'RETURN' STATEMENT IN PROCEDURE T. NULL VALUE WILL BE RETURNED.

Explanation: The PL/I language requires the use of a RETURN statement in a preprocessor procedure; a null value is returned if a procedure without a RETURN statement is invoked.

IEL0144I S '%INCLUDE' INVALID IN PROCEDURE.

'%INCLUDE' STATEMENT INVALID IN PREPROCESSOR PROCEDURE. STATEMENT IGNORED.

Example:

```
%!PROC1: PROCEDURE (P1, P2) RETURNS(CHAR);
INCLUDE RUBBISH;
%END;
```

IEL0146I S INVALID FIELD. TEXT IGNORED FROM T.

INVALID FIELD IN '%INCLUDE' STATEMENT. TEXT IGNORED FROM T TO NEXT COMMA OR SEMICOLON.

Example:

```
%INCLUDE 7RECORDS;
```

IEL0147I S STATEMENT HAS NO OPERAND. IGNORED.

'%INCLUDE' STATEMENT HAS NO OPERAND. STATEMENT IGNORED.

Example:

```
%INCLUDE;
```

IEL0148I E MEMBER NAME TRUNCATED TO N CHARACTERS.

PREPROCESSOR RESTRICTION. MEMBER NAME TRUNCATED TO FIRST N CHARACTERS.

Explanation: Only the first 8 characters of a data-set name are used.

IEL0149I E RIGHT PARENTHESIS ASSUMED.

MISSING RIGHT PARENTHESIS ASSUMED AFTER MEMBER NAME.

IEL0151I S 'DECLARE' STATEMENT IGNORED.

'DECLARE' STATEMENT INVALID AFTER 'THEN' OR 'ELSE'. STATEMENT IGNORED.

Example:

```
%IF C1 = C2
%THEN %DCL C1 FIXED;
```

Explanation: A DECLARE statement can only appear after THEN or ELSE when inside a DO group.

IEL0152I E STATEMENT HAS NO OPERAND. IGNORED.

'%DECLARE' STATEMENT HAS NO OPERAND. STATEMENT IGNORED.

Example:

```
%DECLARE;
```

IEL0153I S MAXIMUM FACTORING LEVEL IS N.

PREPROCESSOR RESTRICTION. N LEVELS MAXIMUM FOR FACTORING IN 'DECLARE' STATEMENT. TEXT TO NEXT SEMICOLON IGNORED.

Explanation: A DECLARE statement with too many levels of factoring has been detected.

Programmer Response: Subdivide the DECLARE statement into two or more separate statements so that the level of factoring becomes acceptable.

IEL0154I E REDUNDANT COMMA IGNORED.

MISSING OPERAND OR REDUNDANT COMMA IN 'DECLARE' STATEMENT. COMMA IGNORED.

Example:

```
%DCL (C1, C2,, C3) CHAR;
```

IEL0155I E DUMMY IDENTIFIER ASSUMED.

IDENTIFIER MISSING WHERE EXPECTED. DUMMY ASSUMED.

Example:

```
DCL () CHAR;
```

Explanation: The preprocessor expected to find an identifier but found a delimiter.

IEL0156I E MULTIPLE DECLARATION OF D.

MULTIPLE DECLARATION OF IDENTIFIER D. REFERENCE WILL END PROCESSING.

Example:

```
%DCL C CHAR;
%DCL C CHAR;
```

Explanation: An identifier can be declared only once. No action is taken unless the multiply-declared identifier is referenced.

IEL0157I S INVALID SYNTAX. TEXT IGNORED FROM T.

INVALID SYNTAX IN 'DECLARE' STATEMENT. TEXT IGNORED FROM T TO NEXT SEMICOLON.

Example:

```
%DCL 7 FIXED;
```

IEL0158I E LABEL D CANNOT BE DECLARED.

LABEL D CANNOT BE DECLARED. REFERENCE WILL END PROCESSING.

Example:

```
%L: ;
%DECLARE L FIXED;
```

IEL0159I E REDUNDANT RIGHT PARENTHESIS IGNORED.

Example:

```
%DCL (B1, E2)) FIXED;
```

IEL0160I E T IGNORED.

INVALID ATTRIBUTE T IGNORED.

Example:

```
%DECLARE B BIT CHAR;
```

Explanation: The position in which an attribute is expected contains something other than FIXED, CHARACTER, BUILTIN, ENTRY, or RETURNS.

IEL0161I E RIGHT PARENTHESIS ASSUMED.

MISSING RIGHT PARENTHESIS ASSUMED.

Example:

```
DCL (C1, C2 CHAR;
```

IEL0162I E 'RETURNS' BUT NO 'ENTRY' ATTRIBUTE FOR D.

'RETURNS' BUT NO 'ENTRY' ATTRIBUTE FOR PROCEDURE D IN 'DECLARE' STATEMENT BEGINNING AT OR BEFORE THIS LINE.

Example:

```
%DCL PROC2 RETURNS(FIXED);
```

Explanation: The identifier is treated as an entry name. The effect of this statement is to activate the entry name. This error will also cause message number IEL0069I to be printed.

IEL0163I W ATTRIBUTE T ASSUMED FOR D.

NO ATTRIBUTES DECLARED FOR IDENTIFIER D. 'CHARACTER' ASSUMED.

Example:

```
%DCL A1, A2 CHAR;
```

Explanation: The attribute CHAR is assumed for an identifier declared without attributes, unless the identifier is given previously as a label on the PROCEDURE statement, in which case ENTRY is assumed.

IEL0164I I REPLACING 'MACRO' BY 'INCLUDE' WILL REDUCE COMPILE TIME.

COMPILE TIME FOR THIS PROGRAM WILL BE REDUCED IF THE 'INCLUDE' COMPILER OPTION IS SPECIFIED INSTEAD OF 'MACRO'.

Explanation: You have specified the MACRO compile-time option. However, since all the preprocessor statements in your source program are %INCLUDE statements, compilation will be faster if you specify the INCLUDE compile-time option instead.

IEL0165I S '%GOTO' D IS AN INVALID BRANCH.

'%GOTO' D IS AN INVALID BRANCH INTO INCLUDED TEXT. EXECUTION WILL END PROCESSING.

Explanation: A source statement module included in the text by a %INCLUDE statement contains a %GOTO statement that refers to a label contained in a source statement module included in the text by a further nested %INCLUDE statement.

IEL0168I E LABEL IGNORED.

LABEL INVALID ON LISTING CONTROL STATEMENT. LABEL IGNORED.

Example:

```
%L: PAGE;
```

Explanation: A listing control statement should not be prefixed by a label.

IEL0169I E CONFLICTING ATTRIBUTE T FOR D IGNORED.

CONFLICTING ATTRIBUTES FOR IDENTIFIER D. ATTRIBUTE T IGNORED.

Example:

```
%DCL P CHAR RETURNS(CHAR);
```

IEL0170I E CONFLICTING DECLARATION OF D.

DECLARATION OF IDENTIFIER D CONFLICTS WITH PREVIOUS USE. REFERENCE WILL END PROCESSING.

Example:

```
%E: PROC RETURNS(CHAR);
%END;
%DCL E CHAR;
```

IEL0171I E ZERO OPERAND ASSUMED.

MISSING OPERAND. FIXED DECIMAL ZERO ASSUMED.

Example:

```
%A = A +  ;
```

IEL0172I S T REPLACED BY PLUS.

INVALID OPERAND T REPLACED BY PLUS.

Example:

```
%A = A**2;
```

Explanation: Operators "**" and "->" are not allowed in preprocessor statements.

IEL0173I W BLANK ASSUMED AFTER T.

BLANK ASSUMED BETWEEN CONSTANT T AND FOLLOWING LETTER.

IEL0174I E 'NOT' REPLACED BY 'NE'.

OPERATOR 'NOT' USED AS INFIX OPERATOR. REPLACED BY 'NE'.

IEL0175I W TEXT FOLLOWING '%PAGE' IGNORED TO NEXT SEMICOLON.

PREPROCESSOR RESTRICTION. TEXT FOLLOWING '%PAGE' IGNORED TO NEXT SEMICOLON.

Example:

```
%PAGE ('NEW TITLE', 200);
```

Explanation: The preprocessor does not implement the TITLE or page numbering option of the %PAGE listing control statement.

IEL0176I E CONFLICTING USE OF D.

USE OF IDENTIFIER D IN EXPRESSION CONFLICTS WITH PREVIOUS USE. REFERENCE WILL END PROCESSING.

Example:

```
%LAB;A = LAB + 2;
```

IEL0177I W TEXT ON SAME LINE AS LISTING CONTROL STATEMENT.

PREPROCESSOR RESTRICTION. TEXT ON SAME LINE AS LISTING CONTROL STATEMENT. STATEMENT NOT IMPLEMENTED.

Example:

```
A = B;
%PAGE; A = B;
```

Explanation: A listing control statement is not implemented by the preprocessor if any other text appears on the same line.

IEL0178I S PLUS ASSUMED AS OPERATOR.

MISSING OPERATOR. PLUS ASSUMED.

Example:

```
%C = A  B;
```

IEL0179I S ZERO EXPRESSION ASSUMED.

EXPRESSION MISSING. FIXED DECIMAL ZERO ASSUMED.

Example:

```
%CL = ;
```

IEL0180I S T REPLACED BY ZERO.

INVALID OPERAND T REPLACED BY FIXED DECIMAL ZERO.

Example:

```
%A = B + 1C;
```

IEL0181I E LEFT PARENTHESIS ASSUMED.

MISSING LEFT PARENTHESIS ASSUMED AT BEGINNING OF EXPRESSION.

Example:

```
%F1 = F2 + F3);
```

IEL0182I U REFERENCE TERMINATED PROCESSING. REFERENCE TO STATEMENT OR IDENTIFIER WHICH IS IN ERROR. PROCESSING TERMINATED.

Explanation: The preprocessor tried to execute a statement or use an identifier which is in error.

Programmer Response: Check the other messages for the error, and correct the program.

IEL0183I W EXCESS ARGUMENTS TO D IGNORED.

TOO MANY ARGUMENTS TO FUNCTION D. EXCESS ARGUMENTS IGNORED.

Example:

```
%E: PROCEDURE(P,Q) RETURNS(FIXED);
%DECLARE (P,Q) FIXED;
%END;
%C = E(A,B,C);
```

Explanation: There are too many arguments in the procedure reference.

IEL0184I W TOO FEW ARGUMENTS TO D.

TOO FEW ARGUMENTS TO FUNCTION D. NULL STRINGS PASSED AS MISSING ARGUMENTS.

Example:

```
%E: PROCEDURE(P,Q) RETURNS(FIXED);
DECLARE (P,Q) FIXED;
%END;
%C = E(A);
```

Explanation: There are too few arguments in the procedure reference. For a fixed argument the null string will be converted to fixed zero.

IEL0186I U PROCEDURE D NOT FOUND.

REFERENCED PROCEDURE D NOT FOUND. PROCESSING TERMINATED.

Explanation: An entry declaration statement has been found for a procedure which is not present in the text.

IEL0187I U RECURSIVE USE OF D INVALID.

RECURSIVE USE OF PROCEDURE D INVALID. PROCESSING TERMINATED.

Example:

```
%P: PROCEDURE RETURNS(CHAR);
  RETURN(P + 7);
%END;
%C = P;
```

IEL0188I E NULL STRING RETURNED FOR 'SUBSTR'.

TOO FEW ARGUMENTS SPECIFIED FOR BUILTIN FUNCTION 'SUBSTR'. NULL STRING RETURNED.

Example:

```
%S = SUBSTR(A);
```

IEL0189I E EXCESS ARGUMENTS TO T IGNORED.

TOO MANY ARGUMENTS SPECIFIED FOR BUILTIN FUNCTION T. EXCESS ARGUMENTS IGNORED.

Example:

```
%S = SUBSTR(A,B,C,D);
```

IEL0190I E RESULT TRUNCATED TO 5 DIGITS.

FIXED OVERFLOW. RESULT TRUNCATED TO RIGHTMOST 5 DIGITS.

Example:

```
%A = 99999;
%A = A + 3;
```

IEL0191I E ZERO DIVIDE. RESULT SET TO ONE.

Example:

```
%A = 0;
%B = B / A;
```

IEL0192I S END OF SOURCE TEXT IN STATEMENT.

END OF SOURCE TEXT IN STATEMENT. STATEMENT EXECUTION WILL END PROCESSING.

IEL0193I E IDENTIFIER BEGINNING T TRUNCATED.

PREPROCESSOR RESTRICTION. IDENTIFIER BEGINNING T IS TOO

LONG. TRUNCATED TO FIRST 31 CHARACTERS.

Explanation: You can not have identifiers that require more than 31 bytes of storage.

IEL0194I S T HAS PRECISION GREATER THAN N.

PREPROCESSOR RESTRICTION. CONSTANT T HAS PRECISION GREATER THAN N. FIXED DECIMAL ZERO ASSUMED.

Example:

```
%A = 123456;
```

Explanation: The precision of fixed decimal numbers is limited to n digits. The value of zero is assumed for the constant.

IEL0195I E QUESTION MARK IGNORED.

Explanation: Question mark has no syntactical meaning.

IEL0196I S PRECISION OF CONVERTED BIT STRING GREATER THAN N.

PREPROCESSOR RESTRICTION. BIT STRING CONVERTS TO FIXED DECIMAL NUMBER WITH PRECISION GREATER THAN N. RESULT TRUNCATED ON THE LEFT.

Example:

```
%DECLARE C CHARACTER, F FIXED;
%C = '100000000000000000'B;
%F = (C&C);
```

Explanation: If the bit string has more than 32 bits, the last 32 bits are taken for the conversion.

IEL0197I S STRING INVALID FOR CONVERSION.

STRING CONTAINS CHARACTER NOT '1' OR '0' AND CANNOT BE CONVERTED TO BIT STRING. '0' ASSUMED FOR INVALID CHARACTERS.

Example:

```
%C = 'A';
%C = (C&(A ¬= B));
```

IEL0198I S STRING INVALID FOR CONVERSION.

STRING CANNOT BE CONVERTED TO FIXED DECIMAL. FIXED ZERO RESULT ASSUMED.

Example:

```
%C = '1B'
%A = 2 + C;
```

IEL0199I E '%' MISSING ON LISTING CONTROL STATEMENT.

'%' MISSING ON LISTING CONTROL STATEMENT IN COMPILE-TIME PROCEDURE.

Example:

```
%P: PROC RETURNS(CHAR);
    SKIP(2);
%END;
```

Explanation: A listing control statement, even when in a preprocessor procedure, must be preceded by "%."

IEL0200I U REFERENCE TO D TERMINATED PROCESSING.

IDENTIFIER D WITH CONFLICTING USE OR MULTIPLE DEFINITIONS REFERENCED. PROCESSING TERMINATED.

Explanation: An attempt has been made to execute a statement referencing an improperly defined identifier.

Programmer Response: Check the other messages for the error, and correct the program.

IEL0201I W D IS UNINITIALIZED.

UNINITIALIZED VARIABLE T USED. NULL STRING OR ZERO VALUE GIVEN.

Example:

```
%DECLARE A FIXED;
%B = A;
```

Explanation: Variables should be assigned values before being used.

IEL0210I U LEVEL OF NESTING OR REPLACEMENT TOO LARGE.

PREPROCESSOR RESTRICTION. LEVEL OF NESTING OR REPLACEMENT GREATER THAN MAXIMUM. PROCESSING TERMINATED.

Explanation: The level of nesting is calculated by summing the number of current unbalanced left parentheses, the number of current nested DOs, the number of nested IFs, and the number of current nested replacements. A level of 25 is always acceptable.

IEL0212I S INPUT RECORD LENGTH LESS THAN LEFT MARGIN.

LENGTH OF INPUT RECORD LESS THAN LEFT MARGIN OF MARGINS OPTION. RECORD IGNORED.

Explanation: The length of an input record is less than the left margin of the MARGINS specification.

Programmer Response: Check the use of the MARGINS compile-time option; check that a short record is intended.

IEL0213I E 'RETURNS(FIXED)' ASSUMED.

DATA ATTRIBUTE IN '%PROCEDURE' STATEMENT IS NOT PARENTHESIZED AND IS NOT PRECEDED BY 'RETURNS'. 'RETURNS(FIXED)' ASSUMED.

Example:

```
%P: PROC FIXED;
```

IEL0214I E 'RETURNS(CHAR)' ASSUMED.

DATA ATTRIBUTE IN '%PROCEDURE' STATEMENT IS NOT PARENTHESIZED AND IS NOT PRECEDED BY 'RETURNS'. 'RETURNS(CHAR)' ASSUMED.

Example:

```
%P: PROC CHAR;
```

IEL0215I E MISSING PARENTHESIS IN D ARGUMENT LIST.

RIGHT PARENTHESIS ASSUMED AT END OF ARGUMENT LIST FOR PROCEDURE D.

IEL0216I U INVALID STATEMENT IN D ARGUMENT LIST.

ARGUMENT LIST FOR PROCEDURE D CONTAINS A PREPROCESSOR STATEMENT. PROCESSING TERMINATED.

Example:

```
%DCL P ENTRY, X FIXED;
P(%X) = 1;
%P:  PROC(A) RETURNS(CHAR);
%END;
```

Explanation: Preprocessor statements cannot be embedded in the argument list of a preprocessor function reference appearing in non-preprocessor text.

IEL0217I W ARGUMENT LIST FOR D MISSING.

ARGUMENT LIST FOR D IS MISSING. PROCEDURE INVOKED WITHOUT ARGUMENTS.

Example:

```
%DCL VAL CHAR;
%VAL = 'BA';
%BA: PROC(A,B) RETURNS(CHAR);
DCL (A,B) CHAR;
RETURN ('Z'||A||B);
%END BA;
%ACT BA;
VAL (C,D);
```

Explanation: When the active identifier VAL is encountered, it is replaced by its current value BA. Since the RESCAN option applies (by default), the replacement value BA is rescanned for possible further replacement. Since this value is an active reference to a procedure with arguments, but no argument list is present in the value being currently scanned, this message is issued. The procedure BA is invoked without arguments, and the returned value Z is inserted into preprocessed text after further rescanning (and replacement if appropriate).

Note: It is not possible for the argument list (C,D) to be associated with the replacement value BA because of the rules for rescanning and replacement. For details of these rules, see the *PL/I VSE Language Reference.*

IEL0218I E D USED FOR REPLACEMENT.

IDENTIFIER HAS MORE THAN N CHARACTERS. REPLACEMENT DONE ON TRUNCATED FORM D.

Explanation: An identifier activated for replacement by the preprocessor has more than the allowed number of characters. Consequently, any replacement will be performed on the given truncated form.

Programmer Response: Modify the program so that the identifier is reduced to an acceptable length or check that the replacement of the truncated form given does not result in further errors.

IEL0219I E THIRD ARGUMENT OF 'SUBSTR' NEGATIVE.

THIRD ARGUMENT OF BUILTIN FUNCTION 'SUBSTR' NEGATIVE. NULL STRING RETURNED.

IEL0220I E THIRD ARGUMENT OF 'SUBSTR' TOO LARGE.

THIRD ARGUMENT OF BUILTIN FUNCTION 'SUBSTR' GREATER THAN STRING LENGTH. RETURNED VALUE TRUNCATED AT END OF SOURCE STRING.

IEL0221I E ARGUMENTS OF 'SUBSTR' TOO LARGE.

THE SUM OF THE SECOND AND THIRD ARGUMENTS OF BUILTIN FUNCTION 'SUBSTR' GREATER THAN STRING LENGTH PLUS ONE. RETURNED VALUE TRUNCATED AT END OF SOURCE STRING.

IEL0222I E SECOND ARGUMENT OF 'SUBSTR' SET TO ONE.

SECOND ARGUMENT OF BUILTIN FUNCTION 'SUBSTR' LESS THAN ONE. VALUE SET TO ONE.

IEL0223I E SECOND ARGUMENT OF 'SUBSTR' TOO LARGE.

SECOND ARGUMENT OF BUILTIN FUNCTION 'SUBSTR' GREATER THAN STRING LENGTH. NULL STRING RETURNED.

IEL0224I S UNINITIALIZED VARIABLE IN ARGUMENT LIST.

UNINITIALIZED VARIABLE USED IN BUILTIN FUNCTION ARGUMENT LIST. NULL STRING ASSUMED.

Explanation: The variable should be initialized before invoking the built-in function. If the FIXED parameter is matched with a null string argument, the parameter will assume a value of zero.

IEL0225I U CHARACTER STRINGS TOO LONG. COMPILATION TERMINATED IN PHASE P.

COMPILER RESTRICTION. CHARACTER STRING VARIABLES AND TEMPORARIES TOO LONG. COMPILATION TERMINATED IN PHASE P.

Explanation: The total length of all character preprocessor variables and all character string temporaries being used in the evaluation of a preprocessor expression cannot exceed a compiler maximum value. Compilation is terminated as the compiler dictionary has been filled up and no further information can be held in it.

Programmer Response: The error message identifies the preprocessor line being handled at the point of termination. Check the program for source errors or extremely long character string variables and correct or redesign the program if necessary. Alternatively, increase the storage available to the compiler; this might alleviate the problem.

IEL0226I E RIGHT PARENTHESIS ASSUMED.

RIGHT PARENTHESIS ASSUMED AFTER RETURNED VALUE IN '%PROCEDURE' STATEMENT.

Example:

```
%P: PROC RETURNS(CHAR;
```

IEL0227I E LEFT PARENTHESIS ASSUMED.

LEFT PARENTHESIS ASSUMED BEFORE MEMBER NAME.

Example:

```
%INCLUDE MEMBER);
```

IEL0228I E 'LENGTH' INVOKED WITH NO ARGUMENTS.

BUILTIN FUNCTION 'LENGTH' INVOKED WITH NO ARGUMENTS. FIXED ZERO RETURNED.

Example:

```
%A = LENGTH;
```

IEL0229I E 'INDEX' INVOKED WITH LESS THAN TWO ARGUMENTS.

BUILTIN FUNCTION 'INDEX' INVOKED WITH LESS THAN TWO ARGUMENTS. FIXED ZERO RETURNED.

Example:

```
   %A = INDEX ('ABCDE');
or %A = INDEX;
```

IEL0230I U COMPILER ERROR OR RESTRICTION NUMBER N DURING PHASE P.

COMPILER ERROR NUMBER N DURING PHASE P.

Explanation: An error has occurred during compilation or a compiler restriction has been exceeded. A detailed explanation of error number N is given at the end of this chapter in "Error and restriction numbers (0 to 946) for IEL0001I, IEL0230I, and IEL0970I" on page 104.

Programmer Response: (for Errors) Check the *PL/I VSE Diagnosis Guide.*

Programmer Response: (for Restrictions) Simplify the source program.

IEL0232I S 'PROCEDURE' ASSUMED AS FIRST STATEMENT.

FIRST STATEMENT NOT 'PROCEDURE'. 'PROCEDURE' STATEMENT ASSUMED.

Explanation: The first statement in a source program must be a PROCEDURE statement.

Programmer Response: The source program should be checked, particularly the control (that is, JCL and *PROCESS) statements and source margins. The source program should be correctly recorded on its input medium. Ensure that a PROCEDURE statement heads the source program.

IEL0233I E COLON ASSUMED [AFTER T].

T ASSUMED TO BE STATEMENT LABEL. COLON ASSUMED.

Example:

```
X GOTO Y:
```

Explanation: A statement keyword is preceded by a possible label, but no colon is present.

IEL0234I S INVALID SYNTAX. T IGNORED.

STATEMENT BEGINS WITH INVALID SYNTAX. T IGNORED.

IEL0235I S STATEMENT ASSUMED TO BE CONTINUATION OF 'DECLARE'.

STATEMENT BEGINS WITH INVALID SYNTAX. ASSUMED TO BE CONTINUATION OF PRECEDING 'DECLARE' STATEMENT.

Example:

```
DCL A FIXED DEC(2,0),
    B FIXED DEC(2,0);
    C FIXED DEC(2,0);
```

Explanation: An unrecognizable statement follows a DECLARE statement and is assumed to be a DECLARE statement also.

IEL0236I W INPUT RECORD LENGTH LESS THAN LEFT MARGIN.

LENGTH OF INPUT RECORD LESS THAN LEFT MARGIN OF 'MARGINS' OPTION. RECORD IGNORED.

Programmer Response: Check the use of the MARGINS compile-time option, and/or that a short record is intentional.

IEL0237I S INVALID CHARACTER [AFTER T]. T IGNORED.

TEXT IN OR FOLLOWING THIS STATEMENT CONTAINS INVALID CHARACTER [AFTER T]. T IGNORED.

Example:

```
CALL E(A,B,?);
```

Explanation: The presence of an invalid character might be detected before the start of a statement. Consequently, the statement number cannot be updated. When such an error is detected, the text is ignored from the start of the statement to the invalid character. The remaining characters in the statement will be treated as the complete statement. Consequently, other errors will almost certainly be indicated. These apparent errors will not be indicated if the program is recompiled with the invalid character corrected.

IEL0238I W CHARACTER STRING CONTAINS SEMICOLON.

CHARACTER STRING CONSTANT CONTAINS SEMICOLON.

Example:

```
STRING = 'B = C;';
```

Explanation: A common error is to omit one of a pair of quotation marks round a character string constant. The presence of a semicolon in a constant could be an indication of such an error, although it is not an error itself.

IEL0239I W COMMENT CONTAINS SEMICOLON.

COMMENTS IN OR FOLLOWING STATEMENT CONTAIN ONE OR MORE SEMICOLONS.

Example:

```
/* A = B; */
```

Explanation: A common error is to omit the delimiter "*/" after a comment. The presence of a semicolon in a comment could be an indication of such an error, although it is not an error in itself.

IEL0240I S QUOTE ASSUMED [AFTER T].

END OF SOURCE TEXT FOUND WITH UNMATCHED QUOTE. QUOTE ASSUMED [AFTER T].

Explanation: A quotation mark has been omitted causing the latter part of the program to appear as a string constant. A quote has been inserted prior to the first semicolon in this string. Note that statement numbers for statements following the statement in which the unmatched quote appears will not be printed.

Programmer Response: Check whether the quote was omitted or the source program is incomplete.

IEL0241I S 'END' STATEMENT(S) ASSUMED.

END OF SOURCE TEXT FOUND BEFORE LOGICAL END OF PROGRAM. N 'END' STATEMENT(S) ASSUMED.

Explanation: There are insufficient END statements to close all blocks. Any incomplete statements are ignored. Sufficient END statements are assumed in order to give valid nesting.

Programmer Response: Check the program block structure and that the source program is complete.

IEL0242I S STATEMENT TOO LARGE. T TO T IGNORED.

COMPILER RESTRICTION. STATEMENT EXCEEDS MAXIMUM LENGTH. TEXT IGNORED FROM T TO T.

Explanation: The statement is too long to be handled in the space that has been allotted to the compiler.

Programmer Response: Allot more storage using the SIZE parameter. If SIZE(MAX) is already being used then try to compile in a larger area. If this is not possible, then divide the statement or remove extra blanks.

IEL0243I S INVALID IDENTIFIER [AFTER T]. T REPLACED BY NULL.

INVALID IDENTIFIER FOLLOWING KEYWORD T. T REPLACED BY NULL STATEMENT.

Example:

1. `GOTO *;`
2. `CALL 1;`
3. `ON(A`

IEL0244I S QUOTE ASSUMED [AFTER T].

STATEMENT LENGTH MORE THAN COMPILER MAXIMUM AND CONTAINS UNMATCHED QUOTE. QUOTE ASSUMED [AFTER T].

Explanation: The compiler has assumed that the statement size appears to be too long because of the omission of a quote, and has assumed a quote prior to a semicolon within the statement. Note that statement numbers for statements following the statement in which the unmatched quote appears will not be printed.

IEL0245I S OPERAND INVALID [AFTER T].

OPERAND MISSING OR INVALID IN EXPRESSION [AFTER T].

Explanation: The compiler action depends on the context of the expression. A further message will indicate the action taken.

Programmer Response: Check for a further message for this statement.

IEL0246I S OPERATOR INVALID [AFTER T].

INVALID USE OF PREFIX OPERATOR [AFTER T].

Example:

```
    A = B + 4 ¬ C;
 |___________|
       T1
```

Explanation: The compiler action depends on the context of the expression. A further message will indicate the action taken.

Programmer Response: Check for the invalid use of an operator.

IEL0247I S INVALID SYNTAX. T REPLACED BY N.

INVALID SYNTAX IN 'IF' STATEMENT EXPRESSION. T HAS BEEN REPLACED BY N.

Example:

```
      T
  |----|
IF A+,B THEN GO TO LAB;
     ↑
   error
```

Explanation: The reason for the syntax error is diagnosed separately.

IEL0249I E T SHORTENED TO T.

COMPILER RESTRICTION. IDENTIFIER T TOO LONG. SHORTENED TO T.

Explanation: The identifier is more than 31 characters long. The first 16 and last 15 characters are retained. This can cause the identifier to be no longer unique.

IEL0250I W OPTION T OBSOLETE BUT ACCEPTED.

'ENVIRONMENT' OPTION T IS OBSOLETE BUT IS ACCEPTED.

Example:

```
Old - DCL F FILE ENV(V(100)...);
New - DCL F FILE ENV(V BLKSIZE(100)...):
```

IEL0251I S CONSTANT T TOO LONG.

COMPILER RESTRICTION. ARITHMETIC CONSTANT T IS TOO LONG.

Explanation: The number of digits allowed depends on the type of constant, that is, fixed or float. The expression containing the constant is ignored. Further action is indicated by subsequent messages depending on the context of the expression.

Programmer Response: Check the limits of the arithmetic constant and reduce it to an acceptable size.

IEL0252I S EXPONENT MISSING IN T.

EXPONENT MISSING IN FLOATING POINT CONSTANT T.

Example:

```
A = 123E * B
```

Explanation: The character E is present but there are no digits following it. The expression containing the constant is ignored. Further action is indicated by subsequent messages depending on the context of the expression.

IEL0253I S CHARACTER IN T NOT ZERO OR ONE.

CHARACTER IN BINARY CONSTANT T IS NOT ZERO OR ONE.

Explanation: The expression containing the constant is ignored. Further action is indicated by subsequent messages depending on the context of the expression.

Programmer Response: Check for a further message for this statement.

IEL0254I W BLANK ASSUMED [AFTER T].

NO BLANK BETWEEN CONSTANT AND FOLLOWING LETTER. BLANK ASSUMED [AFTER T].

Example:

```
DCL 1 STRUC, 2 CODE CHAR(3),
             2TEXT CHAR(77);
             ↑
             T
```

IEL0255I S EXPONENT OF T TOO LONG.

COMPILER RESTRICTION. EXPONENT OF CONSTANT T TOO LONG.

Explanation: A floating-point constant has an exponent that exceeds the implementation-defined limit. The expression containing the constant is ignored. Further action is indicated by subsequent messages depending on the context of the expression.

Programmer Response: Check for a further message for this statement.

IEL0256I S NO SIGNIFICANT DIGITS IN T.

CONSTANT T HAS NO SIGNIFICANT DIGITS.

Example:

```
1. A = .E2;
2. A = .E;
```

Explanation: The expression containing the "constant" is ignored. Further action is indicated by subsequent messages depending on the context of the expression.

Programmer Response: Check for a further message for this statement.

IEL0257I S CHARACTER IN T NOT ZERO OR ONE.

CHARACTER IN BIT STRING CONSTANT T IS NOT ZERO OR ONE.

Explanation: The expression containing the constant is ignored. Further action is indicated by subsequent messages depending on the context of the expression.

Programmer Response: Check for a further message for this statement.

IEL0258I S INVALID PRECISION T IGNORED.

PRECISION SPECIFICATION NOT AN UNSIGNED INTEGER. T IGNORED.

Example:

```
DCL G FIXED (+ABC) DECIMAL
             |---|
               T
```

IEL0259I S PRECISION TRUNCATED [AFTER T].

SECOND INTEGER MISSING FROM PRECISION SPECIFICATION. PRECISION TRUNCATED [AFTER T] AFTER FIRST INTEGER.

Example:

```
1. DCL A FIXED (9,X)
        --------------->
                T
2. DCL B FIXED (3,)
        --------->
          T
3. DCL C FIXED (4,D FLOAT;
        --------->
          T
```

Explanation: The base factor is assumed to be zero.

IEL0260I S INVALID CHARACTER [AFTER T1]. T2 IGNORED.

INVALID CHARACTER IN PICTURE [AFTER T1]. T2 IGNORED.

Example:

```
    T2
---------------
PIC '99W9'
--------
    T1
```

IEL0261I S PARENTHESIS MISSING [AFTER T1]. T2 IGNORED.

RIGHT PARENTHESIS MISSING FROM SCALING FACTOR OR REPETITION FACTOR IN PICTURE [AFTER T1]. T2 IGNORED.

Example:

```
     T2
|------------|
PIC '99F(2 ';
---------->
        T1
```

IEL0262I S INVALID REPETITION FACTOR [AFTER T1]. T2 IGNORED.

REPETITION FACTOR NOT AN UNSIGNED INTEGER IN PICTURE [AFTER T1]. T2 IGNORED.

Example:

```
1.
        T2
   |------------|
   DCL A PIC'(+3)9'
   ------->
        T1
2.
        T2
   |------------|
   DCL A PIC'S(A)9'
   ------->
        T1
```

IEL0263I S PICTURE INVALID [AFTER T1]. T2 IGNORED.

NO CHARACTER FOLLOWS REPETITION FACTOR IN PICTURE [AFTER T1]. T2 IGNORED.

Example:

```
        T2
      |------------|
DCL A PIC'99(3)';
      ------>
        T1
```

IEL0264I S PICTURE INVALID [AFTER T1]. T2 IGNORED.

'F' NOT FOLLOWED BY LEFT PARENTHESIS IN PICTURE [AFTER T1]. T2 IGNORED.

Example:

```
        T2
      |-------------|
DCL A PIC'99F3';
      ------>
        T1
```

IEL0265I S PICTURE INVALID [AFTER T1]. T2 IGNORED.

INVALID SCALING FACTOR IN PICTURE [AFTER T1]. T2 IGNORED.

Example:

```
1.
          T2
        |-------------|
  DCL A PIC'99F(*)';
          -------->
          T1
2.
          T2
        |-------------|
  DCL A PIC'99F( )';
          -------->
          T1
```

IEL0266I S STATEMENT INVALID AFTER 'ELSE'

NON-EXECUTABLE STATEMENT FOLLOWING 'ELSE'. NULL STATEMENT ASSUMED AS 'ELSE' CLAUSE.

Explanation: A null statement is assumed after the word ELSE so that the non-executable statement is no longer the ELSE clause.

IEL0267I S STATEMENT INVALID AFTER 'THEN'.

NON-EXECUTABLE STATEMENT FOLLOWING 'THEN'. NULL STATEMENT ASSUMED AS 'THEN' CLAUSE.

Explanation: A null statement is assumed as the THEN clause, forcing the non-executable statement out of the compound IF statement.

IEL0268I S REFERENCE TO UNKNOWN LABEL IGNORED.

LABEL REFERENCED BY 'END' STATEMENT CANNOT BE MATCHED. REFERENCE IGNORED.

IEL0269I U TOO MANY 'PROCEDURE' 'BEGIN' AND 'ON' STATEMENTS.

COMPILER RESTRICTION. TOO MANY 'PROCEDURE' 'BEGIN' AND 'ON' STATEMENTS IN THE PROGRAM.

Explanation: The implementation restriction on the number of blocks in a compilation has been exceeded.

Programmer Response: Subdivide the program into two or more procedures for separate compilation, or rewrite it with less blocks.

IEL0270I U 'BEGIN' OR 'PROCEDURE' NESTING EXCEEDS MAXIMUM.

COMPILER RESTRICTION. 'BEGIN' OR 'PROCEDURE' STATEMENT NESTING MORE THAN MAXIMUM LEVEL.

Explanation: The implementation restriction on the level to which blocks can be nested has been exceeded.

Programmer Response: Reorganize the program to contain fewer levels of nested blocks.

IEL0271I S 'THEN' ASSUMED [AFTER T].

KEYWORD 'THEN' ASSUMED [AFTER T] IN 'IF' STATEMENT.

Example:

```
1. IF A = B GOTO L;
2. IF B&C IF D&E THEN DO;.....
```

Explanation: The keyword THEN is missing from or incorrectly placed in the IF statement.

Programmer Response: Check the IF statement.

IEL0272I S INVALID 'ON' UNIT. NULL STATEMENT ASSUMED.

INVALID ON-UNIT SPECIFIED. NULL STATEMENT ASSUMED.

Explanation: The specified statement might be a labeled statement, or an unlabeled statement not allowed as an ON-unit. The null statement is assumed as the ON-unit, and the text of the invalid ON-unit is treated as one or more separate statements.

IEL0273I E INVALID PREFIXES ON KEYWORD 'ELSE' OR 'WHEN' OR 'OTHERWISE'. FOR 'ELSE' PREFIXES ASSUMED TO PRECEDE FOLLOWING STATEMENT. FOR 'WHEN' OR 'OTHERWISE' PREFIXES IGNORED.

Example:

```
IF A THEN B = 3;
L: ELSE B = 4;
```

Explanation: Labels and condition prefixes are transferred to the statement following ELSE.

IEL0274I S STATEMENT INVALID AFTER 'THEN'.

STATEMENT MISSING OR INVALID AFTER 'THEN'. NULL STATEMENT ASSUMED AS 'THEN' CLAUSE.

Example:

```
IF A THEN ELSE B = 4;
```

Explanation: No unit has been provided for the THEN clause.

IEL0275I S STATEMENT INVALID AFTER 'ELSE'.

STATEMENT MISSING OR INVALID AFTER 'ELSE'. NULL STATEMENT ASSUMED AS 'ELSE' CLAUSE.

Example:

```
IF A THEN IF B THEN C = D; ELSE ELSE E = 4;
```

Explanation: No unit has been provided for the ELSE clause.

IEL0276I S 'ELSE' IN INVALID POSITION IGNORED.

KEYWORD 'ELSE' APPEARS IN INVALID POSITION. 'ELSE' IGNORED.

Example:

```
IF A THEN B = C; D = E; ELSE J = K;
```

Programmer Response: Correct the source program. Check that THEN clause in nested IF statements are correct.

IEL0277I W 'SYSIN' OR 'SYSPRINT' ASSUMED FOR I/O 'ON' CONDITION.

I/O ON CONDITION HAS NO FILE NAME SPECIFIED. 'SYSIN' OR 'SYSPRINT' ASSUMED.

Example:

```
ON ENDFILE SNAP;
ON ENDPAGE PUT PAGE;
```

Explanation: ENDFILE (SYSIN) is assumed for input, and ENDPAGE (SYSPRINT) is assumed for output. All other I/O conditions are ignored and are assumed to be replaced by ON ERROR.

IEL0278I S INVALID CONDITION [AFTER T1]. T2 REPLACED BY 'ERROR'.

INVALID 'ON' CONDITION NAME [AFTER T1]. T2 REPLACED BY 'ERROR'.

Example:

```
    ON FRED A = B;
----> |---------|
    T1     T2
```

IEL0279I S REDUNDANT COMMA [AFTER T] IGNORED.

MISSING ITEM OR REDUNDANT COMMA IN LIST [AFTER T]. COMMA IGNORED.

Example:

```
PUT DATA (,B,C);
```

Explanation: An expected item has not been found following a left parenthesis or comma in a list, for example: a parameter list, a FREE statement list, or a data list. The comma is ignored, or the whole list is ignored if it becomes null. Further action in addition to ignoring the null list is indicated by subsequent messages depending on the type of list concerned.

Programmer Response: Correct the source program. Check also for further messages.

IEL0280I E LEFT PARENTHESIS ASSUMED [AFTER T].

Example:

```
DO WHILE X = Y);
```

Explanation: A left parenthesis has been omitted.

IEL0281I S ITERATIVE SPECIFICATION INVALID [AFTER T1].

Example:

```
DO I TO 3;
```

Explanation: The control variable of expression 1 is missing.

IEL0282I S EXPRESSION MISSING AFTER 'TO' OR 'BY'.

EXPRESSION FOLLOWING 'TO' OR 'BY' IS MISSING IN 'DO' STATEMENT. NON-ITERATIVE 'DO' ASSUMED.

IEL0283I S 'RETURN' STATEMENT WITHIN ON-UNIT IGNORED.

'RETURN' STATEMENT IS WITHIN ON-UNIT. STATEMENT IGNORED.

Example:

```
ON OVERFLOW RETURN;
```

IEL0284I S 'IN' NOT FOLLOWED BY LEFT PARENTHESIS.

KEYWORD 'IN' NOT FOLLOWED BY LEFT PARENTHESIS. 'IN' IGNORED.

IEL0285I S LABEL MISSING. DUMMY ASSUMED.

LABEL MISSING FROM 'PROCEDURE' OR 'ENTRY' STATEMENT. ONE HAS BEEN ASSUMED.

IEL0286I S 'ENTRY' IN 'BEGIN' BLOCK IGNORED.

'ENTRY' STATEMENT IS IN A 'BEGIN' BLOCK. STATEMENT IGNORED.

Example:

```
E: BEGIN;
E: ENTRY;
END E;
```

Explanation: The ENTRY statement and its labels are ignored.

IEL0287I S 'IN' OPTION INVALID [AFTER T¹]. T² IGNORED.

INVALID 'IN' OPTION [AFTER T¹] IN 'FREE' STATEMENT. T² IGNORED.

Example:

```
    FREE FRED IN (25 + AREA);
                ↑
                T¹
             |-------------|
                   T²
```

IEL0288I S INVALID TEXT. T IGNORED.

INVALID TEXT WITHIN STATEMENT. T IGNORED.

Explanation: Invalid text has been found within a statement, for example, an invalid attribute or option. The text is ignored. Scanning of the source program restarts at the next recognizable item.

IEL0289I S 'END' FOUND BEFORE END OF SOURCE TEXT.

LOGICAL END OF PROGRAM FOUND BEFORE END OF SOURCE TEXT. STATEMENT IGNORED.

Example:

```
P:  PROC OPTIONS (MAIN);
END;  (message produced here)
GOTO LAB;
END;
```

Explanation: In order to check the syntax of the whole source text, the END statement which prematurely terminates the program has been ignored. This might cause some extra errors in subsequent PROC, BEGIN, or END statements.

IEL0290I S INVALID OPTION [AFTER T¹]. T² IGNORED.

INVALID OR MULTIPLE SPECIFICATION OF OPTION [AFTER T¹]. T² IGNORED.

Explanation: The option might:

1. Have an invalid argument
2. Be specified more than once
3. Be spelled incorrectly
4. Have no argument

IEL0291I E INVALID SYNTAX [AFTER T] IN 'LABEL' ATTRIBUTE.

INVALID SYNTAX FOR LABEL CONSTANT [AFTER T¹]. T² IGNORED.

Example:

```
           T²
           |----------------|
DCL LAB LABEL(LAB1,6AB2,LAB3);
        ---------->
                  T¹
```

Explanation: The compiler has detected an item in the list of label constants which does not begin with an alphabetic character.

Programmer Response: Correct the specification of the label constant.

IEL0292I S LABEL LIST TOO LONG. T IGNORED.

COMPILER RESTRICTION. LABEL PREFIX LIST TOO LONG. LABEL T HAS BEEN IGNORED.

Explanation: The number of label prefixes plus the total number of characters in the label list must not exceed 254. The label prefix list is truncated at the nearest point below the allowed maximum.

Programmer Response: The program should be rewritten with shorter or fewer labels prefixed to this statement. Excess labels might be transferred to an immediately preceding null statement.

IEL0293I S INVALID PREFIX [AFTER T]. T IGNORED.

INVALID CONDITION PREFIX [AFTER T]. T IGNORED.

Example:

```
(DUBRG) PROC1: PROCEDURE;
```

Explanation: An invalid condition prefix is specified.

IEL0294I E T FOLLOWS LABEL BUT IS ACCEPTED.

CONDITION PREFIX T FOLLOWS LABEL BUT IS ACCEPTED.

Example:

```
L: (FOFL): A = B;
```

Explanation: Condition prefix lists should precede any statement label lists. However, this compiler allows condition prefixes to follow any statement labels.

IEL0295I S T FOLLOWS LABEL AND IS IGNORED.

'CHECK' OR 'NOCHECK' CONDITION PREFIX FOLLOWS LABEL. T IGNORED.

Example:

```
L: (CHECK(A),FOFL) : A = B;
   |-------|
       T
```

Explanation: The check list should precede a label. The syntax of the CHECK condition is still analyzed at compile time; however, the CHECK condition is no longer supported and is always disabled at run time.

IEL0296I E COLON ASSUMED [AFTER T].

COLON ASSUMED AFTER T. PARENTHESIZED ITEM ASSUMED TO BE CONDITION PREFIX.

Example:

```
(FIXEDOVERFLOW) A = B*C;
```

IEL0297I S ARGUMENT LIST INVALID [AFTER T^1]. T^2 IGNORED.

ARGUMENT LIST MISSING OR INVALID [AFTER T^1]. T^2 IGNORED.

Example:

```
READ FILE(F) INTO (3);
    -------------↑
                T1
          |-------|
              T2
```

IEL0298I E CONDITION PREFIX INVALID.

CONDITION PREFIX INVALID ON THIS STATEMENT. PREFIX LIST IGNORED.

Explanation: Condition prefix lists are invalid on ENTRY, DECLARE, DEFAULT, and FORMAT statements.

IEL0299I S FACTORING INVALID [AFTER T].

FACTORING SPECIFIED IN 'ALLOCATE' STATEMENT [AFTER T]. TEXT IGNORED TO NEXT SEMICOLON.

Explanation: No factoring of parentheses or factored attributes are allowed in an ALLOCATE statement. The ALLOCATE statement is ignored and a null statement is assumed.

IEL0300I S 'INITIAL' FACTORING LEVEL [AFTER T] EXCEEDS N.

COMPILER RESTRICTION. FACTORING LEVEL [AFTER T] IN 'INITIAL' EXCEEDS N. ATTRIBUTE IGNORED.

IEL0301I S SIGN IN T IGNORED.

SIGN IN STRUCTURE LEVEL NUMBER T IGNORED.

Example:

```
DCL + 1 A,
      2 B,
      2 C;
```

Explanation: The level number in a DECLARE statement must be an unsigned decimal integer.

IEL0302I S ZERO [AFTER T] ASSUMED TO BE ONE.

ZERO LEVEL NUMBER [AFTER T] ASSUMED TO BE ONE.

Explanation: The level number in a DECLARE statement must be an unsigned nonzero integer.

IEL0303I S T IN 'RETURNS' INVALID.

ATTRIBUTE T IN 'RETURNS' INVALID. ATTRIBUTE IGNORED.

IEL0304I S INVALID SYNTAX [AFTER T^1]. T^2 IGNORED.

INVALID SYNTAX IN ASSIGNMENT STATEMENT [AFTER T^1]. T^2 IGNORED.

Example:

```
1.
      T2
   |--------|
   A + B = C;
   --->
      T1
2.
   |----|
   A = ;
   ---->
   T1
```

IEL0305I W INVALID USE OF LISTING CONTROL STATEMENT.

INVALID USE OF LISTING CONTROL STATEMENT. STATEMENT NOT IMPLEMENTED.

Explanation: The listing control statements must appear between statements and on a separate line from them.

IEL0306I S NO MATCHING FORMAT LIST [AFTER T].

EDIT DATA LIST HAS NO MATCHING FORMAT LIST [AFTER T]. T FORMAT ASSUMED.

Explanation: Edit-directed transmission statements require format lists.

IEL0307I S INVALID SYNTAX [AFTER T1]. T2 IGNORED.

INVALID SYNTAX IN DATA LIST [AFTER T1]. T2 IGNORED.

Example:

```
GET DATA (A,B,(C(I) DO I = 1 TO 3),D);
```

Explanation: The data list has an item missing or has an error in a DO-loop specification. The data list is ignored from the invalid item.

IEL0308I S FORMAT LIST INVALID [AFTER T1]. T2 IGNORED.

FORMAT ITEM MISSING OR INVALID [AFTER T1]. T2 IGNORED.

Example:

```
PUT EDIT (C) (A(3), J(2));
```

Explanation: The format item has been omitted, has an invalid argument, or is incorrectly spelled. The invalid item is ignored, and the text is scanned for the next item.

IEL0309I W 'FORMAT' STATEMENT HAS NO LABEL.

Explanation: A FORMAT statement cannot be referenced without a label.

IEL0310I S IDENTIFIER REFERENCED BY 'LEAVE' STATEMENT CANNOT BE MATCHED. REFERENCE IGNORED.

IDENTIFIER REFERENCED BY 'LEAVE' STATEMENT IS EITHER MISSING OR NOT ON A 'DO' STATEMENT. REFERENCE IS IGNORED.

Example:

```
P: PROC OPTIONS(MAIN);
LAB1: DO;
LEAVE LAB2;
END P;
```

IEL0311I S COMMENT DELIMITER ASSUMED [AFTER T].

END OF SOURCE TEXT FOUND WITHIN A COMMENT. COMMENT DELIMITER ASSUMED [AFTER T].

Explanation: A comment delimiter might have been omitted, causing the latter part of the program to appear as a comment. A comment delimiter is inserted at the end of the last source statement.

For graphic support, a right delimiter might have been omitted, causing the latter part of the program to appear as a comment. A right delimiter is inserted at the end of the last source statement.

Programmer Response: Check whether the comment or right delimiter has been omitted or if the source program is incomplete.

IEL0312I U NO TEXT IN PROGRAM.

Example:

```
*PROCESS A,X;
/* PROGRAM STARTS HERE */
  ⋮
```

Explanation: The first comment delimiter is in positions 1 and 2 of the record following the PROCESS statement, and is interpreted as an end-of-file delimiter for the input to the compiler. The compiler has not received any source statements to compile into an object module. Reasons for this include the error shown above, control statements out of sequence, and so on.

IEL0313I S INVALID KEYWORD [AFTER T1]. T2 IGNORED.

INVALID KEYWORD [AFTER T1] IN REPETITIVE SPECIFICATION. T2 IGNORED.

Example:

```
PUT LIST((A(I) DO I = 3 IF A > B));
                        ↑
                        T1
                  |---------------|
                        T2
```

IEL0314I S END OF SOURCE TEXT FOUND. T IGNORED.

END OF SOURCE TEXT FOUND BEFORE END OF STATEMENT. T IGNORED.

Example:

```
1. DCL A FIXED, B FLOAT, C STATIC
   (end of file)
2. A = B; C = D + (end of file)
```

Explanation: This can happen in addition to “end of source text found before logical end of program”.

IEL0315I S LABEL ON 'ON' UNIT IGNORED.

'ON' UNITS CANNOT BE LABELED. LABEL IGNORED.

Example:

```
ON OFL L: GOTO LAB;
The label L is invalid.
```

IEL0316I S SEMICOLON ASSUMED [AFTER T].

END OF STATEMENT ASSUMED [AFTER T]. TEXT IGNORED TO NEXT SEMICOLON.

Example:

```
DELAY (25) CALL SUBRTN;
```

Explanation: A semicolon has not been found where expected after a syntactically correct statement (after the (25) in the above example), so one is assumed.

IEL0317I S ATTRIBUTE INVALID [AFTER T1]. T2 IGNORED.

INVALID ATTRIBUTE SPECIFICATION [AFTER T1]. T2 IGNORED.

Example:

```
        T2
       |---------|
DCL JOE FOXED;
------->
   T1
```

IEL0318I S 'DO' IN 'ON-UNIT' REPLACED BY 'BEGIN'.

'DO' STATEMENT IS INVALID IN 'ON' UNIT. REPLACED BY 'BEGIN'.

Example:

```
ON OFL DO;
PUT SKIP;
END;
```

Explanation: The only valid ON-units are single statements or begin blocks.

IEL0319I S MULTIPLE USE OF OPTION. T IGNORED.

STATEMENT USES AN OPTION MORE THAN ONCE. T IGNORED.

Example:

```
                        T
                    |--------|
DISPLAY(A) EVENT(B) EVENT(C) REPLY(R);
```

IEL0320I S NO 'REPLY' OPTION. TEXT [AFTER T] IGNORED.

'DISPLAY' STATEMENT HAS NO 'REPLY' OPTION. TEXT [AFTER T] IGNORED.

Example:

```
DISPLAY ('HELP') EVENT (E);
 |--------------|
       T
```

IEL0321I E LEFT PARENTHESIS ASSUMED BEFORE EXPRESSION.

MISSING LEFT PARENTHESIS ASSUMED BEFORE EXPRESSION IN 'DELAY' OR 'DISPLAY' STATEMENT.

Example:

```
DISPLAY MESSAGE);
```

IEL0322I S INVALID FORMAT ITEM [AFTER T1]. T2 IGNORED.

INVALID SPECIFICATION IN FORMAT ITEM [AFTER T1]. T2 IGNORED.

IEL0323I E RIGHT PARENTHESIS ASSUMED [AFTER T].

REPETITIVE SPECIFICATION ENDING AT T IN DATA LIST NOT FOLLOWED BY RIGHT PARENTHESIS. ONE HAS BEEN ASSUMED.

Example:

```
PUT EDIT ((A(I) DO I = 1 TO 3  (F(3));
                              ↑

Missing right parenthesis assumed to be here.
```

Explanation: Repetitive specifications in data lists must be enclosed in brackets.

IEL0324I S NESTING LEVEL EXCEEDS N [AFTER T].

COMPILER RESTRICTION. LEVEL OF NESTING EXCEEDS N IN DATA LIST [AFTER T]. STATEMENT IGNORED.

Explanation: If there are no redundant brackets, rewrite the statement within the implementation limits.

IEL0325I S INVALID SYNTAX IN 'CALL' STATEMENT.

INVALID SYNTAX IN 'CALL' STATEMENT. STATEMENT IGNORED.

Example:

```
CALL (A,B);
```

IEL0326I S 'ENTRY' AND LABEL INSIDE 'DO' IGNORED.

'ENTRY' STATEMENT AND LABEL INSIDE ITERATIVE 'DO' IGNORED.

Example:

```
DO I = 1 TO 3;
A(I) = B(I);
E: ENTRY;
A(I) = C(I);
END;
```

Explanation: Because the label is ignored, calls to it will be unresolved.

IEL0327I S INVALID SYNTAX. T IGNORED.

STATEMENT BEGINS WITH INVALID SYNTAX. T IGNORED.

Example:

```
1. 13 14) * X¬ |);
   No identifier found in this statement.
2. IF A ,GOTO LAB;
      ↑
   'THEN' assumed here
```

",GOTO LAB;" ignored since error follows a fix.

Explanation: Either the statement type could not be identified, or, due to fixing an error in the previous statement, recovery was not attempted from the error in the current statement.

IEL0328I S INVALID OPTION [AFTER T]. T IGNORED.

INVALID OPTION [AFTER T] IN 'PROCEDURE' 'BEGIN' OR 'ENTRY' STATEMENT. T IGNORED.

Example:

```
1. P: PROC MAIN;                       /* invalid */

2. B: BEGIN (A,B);                     /* invalid */

3. P: PROC EXT('P32WZ') OPTIONS(ASM)   /* invalid */
```

IEL0329I S NESTING LEVEL EXCEEDS N [AFTER T].

COMPILER RESTRICTION. LEVEL OF NESTING EXCEEDS N IN FORMAT LIST [AFTER T]. STATEMENT IGNORED.

Programmer Response: If there are no redundant brackets, rewrite the statement within the implementation limits.

IEL0330I S NO '=' [AFTER T1]. T2 IGNORED.

NO '=' [AFTER T1] IN REPETITIVE SPECIFICATION. T2 IGNORED.

Example:

```
PUT LIST ((A(I) DO I TO N));
                     ↑
                     T1
                 |--------|
                     T2
```

IEL0331I S INVALID CONTROL VARIABLE [AFTER T1]. T2 IGNORED.

INVALID CONTROL VARIABLE [AFTER T1] IN REPETITIVE SPECIFICATION. T2 IGNORED.

Example:

```
PUT LIST ((A(I) DO 3 TO 4));
                   ↑
                   T1
                |--------|
                    T2
```

IEL0332I S PARENTHESIS NESTING LEVEL EXCEEDS N [AFTER T].

COMPILER RESTRICTION. LEVEL OF PARENTHESIS NESTING GREATER THAN N [AFTER T]. STATEMENT IGNORED.

Example:

```
A(B(C(_________(A3(B3
                ↑
                T
```

IEL0333I U STATEMENT NESTING LIMIT EXCEEDED.

COMPILER RESTRICTION. NESTING LIMIT OF 'PROCEDURE'| 'BEGIN'|'IF'|'DO'|'SELECT' STATEMENT HAS BEEN EXCEEDED. PROCESSING TERMINATED.

Explanation: The stack containing PROCEDURE, BEGIN, IF, DO, and SELECT statements and their labels has overflowed.

Programmer Response: Either reduce the number or length of the labels on these statements or restructure the program to reduce the depth of nesting.

IEL0334I S OPTION(S) T MISSING FROM STATEMENT.

OPTION(S) T MISSING FROM RECORD I/O STATEMENT. STATEMENT IGNORED.

Example:

```
READ FILE(F) KEYTO(K);
(INTO option missing)
WRITE FILE(F);
(FROM option missing)
```

Programmer Response: Ensure that a correct set of options is specified for this statement.

IEL0335I S T IN 'OPTIONS' LIST IGNORED.

INVALID ITEM IN 'OPTIONS' LIST. T IGNORED.

Example:

```
1. P: PROC OPTIONS(NOAN);
2. P: PROC OPTIONS (NOMAPIN(3));
```

Programmer Response: Check the list of valid options and their specification.

IEL0336I S VARIABLE MISSING FROM 'LOCATE'.

VARIABLE MISSING FROM 'LOCATE' STATEMENT. STATEMENT IGNORED.

IEL0337I E COLON ASSUMED [AFTER T].

CONDITION PREFIX NOT FOLLOWED BY COLON. COLON ASSUMED [AFTER T].

IEL0338I S MULTIPLE 'TO', OR 'BY', OR 'REPEAT' [AFTER T1]. T2 IGNORED.

MULTIPLE 'TO', OR 'BY', OR 'REPEAT' [AFTER T1]. IN REPETITIVE SPECIFICATION T2 IGNORED.

IEL0339I S FILE OPTION MISSING. T IGNORED.

MISSING FILE OPTION OR REDUNDANT COMMA IN 'OPEN' OR 'CLOSE' STATEMENT. T IGNORED.

Example:

```
OPEN FILE(F2), FILE(F3) STREAM, OUTPUT;
                                |----|
                                  T
```

IEL0340I S INVALID SYNTAX [AFTER T1]. T2 IGNORED.

INVALID SYNTAX IN 'ENVIRONMENT' OPTION [AFTER T1]. T2 IGNORED.

Example:

```
DCL JOB ENV(REROAD, HIGHINDEX(2741));
           ↑          ↑
         error      error
```

Explanation: Possible causes for this are:

1. An invalid keyword, or keyword subset has been used (only LEAVE and REREAD are valid in the CLOSE statement).
2. An option has an incorrect or missing argument.

IEL0341I S INVALID ITEM [AFTER T1]. T2 IGNORED.

INVALID ITEM IN PARAMETER LIST [AFTER T1]. T2 IGNORED.

Example:

```
            T2
           |---|
P: PROC(P1, 3*P2);
-----------↑
    T1
END P;
```

IEL0342I S INVALID OPTION IN [AFTER T]. T IGNORED.

INVALID OPTION IN 'DEFAULT' STATEMENT [AFTER T]. T IGNORED.

Example:

```
DEFAULT LIST(A:B) FIXED;
```

Explanation: The only valid options of the DEFAULT statement are RANGE and DESCRIPTORS.

IEL0343I S INVALID IDENTIFIER AFTER [AFTER T1]. T2 IGNORED.

INVALID IDENTIFIER IN 'RANGE' SPECIFICATION.

Example:

```
DEFAULT RANGE (A:BC) BINARY;
```

Explanation: The syntax rules for the RANGE option of the DEFAULT statement are given in the *PL/I VSE Language Reference.*

IEL0344I S INVALID IDENTIFIER [AFTER T1]. T2 IGNORED.

INVALID IDENTIFIER [AFTER T1] IN 'GENERIC' SPECIFICATION. T2 IGNORED.

Explanation: The syntax rules for the GENERIC attribute are given in the *PL/I VSE Language Reference.*

IEL0345I S INVALID EXPRESSION IN 'POSITION' ATTRIBUTE. 'POS()' IGNORED.

INVALID EXPRESSION IN 'POSITION' ATTRIBUTE. T IGNORED.

Example:

```
DCL P1 CHAR(8),
    P2 CHAR(4) DEF P1 POS();
```

IEL0346I S INVALID IDENTIFIER [AFTER T^1]. T^2 IGNORED.

INVALID IDENTIFIER [AFTER T^1] IN NAME LIST. T^2 IGNORED.

Example:

```
ON CHECK(A,3) GOTO LAB;
          ↑↑
         T1T2
```

IEL0347I S INVALID KEYWORD. T IGNORED.

INVALID KEYWORD IN ATTRIBUTE SPECIFICATION IN 'DEFAULT' STATEMENT. T IGNORED.

Example:

```
DEFAULT RANGE(A:B) FIXED READ;
```

Explanation: The syntax rules for the DEFAULT statement are given in the *PL/I VSE Language Reference.*

IEL0348I S 'WHEN' OPTION MISSING [AFTER T^1]. T^2 IGNORED.

'WHEN' OPTION MISSING [AFTER T^1] IN 'GENERIC' SPECIFICATION. T^2 IGNORED.

Example:

```
DCL E ENTRY GENERIC(E1 IF (FLOAT));
          __________↑
                     T1
                     |-------------|
                           T2
```

Explanation: The rules for the GENERIC attribute are given in the *PL/I VSE Language Reference.*

IEL0349I S INVALID EXPRESSION [AFTER T^1]. T^2 REPLACED BY 10.

INVALID EXPRESSION [AFTER T^1] IN DIMENSION SPECIFICATION. T^2 REPLACED BY 10.

Example:

```
DCL A(P + Q,P - Q,P/+-):
          ------> ↑
            T1     T2
           |--|  |--|
```

Explanation: The erroneous expression is replaced to ensure that the required number of array dimensions is maintained. Subsequent subscripted references to the array will be correct if this number of dimensions is used.

IEL0350I S INVALID OPTION [AFTER T^1]. T^2 IGNORED.

INVALID OPTION IN 'GET' OR 'PUT' STATEMENT [AFTER T^1]. T^2 IGNORED.

Example:

```
PUT LIST(A) TWICE;
----------↑ |----|
      T1      T2

GET PAGE DATA(D);
--↑ |--|
 T1  T2
```

Explanation: The option is invalid or inapplicable to this type of statement.

IEL0351I S EXPRESSION INVALID OR MISSING.

EXPRESSION INVALID OR MISSING IN 'DELAY' OR 'DISPLAY' STATEMENT. STATEMENT IGNORED.

Example:

```
1. DELAY;
2. DISPLAY ) ++;
```

Explanation: If an erroneous expression causes this message to be produced, it will also be indicated by a separate message.

IEL0352I S INVALID OPTION [AFTER T^1]. T^2 IGNORED.

INVALID SPECIFICATION OF OPTION [AFTER T^1]. T^2 IGNORED.

IEL0353I E COMMA ASSUMED [AFTER T].

CONSTANT FOUND IN ATTRIBUTE LIST. COMMA ASSUMED [AFTER T].

Example:

```
DCL 1 STRUCT,
    2 FRED (comma missing here)
    3 JOE FLOAT;
```

Explanation: This error, and its correction by the compiler, can only occur where structure levels are used in a DECLARE statement.

IEL0354I S NO IDENTIFIER [AFTER T^1]. T^2 IGNORED.

'DECLARE' 'DEFAULT' OR 'ALLOCATE' DOES NOT HAVE AN IDENTIFIER [AFTER T^1]. T^2 IGNORED.

Example:

```
DCL 1 J, 2 + FIXED, 2 F FLOAT;
    ------↑      |
          T1     |
        |---------|
          T2
```

IEL0355I S DATA LIST MISSING [AFTER T].

DATA LIST MISSING [AFTER T]. STATEMENT IGNORED.

Example:

```
PUT EDIT SKIP(3);
--------↑
        T
```

Explanation: Only data-directed output statements can be used without a data list.

IEL0356I S INVALID IDENTIFIER [AFTER T1]. T2 IGNORED.

INVALID IDENTIFIER [AFTER T1] IN 'FREE' STATEMENT. T2 IGNORED.

Example:

```
FREE A,B, (C.D) IN (AREA);
     ---↑
        T1
        |------|
          T2
```

IEL0357I W TOO FEW PARENTHESES FOR TEXT [AFTER T] TO BE 'DO' SPECIFICATION.

DATA LIST CONTAINS TOO FEW PARENTHESES FOR TEXT [AFTER T] TO BE REPETITIVE SPECIFICATION. ASSUMED TO BE DATA LIST ITEMS.

Example:

```
PUT DATA (A(I) DO I = 3 TO 4);
should be:
PUT DATA ((A(I) DO I = 3 TO 4));
but is assumed to be:
PUT DATA (A(I), DO...etc.,
```

or

Example:

```
PUT LIST (((A(I,J) DO I = 1 TO 2)
    DO J = 3 TO 4));
```

Explanation: A repetitive specification must leave extra brackets for each do-group.

IEL0358I S NO EXPRESSION [AFTER T1]. T2 ASSUMED.

EXPRESSION MISSING FROM FORMAT ITEM [AFTER T1]. T2 ASSUMED.

Example:

```
PUT EDIT (A) (F(3),X );
--------------------↑
                    T1
```

IEL0359I S PREFIX OPTIONS CONFLICT.

PREFIX OPTIONS CONFLICT. THE DISABLING PREFIX HAS BEEN ASSUMED.

Example:

```
(CONV,OFL,NOCONV): A = B + C;
```

IEL0360I S NO EXPRESSION [AFTER T].

EXPRESSION MISSING FROM 'A' FORMAT ITEM [AFTER T]. 'ERROR' CONDITION WILL BE RAISED ON EXECUTION.

Explanation: On input, an edit-directed A-format item must specify the number of characters to be read.

IEL0361I S WRONG NUMBER OF ARGUMENTS [AFTER T1]. T2 IGNORED.

WRONG NUMBER OF ARGUMENTS IN 'FORMAT' ITEM [AFTER T1]. T2 IGNORED.

Example:

```
PUT EDIT (A) (F(A,B,3,4), E(3));
```

IEL0362I E COMMA ASSUMED [AFTER T].

Example:

```
PUT EDIT (A) (B,A X(2));
```

Explanation: A comma is assumed wherever the syntax of a statement requires one in order to be valid.

IEL0363I S PICTURE INVALID [AFTER T]. T IGNORED.

CHARACTER SPECIFICATION [AFTER T] IN PICTURE IS INVALID IN COMPLEX FORMAT ITEM. T IGNORED.

Example:

```
PUT EDIT (A) (C(F(3),P'99A'));
```

IEL0364I E INVALID SYNTAX [AFTER T].

INVALID SYNTAX IN LISTING CONTROL STATEMENT [AFTER T]. TEXT IGNORED TO NEXT SEMICOLON.

Example:

```
%SKIP(1
-------↑
       T
  A=B;
```

IEL0365I S INVALID SYNTAX [AFTER T].

INVALID SYNTAX IN 'DECLARE' OR 'DEFAULT' STATEMENT [AFTER T]. STATEMENT IGNORED.

Explanation: A statement beginning with either "DCL(..." or "DEFAULT(..." that is not a DECLARE or DEFAULT statement has been encountered and cannot be compiled.

Programmer Response: Replace the identifier DCL (or DECLARE) or DEFAULT with an identifier that is not a keyword, and recompile the program.

IEL0366I W STATEMENT NOT SUPPORTED.

STATEMENT IS NOT SUPPORTED AND IS IGNORED.

Example:

```
HALT;
```

Explanation: The PL/I statements CHECK, NOCHECK, FLOW, NOFLOW, and HALT are not supported by the compiler and are ignored if they appear in the source program.

IEL0367I W INVALID SYNTAX AFTER T. STATEMENT NOT SUPPORTED.

INVALID SYNTAX AFTER T. STATEMENT IS NOT SUPPORTED AND IS IGNORED.

Example:

```
CHECK(A,2,B);
```

Explanation: The PL/I statements CHECK and NOCHECK are not recognized by the compiler.

IEL0368I W OPTION T NOT SUPPORTED.

OPTION T NOT SUPPORTED. STATEMENT IGNORED.

Example:

```
PUT ALL;
```

Explanation: The ALL, FLOW, and SNAP options of the PUT statement are not recognized by the compiler.

IEL0370I S DATA LIST INVALID [AFTER T1]. T2 IGNORED.

INVALID USE OF REPETITIVE SPECIFICATION IN DATA LIST FOR 'GET' STATEMENT [AFTER T1]. T2 IGNORED.

Explanation: A repetitive specification is not allowed in a GET DATA statement.

IEL0371I S FORMAT LIST INVALID [AFTER T].

FORMAT LIST MISSING OR INVALID AFTER T IN 'FORMAT' STATEMENT. 'A' FORMAT ASSUMED.

IEL0372I W INVALID CARRIAGE CONTROL CHARACTER T.

CARRIAGE CONTROL CHARACTER T IS INVALID. BLANK ASSUMED FOR CHARACTER.

Explanation: An invalid ANS print control character has been specified in a source record associated with the given statement. The permissible characters are: blank, 0, -, +, and 1.

IEL0373I S PICTURE T EXCEEDS MAXIMUM LENGTH.

COMPILER RESTRICTION. 'PICTURE' SPECIFICATION EXCEEDS MAXIMUM LENGTH. 'PICTURE' SPECIFICATION T IGNORED.

Example:

```
DCL PICTUREA PIC'(600)X',
    PICTUREB PIC'(255)9V(2)9';
```

Explanation: The maximum length of a character-string PICTURE variable is 511 characters. The maximum length of a numeric PICTURE variable is 256 characters including insertion characters.

IEL0374I I TOO MANY STATEMENTS IN THIS RECORD FOR CORRECT NUMBERING.

LINE CONTAINS MORE THAN 30 STATEMENTS. NUMBER OF ALL FOLLOWING STATEMENTS IN LINE SET TO CONSTANT VALUE.

Explanation: The constant value set for statements that cannot be individually numbered is N + 1.

IEL0375I E FACTOR NESTING LEVEL EXCEEDS MAXIMUM AFTER T. T IGNORED.

COMPILER RESTRICTION. MAXIMUM FACTOR DEPTH EXCEEDED AFTER T. T IGNORED.

Explanation: The depth of factorization used in this statement has exceeded the maximum allowed by the compiler.

IEL0376I I 'TASK' SPECIFIED. PROCEDURE ASSUMED REENTRANT.

'TASK' OPTION SPECIFIED. PROCEDURE ASSUMED TO BE REENTRANT.

Explanation: The compiler does not generate special code for the TASK option, but as tasking procedures must normally be reentrant, the REENTRANT option is assumed.

IEL0377I W BLANK ASSUMED [AFTER T].

NO BLANK BETWEEN KEYWORD AND FOLLOWING STRING. BLANK ASSUMED [AFTER T].

Example:

```
DO I = 1 TO'3';
```

IEL0378I I NON-INCREASING RECORD SEQUENCE NUMBER FOLLOWS.

NON-INCREASING RECORD SEQUENCE NUMBER FOLLOWS THIS STATEMENT. LINE NUMBERS MODIFIED.

Example:

```
     /* NUMBER OPTION REQUIRED IN THE PROCESS STATEMENT */
       TEST:  PROC OPTIONS(MAIN);        00020
       A = B;                            00030
                                         00050
100040       END TEST;                   00040
```

Explanation: The compiler checks the sequence number given in the sequence number field of each source statement record. If the number is equal to or less than the preceding number, the number in the sequence number field is increased by 100000 for the purposes of the number used for the GONUMBER compile-time option. The sequence number quoted in the message refers to the record in which the latest PL/I statement began. Thus, in the example above, although the message would refer to record number 4, the record number actually quoted in the message would be '2'.

IEL0380I S 'LIKE' IGNORED.

'LIKE' IS INVALID IN ENTRY PARAMETER DESCRIPTOR LIST AND IS IGNORED.

Example:

```
DCL TEST1 ENTRY (LIKE TEST) EXTERNAL;
```

IEL0381I E INVALID 'INITIAL' ATTRIBUTE [AFTER T].

INVALID SPECIFICATION OF 'INITIAL' ATTRIBUTE [AFTER T]. ATTRIBUTE IGNORED.

Example:

```
DCL A(4) FIXED INIT (+1,+2,+3,+X);
    -------------------------->|
                T
```

Explanation: Invalid syntax has been detected in the specification of a constant, expression, or function reference in the INITIAL attribute. Thus, in the above example, an invalid arithmetic constant X would be diagnosed.

IEL0382I E INVALID OPTION AFTER T1.

INVALID OPTION AFTER T1 IN RECORD I/O STATEMENT. OPTION T2 ASSUMED.

Example:

```
WRITE FILE (F) FROM (CARD) KEY (NUM);
                 -----------↑
                            T1
  T2 = KEYFROM
```

Explanation: An inappropriate KEY, KEYTO, or KEYFROM, option has been specified for this RECORD I/O statement.

IEL0383I W LINE NUMBER EXCEEDS N.

MAXIMUM LINE NUMBER EXCEEDED. LINE NUMBERS OF FOLLOWING RECORDS SET TO N.

Explanation: The NUMBER compile-time option has been specified, and the compiler has detected more than 1339 records specifying non-increasing sequence fields. Consequently, it has attempted to generate a line number greater than 134,000,000, which is the maximum possible value. The line number for each of the subsequent records will be set to this maximum value.

Programmer Response: Ensure that the source program has increasing sequence fields.

IEL0384I E ENVIRONMENT OPTION [AFTER T1] IS NOT SUPPORTED. T2 IGNORED.

ENVIRONMENT OPTION [AFTER T1] NOT SUPPORTED. T2 IGNORED.

Example:

```
DCL F FILE RECORD INPUT ENV(NCP);
```

Explanation: In the example shown, the NCP option is an environment option not supported by PL/I VSE.

Programmer Response: Remove the environment option denoted by T2 in the message, to avoid messages in subsequent compilations.

IEL0385I W N EXTRA 'END' STATEMENT(S) ASSUMED.

MULTIPLE CLOSURE OF BLOCK. N EXTRA 'END' STATEMENTS(S) ASSUMED.

Example:

```
A:PROC;
B:BEGIN;
C:DO;
/*PROCESSING*/
END A;
```

Explanation: END statements have been assumed for all open blocks and groups contained within the block being closed by the END statement referring to the label.

Programmer Response: Ensure that you have not unintentionally omitted any END statements.

IEL0386I S 'LEAVE' STATEMENT OUTSIDE 'DO' GROUP.

'LEAVE' STATEMENT NOT CONTAINED IN A 'DO' GROUP IN THE CURRENT BLOCK. STATEMENT REPLACED BY A 'NULL' STATEMENT.

Example:

```
P:PROC OPTIONS(MAIN);
LEAVE;
END P;
or
P:PROC OPTIONS(MAIN);
DO;
BEGIN;
LEAVE;
END P;
```

Explanation: The LEAVE statement must refer to a DO-group in the immediately enclosing block.

IEL0387I S NULL OR INVALID 'WHEN' EXPRESSION.

NULL OR INVALID 'WHEN' EXPRESSION. BIT CONSTANT OF LENGTH AND VALUE ONE ASSUMED.

IEL0388I E NON-EXECUTABLE UNIT FOLLOWING 'WHEN' OR 'OTHERWISE' CLAUSE.

NON-EXECUTABLE UNIT FOLLOWING 'WHEN' OR 'OTHERWISE' CLAUSE. UNIT ASSUMED TO BE IN A 'DO' GROUP.

Example:

```
SELECT(I);
WHEN(1) DCL J FIXED BIN;
END;
```

Explanation: The unit following a WHEN or OTHERWISE clause must be an executable unit.

IEL0389I S 'WHEN' OR 'OTHERWISE' CLAUSE APPEARS IN AN INVALID POSITION.

'WHEN' OR 'OTHERWISE' CLAUSE APPEARS IN AN INVALID POSITION. CLAUSE IGNORED.

IEL0390I E MORE THAN ONE EXECUTABLE UNIT SPECIFIED FOR 'WHEN' OR 'OTHERWISE' CLAUSE.

MORE THAN ONE EXECUTABLE UNIT FOLLOWING 'WHEN' OR 'OTHERWISE' CLAUSE. UNITS ASSUMED TO BE CONTAINED IN A 'DO' GROUP.

Example:

```
SELECT(I);
WHEN(1) J = K;
L = M;
END;
```

IEL0391I S 'ENTRY' STATEMENT SPECIFIED AS EXECUTABLE UNIT OF 'WHEN' OR 'OTHERWISE' CLAUSE.

'ENTRY' STATEMENT SPECIFIED AS EXECUTABLE UNIT OF 'WHEN' OR 'OTHERWISE' CLAUSE. STATEMENT IGNORED.

IEL0392I S 'SELECT' STATEMENT IS NOT FOLLOWED BY 'END' STATEMENT OR 'WHEN' OR 'OTHERWISE' CLAUSE.

'SELECT' STATEMENT IS NOT FOLLOWED BY 'END' STATEMENT OR 'WHEN' OR 'OTHERWISE' CLAUSE. STATEMENT IGNORED.

IEL0393I S 'PROCEDURE' STATEMENT SPECIFIED AS EXECUTABLE UNIT FOR 'WHEN' OR 'OTHERWISE' CLAUSE.

'PROCEDURE' STATEMENT SPECIFIED AS EXECUTABLE UNIT FOR 'WHEN' OR 'OTHERWISE' CLAUSE. KEYWORD 'PROCEDURE' REPLACED BY 'BEGIN'.

IEL0394I E LINE NUMBER EXCEEDS 33,000,000.

MAXIMUM LINE NUMBER EXCEEDED. LINE NUMBERS OF FOLLOWING RECORDS SET TO 33,000,000.

Explanation: The NUMBER option was specified with the COUNT or the FLOW option. With these option combinations, the maximum number of records specifying non-increasing sequence fields is 329. The compiler has detected more than this number of such records. Consequently, the compiler attempted to generate a line number greater than the maximum allowed for FLOW or COUNT (33,000,000). All line numbers for subsequent records are set to 33,000,000.

IEL0395I S INVALID CHARACTER AFTER T SET TO A BLANK.

Explanation: An invalid character was encountered. This character is set to a blank to allow the scan to continue. An invalid character is any character not in the PL/I character set [a double quote("), a cent sign (¢), or an exclamation point (!)]. This message might be a failure to supply the terminal quote mark for the string being scanned.

IEL0396I E AN UNSUBSCRIPTED QUALIFIED NAME IS NOT A VALID LABEL. T IGNORED.

Explanation: A label of the form "identifier.identifier:" has been found; such labels are invalid and ignored.

IEL0397I E REDUNDANT PARENTHESES IN DATA LIST IGNORED.

IEL0398I W EXTRANEOUS COMMA IGNORED.

EXTRANEOUS COMMA DETECTED AFTER 'T'. COMMA IGNORED.

Example:

```
DCL I,J,;
 ------↑
      T
```

IEL0399I E SEMICOLON ASSUMED [AFTER T].

Example:

```
IF X THEN GOTO Y ELSE;
          -----↑
               T
```

IEL0400I E RIGHT PARENTHESIS ASSUMED [AFTER T].

Example:

```
1. A = B + (C*D;
       ------↑
             T
2. DO WHILE (A>E;
       ------↑
             T
```

IEL0401I S MORE THAN N QUALIFICATIONS IN NAME BEGINNING T.

COMPILER RESTRICTION. MORE THAN N QUALIFICATIONS IN NAME BEGINNING T. EXCESS QUALIFICATIONS IGNORED.

Explanation: The compiler allows up to 15 levels of structuring.

IEL0402I E LABEL VALUE LIST IGNORED.

LABEL VALUE LIST INVALID FOR 'DEFAULT' STATEMENT. LIST IGNORED.

Example:

```
DEFAULT RANGE (L) LABEL (LAB1,LAB2);
This becomes:
DEFAULT RANGE (L) LABEL;
```

Explanation: A label value list cannot be used with the DEFAULT statement.

IEL0403I S QUALIFICATION OR SUBSCRIPT ON ENTRY PREFIX IGNORED.

COMPILER RESTRICTION. QUALIFIED OR SUBSCRIPTED ENTRY PREFIX ON 'PROCEDURE' OR 'ENTRY' STATEMENT. QUALIFICATION OR SUBSCRIPT IGNORED.

Explanation: The compiler does not allow initialization of aggregates of entry variables by the appearance of the subscripted or qualified entry variable name as a prefix to an ENTRY statement.

IEL0404I E ADJUSTABLE EXTENT INVALID IN 'RETURNS'.

ADJUSTABLE EXTENT INVALID IN 'RETURNS' SPECIFICATION. EXTENT IGNORED.

Example:

```
DCL X RETURNS(CHAR(Y));
```

IEL0405I E ARGUMENT SPECIFICATION T IGNORED.

INVALID ARGUMENT SPECIFICATION IN INTERLANGUAGE OPTION. T IGNORED.

Example:

```
DCL E ENTRY OPTIONS (COBOL NOMAP(FRED));
```

Explanation: The argument should be specified as ARGn where "n" is the number indicating the position in the argument list of the argument to which the interlanguage option is to apply.

IEL0406I E PARAMETER SPECIFICATION T IGNORED.

INVALID PARAMETER SPECIFICATION IN INTERLANGUAGE OPTION. T IGNORED.

Example:

```
E: ENTRY(X) OPTIONS(COBOL NOMAP(Y));
```

Explanation: The argument to the NOMAP, NOMAPIN, or NOMAPOUT options must be a parameter specified in the same PROCEDURE or ENTRY statement.

IEL0407I E INVALID OPTION T IGNORED.

INVALID OPTION T IGNORED.

Example:

```
DCL E ENTRY OPTIONS(MAIN);
```

IEL0408I E CONFLICTING 'OPTIONS' SPECIFICATION. T ASSUMED.

CONFLICTING SPECIFICATION OF INTERLANGUAGE OPTIONS. T ASSUMED.

Example:

```
DCL SUB ENTRY OPTIONS(FORTRAN, COBOL);
```

Explanation: Conflicting interlanguage options have been found in an options list. The COBOL, FORTRAN, and ASSEMBLER options conflict with each other. The last of these to be specified is assumed.

IEL0409I E LENGTH OR PRECISION NOT IN 'VALUE' CLAUSE.

STRING OR AREA LENGTH OR PRECISION SPECIFICATION IN 'DEFAULT' STATEMENT IS NOT IN 'VALUE' CLAUSE. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DEFAULT RANGE(S) CHAR(3);
```

Explanation: String lengths and area sizes should be specified inside a VALUE clause.

IEL0410I E ATTRIBUTE T INVALID FOR 'DEFAULT'.

ATTRIBUTE T INVALID FOR 'DEFAULT' STATEMENT. ATTRIBUTE IGNORED.

Example:

```
DEFAULT RANGE(A) ENTRY
  (ENTRY ignored)

DEFAULT RANGE(B) UNBUFFERED
   (UNBUFFERED ignored)
```

Explanation: Neither the RETURNS, ENTRY, and LIKE attributes, nor file description attributes are allowed in a DEFAULT statement.

IEL0411I U ATTRIBUTE FACTORING LEVEL EXCEEDS N.

COMPILER RESTRICTION. ATTRIBUTE FACTORING LEVEL GREATER THAN N. PROCESSING TERMINATED.

Explanation: More than 15 levels of attribute factorization have been used.

Programmer Response: Expand the declaration containing the error into separate declarations.

IEL0412I S MORE THAN 64 PARAMETERS.

COMPILER RESTRICTION. MORE THAN 64 PARAMETERS. LIST TRUNCATED.

Explanation: More than 64 parameters have been declared in a PROCEDURE or ENTRY statement or in an ENTRY attribute.

IEL0413I E DECLARATION OF D IGNORED.

DECLARATION OF INTERNAL ENTRY NOT ALLOWED. DECLARATION OF D IGNORED.

Example:

```
A: PROC;
  DCL B ENTRY RETURNS (FIXED);
  B:  PROC ENTRY RETURNS(FIXED);

    ⋮
    END B;

⋮
 END A;
```

Explanation: An internal entry point is declared according to its PROCEDURE or ENTRY statement. It cannot be declared in the invoking block in a DECLARE statement.

IEL0414I S NESTED 'LIKE' ATTRIBUTE IN DECLARATION OF D.

'LIKE' ATTRIBUTE IN DECLARATION OF D REFERENCES STRUCTURE WHICH CONTAINS 'LIKE'. EXPANSION TRUNCATED AT LATTER 'LIKE'.

Example:

```
    DCL 1 A, 2 B, 2 C LIKE A, 2 D, 3 B;
This becomes:
    DCL 1 A, 2 B, 2 C, 3 B, 3 C,
    (expansion truncated here)
    2 D, 3 B;
```

IEL0415I S 'LIKE' REFERENCE FOR D IS NOT A STRUCTURE.

'LIKE' REFERENCE IN DECLARATION OF D NOT A STRUCTURE. 'LIKE' ATTRIBUTE IGNORED.

Example:

```
DCL A, 1 B LIKE A;
```

IEL0416I S 'LIKE' REFERENCE FOR D IS AMBIGUOUS.

AMBIGUOUS 'LIKE' REFERENCE IN DECLARATION OF D. UNDEFINED SELECTION OF POSSIBILITIES MADE.

Example:

```
DCL 1 A, 2 B, 3 C, 4 D, 2 E, 3 C;
DCL 1 X LIKE A.C;
```

Explanation: An ambiguity has arisen through an incomplete qualification, and an undefined selection of one of the possible resolutions is made.

IEL0417I S 'LIKE' ATTRIBUTE FOR D REFERS TO INVALID STRUCTURE.

'LIKE' ATTRIBUTE IN DECLARATION OF D REFERENCES STRUCTURE WHICH IS UNDECLARED OR CONTAINS 'LIKE' ATTRIBUTE. FORMER 'LIKE' ATTRIBUTE IGNORED.

Example:

1. `X: PROC; DCL 1 A LIKE B; END;`
2. `DCL 1D, 2E LIKE F;DCL 1F,2 G LIKE H;`

IEL0418I U TOO MANY 'DEFAULT' SPECIFICATIONS AND 'LIKE' ATTRIBUTES.

COMPILER RESTRICTION. TOO MANY DEFAULT SPECIFICATIONS AND 'LIKE' ATTRIBUTES IN ONE BLOCK. PROCESSING TERMINATED.

Explanation: Details of default specification within the current scope, and LIKE attributes not yet resolved for the current blocks, are held in a directory. The total number of default specifications and unresolved LIKE attributes that can be handled depends on the environment in which the compiler is working; however, the directory should hold a minimum of 125 entries.

Programmer Response: Reduce the number of active default specifications and unresolved LIKE declarations to about 100 by expanding LIKE declarations and merging defaults.

IEL0419I S INVALID ATTRIBUTE SPECIFICATION IN 'VALUE' CLAUSE.

CONFLICTING OR REPEATED OR INVALID ATTRIBUTE SPECIFICATION IN 'VALUE' CLAUSE. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DEFAULT RANGE(*) VALUE (FIXED
 CHAR(1), (BIN(17), FLOAT(3))DEC);
```

Explanation: When an illegal combination of attributes appears (after any defactoring of attributes has been preformed) the combination has no effect. Individual attributes can still appear, however, and have effect in other combinations. In the above example, the attribute combinations FIXED CHAR(1) and DEC BIN(17) will be ignored, whereas the combination DEC FLOAT(3) will be accepted.

IEL0420I E PRECISION OR EXTENT MISSING IN 'VALUE' CLAUSE.

PRECISION OR EXTENT SPECIFICATION MISSING FOR ATTRIBUTE IN 'VALUE' CLAUSE. ATTRIBUTE IGNORED.

Example:

```
DEFAULT RANGE(I) VALUE (CHAR,FIXED BIN);
```

Explanation: The precision or extent specification must be included in an attribute specification in a VALUE clause.

IEL0421I S MULTIPLE DECLARATION OF D.

MULTIPLE DECLARATION OF D IN SAME STRUCTURE.

Example:

```
DCL 1 A, 2 B, 2 C, 2 B;
```

Explanation: For fully qualified references to the multiply-defined structure number, the last declaration will be taken. Incompletely qualified references will be further diagnosed as being ambiguous.

IEL0422I S MULTIPLE DECLARATION OF D IGNORED.

MULTIPLE DECLARATION OF D. DECLARATION IGNORED.

Example:

```
1. DCL A, A;
2. DCL B;
   DCL B;
```

Explanation: For a multiply-declared item, all declarations but one are ignored.

IEL0423I S MAJOR STRUCTURE LEVEL NUMBER ASSUMED TO BE 1.

MAJOR STRUCTURE LEVEL NUMBER NOT ONE. NUMBER REPLACED BY ONE.

Example:

```
DCL 2 G, 3 H;
```

IEL0424I S LOGICAL LEVEL NUMBER OF MEMBER REDUCED TO N.

COMPILER RESTRICTION. LOGICAL LEVEL NUMBER OF STRUCTURE MEMBER TOO LARGE. REDUCED TO N.

Example:

```
DCL 1 A, 2 B, 3 C, 4 D, 5 E, 6 F, 7 G, 8 H, 9 J,
    10 K, 11 L, 12 M, 13 N, 14 O, 15 P, 16 Q;
```

IEL0425I S DECLARED LEVEL NUMBER OF MEMBER REDUCED TO N.

COMPILER RESTRICTION. DECLARED LEVEL NUMBER OF STRUCTURE MEMBER TOO LARGE. REDUCED TO N.

Example:

```
DCL 1 A, 300 B;
```

IEL0426I E INVALID REPETITION OF T.

INVALID REPETITION OF ATTRIBUTE T. SECOND SPECIFICATION IGNORED.

Example:

```
DCL (X,Y) CHAR(1) CHAR(2);
                |-------|
                    T
```

IEL0427I E ATTRIBUTE T FOR D IGNORED.

ATTRIBUTE T IN DECLARATION OF D IGNORED.

Example:

```
DCL X FIXED FLOAT;
        |
        T
```

Explanation: A conflicting, invalid, or repeated attribute in a declaration will be ignored. The particular attribute that is ignored is the one that also conflicts with the declaration of the identifier after default attributes have been applied, or that is invalid or repeated.

IEL0428I E AMBIGUOUS 'DEFAULT' FOR D. T IGNORED.

AMBIGUOUS DEFAULT SPECIFICATION IN DECLARATION OF D. ATTRIBUTE T IGNORED.

Example:

```
DEFAULT RANGE(X) FLOAT;
DEFAULT RANGE(V:Z) FIXED;
DCL X;
```

IEL0429I E 'DEFAULT' AMBIGUOUS FOR RANGE T. T IGNORED.

'DEFAULT' SELECTION IS AMBIGUOUS FOR ANY CONTEXTUAL OR IMPLICIT DECLARATION IN RANGE T. ATTRIBUTE T IGNORED.

Example:

```
DEFAULT RANGE (J:R) FIXED;
DEFAULT RANGE (H:N) FLOAT;

An ambiguity exists for the range (J:N).
```

Explanation: The default ranges given in two or more range specifications should not overlap. The range given in the message is the extent of the ambiguous range. This message is produced even when there are no implicit declarations within the ambiguous range.

IEL0430I I NO 'MAIN' OPTION ON PROCEDURE.

NO 'MAIN' OPTION ON EXTERNAL PROCEDURE.

Example:

```
P:PROC;
```

Explanation: An external procedure without the main option cannot be run unless link-edited with another external procedure with the MAIN option.

IEL0431I S PICTURE CHARACTER AFTER T REPLACED BY T.

INVALID 'PICTURE' SPECIFICATION. CHARACTER [AFTER T] REPLACED BY T.

Example:

```
DCL P PIC'+99+';
is assumed to be:
DCL P PIC'+999';
```

Explanation: Depending on circumstances, an invalid picture specification character is replaced either by "9" when valid or by ".".

IEL0432I S SUBFIELD OF T HAS NO DIGIT POSITIONS.

SUBFIELD OF 'PICTURE' T HAS NO DIGIT POSITIONS. RESULTS OF EXECUTION ARE UNDEFINED.

Example:

```
DCL P PIC'$CR';
```

IEL0433I S PRECISION OF SUBFIELD OF T EXCEEDS N.

COMPILER RESTRICTION. PRECISION OF SUBFIELD OF PICTURE T EXCEEDS N. PICTURE SPECIFICATION IGNORED.

Example:

1. `PIC '(16)9'`
2. `PIC '9E999'`

Explanation: The maximum precision of a numeric picture is 15 for the fraction and 2 for the exponent.

IEL0434I S T TRUNCATED AT INVALID 'F'.

PICTURE T TRUNCATED AT INVALID 'F' SPECIFICATION.

Example:

```
DCL P PIC'9E9F(3)';
is assumed to be:
DCL P PIC '9E9';
```

IEL0435I S INVALID PICTURE T.

INVALID PICTURE T. PICTURE TEXT FOLLOWING 'F' SPECIFICATION IGNORED.

Example:

```
DCL P PIC '99V9F(-3)9';
is assumed to be:
DCL P PIC '99V9 F(-3)';
```

IEL0436I E INVALID PICTURE. T REPLACED BY 'X'.

INVALID CHARACTER PICTURE SPECIFICATION. T REPLACED BY 'X'.

Example:

```
PIC '9XR'
is assumed to be:
PIC '9XX'
```

IEL0437I E PRECISION OF D REDUCED TO N.

COMPILER RESTRICTION. PRECISION OF D TOO LONG. N ASSUMED FOR PRECISION.

Example:

```
DCL B BINARY (32,0),
    D DEC(17);
```

Explanation: The maximum precisions for arithmetic data types are given in the language reference manual for this compiler.

IEL0438I E INVALID 'RANGE' T IGNORED.

INVALID 'RANGE' SPECIFICATION T. SPECIFICATION IGNORED.

Example:

```
DEFAULT RANGE (C:B) BIN;
```

IEL0439I E ZERO VALUE ASSUMED FOR SCALE FACTOR.

COMPILER RESTRICTION. SCALE FACTOR IS OUTSIDE VALID RANGE. ZERO VALUE ASSUMED.

Example:

```
DCL F FIXED (6,-200);
```

IEL0440I E T WITHIN 'RETURNS' IGNORED.

COMPILER RESTRICTION. ATTRIBUTE T INVALID IN 'RETURNS' SPECIFICATION. ATTRIBUTE IGNORED.

Example:

```
1. DCL X RETURNS(RETURNS(FIXED));
2. DCL Y RETURNS(ENTRY);
```

Explanation: A function procedure cannot return a value that is an entry name.

IEL0443I S LOWER BOUND OF D GREATER THAN HIGHER BOUND.

LOWER BOUND GREATER THAN HIGHER BOUND IN DECLARATION OF D. BOUNDS INTERCHANGED.

Example:

```
DCL A(5:2);
```

Explanation: The lower bound of an array dimension must be declared to be numerically lower than the higher bound.

IEL0444I S D HAS MORE THAN N DIMENSIONS.

COMPILER RESTRICTION. D DECLARED WITH NUMBER OF DIMENSIONS GREATER THAN N. NUMBER OF DIMENSIONS REDUCED.

Example:

```
DCL A(1,2,3,4,5,6,7,8,9,10,11,12,
      13,14,15,16);
```

Explanation: An array cannot be declared with more than 15 dimensions.

IEL0445I S T NOT AN ENTRY NAME. IGNORED.

T IN 'GENERIC' SPECIFICATION IS NOT AN ENTRY NAME AND IS IGNORED.

Example:

```
DCL E1 ENTRY;
DCL E2 FILE VARIABLE;
DCL F  GENERIC
(E1 WHEN (FIXED), E2 WHEN(FLOAT));
```

Explanation: Only names of entry points can begin in the declaration of a GENERIC entry name.

IEL0446I S REFERENCE TO T IS AMBIGUOUS.

REFERENCE TO T IN 'GENERIC' SPECIFICATION IS AMBIGUOUS. UNDEFINED SELECTION MADE.

Example:

```
DCL F GENERIC
     (E1 WHEN(FIXED), E2 WHEN(FLOAT)),
E2 ENTRY,
1 X, 2 E1 ENTRY,
Y LIKE X;
```

Explanation: An entry expression in the declaration of a GENERIC entry name must be an unambiguous reference to an entry constant or variable.

IEL0447I E QUALIFICATION OF ATTRIBUTE T FOR D INVALID.

QUALIFICATION OF ATTRIBUTE T SPECIFIED FOR MEMBER D IN 'GENERIC' SPECIFICATION IS INVALID. QUALIFICATION IGNORED.

Example:

```
DCL E1 ENTRY;
DCL E2 ENTRY;
DCL G GENERIC (E1 WHEN (BIT),
    E2 WHEN(CHAR(3)));
```

Explanation: Details of the attributes allowed in a generic descriptor list are given in the language reference manual for this compiler.

IEL0448I S ATTRIBUTE T FOR D INVALID.

INVALID ATTRIBUTE T FOR MEMBER D IN 'GENERIC' SPECIFICATION. ATTRIBUTE IGNORED.

Example:

```
DCL E1 ENTRY;
DCL E2 ENTRY;
DCL F GENERIC
      (E1 WHEN (FIXED),
      E2 WHEN (FLOAT,BASED));
```

Explanation: Only the following attributes can be used in a generic descriptor: ALIGNED, AREA, base, BIT, CHARACTER, ENTRY, EVENT, FILE, LABEL, mode, OFFSET, PICTURE "picture specifications," POINTER, precision, scale, UNALIGNED, and VARYING. String lengths, area sizes, and label lists are not allowed.

IEL0449I S T CONFLICTS WITH PREVIOUS ATTRIBUTES FOR D.

ATTRIBUTE T CONFLICTS WITH PREVIOUS ATTRIBUTES OF MEMBER D IN 'GENERIC' SPECIFICATION. ATTRIBUTE IGNORED.

Example:

```
DCL E1 ENTRY;
DCL E2 ENTRY;
DCL F GENERIC
 (E1 WHEN(FIXED),E2 WHEN(FLOAT FIXED));
```

Explanation: When the attributes in a generic descriptor conflict, the second of the conflicting attributes is ignored.

IEL0450I E T IN VALUE LIST OF D NOT A LABEL CONSTANT.

T IN LABEL VALUE LIST OF LABEL VARIABLE D IS NOT A LABEL CONSTANT AND IS IGNORED.

Example:

```
   DCL L LABEL (L1,L2,L3);
L1: ;
L2: ;
   END L3 (is not a label)
```

Explanation: A label constant given in a label list should appear in the block within the scope of the label list.

IEL0451I S ADJUSTABLE EXTENTS FOR D INVALID WITH 'STATIC'.

ADJUSTABLE EXTENTS INVALID WITH 'STATIC' STORAGE CLASS IN DECLARATION OF D. N ASSUMED FOR EXTENT.

Example:

```
DCL A (4:N) STATIC;
DCL C CHAR(N) STATIC;
```

Explanation: Static variables cannot have an adjustable bound, extent, or length.

IEL0452I S ADJUSTABLE EXTENT INVALID FOR PARAMETER D.

ADJUSTABLE EXTENT INVALID WITH 'PARAMETER' STORAGE CLASS IN DECLARATION OF D. '*' ASSUMED FOR EXTENT.

Example:

```
X: PROC (P);
DCL P(Y); (becomes P(*))
```

Explanation: A parameter cannot have an adjustable bound, extent, or length, but it can assume that of its argument if specified as "*."

IEL0453I S ADJUSTABLE EXTENT INVALID FOR BASED D.

ADJUSTABLE EXTENT INVALID WITH 'BASED' STORAGE CLASS IN DECLARATION OF D. N ASSUMED FOR EXTENT.

Example:

1.
   ```
   DCL A(I:8) BASED;
     in this case I is assumed
     to be 1.
   ```

2.
   ```
   DCL B(4:J) BASED;
     in this case J is assumed
     to be 10.
   ```

Explanation: Unless the REFER option is specified, a based area cannot have an adjustable extent, and a based string cannot have an adjustable length. If specified, an adjustable lower bound is assumed to be 1, an adjustable upper bound is assumed to be 10, an adjustable string length is assumed to be 1, and an adjustable area extent is assumed to be 1000.

IEL0454I S '*' EXTENT INVALID FOR D NOT 'CONTROLLED' OR 'PARAMETER'.

'*' EXTENT SPECIFIED IN DECLARATION OF D BUT NOT 'CONTROLLED' OR 'PARAMETER'. N ASSUMED FOR EXTENT.

Example:

```
DCL A (*) STATIC;
```

Explanation: An "*" bound, extent, or length can only be used to declare an adjustable bound, extent, or length for a controlled variable or a parameter.

IEL0455I S 'REFER' EXTENT INVALID FOR NON-BASED D.

'REFER' EXTENT SPECIFIED IN DECLARATION OF D BUT NOT IN 'BASED' STRUCTURE. N ASSUMED FOR EXTENT.

Example:

1. `DCL 1 A, 2 B, 2 C(X REFER(B):8);`
2. `DCL 1 D, 2 E, 2 F(4:Y REFER (E));`

Explanation: The REFER option can only be used in the declaration of a based structure that contains an adjustable array dimension. If REFER is used in this way for the lower bound, 1 is assumed; if it is used for the upper bound, 10 is assumed.

IEL0456I S AMBIGUOUS 'REFER' ITEM T FOR D.

'REFER' ITEM T FOR EXTENT IN DECLARATION OF D IS AMBIGUOUS. UNDEFINED SELECTION MADE.

Example:

```
DCL 1 A BASED,
      2 B, 2 C, 3 B,
      3 D (X REFER B:10);

The reference B is ambiguous.
```

IEL0457I W 'REFER' T FOR D MAY BE INVALID.

IF THE STRUCTURE CONTAINS PADDING USE OF 'REFER' T FOR EXTENT OF D WILL BE INVALID AND RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL 1 A BASED,
     2 B FIXED BIN,
     2 C,
       3 D FIXED DEC,
       3 E (X REFER(D)) FLOAT DEC,
       3 F FIXED DEC;
```

Explanation: Although structure A contains padding, this message is pictured for structures that, when mapped, do not contain padding.

IEL0458I S 'REFER' T FOR D NOT PREVIOUS BASE ELEMENT.

'REFER' ITEM T FOR EXTENT IN DECLARATION OF D IS NOT A PREVIOUS SCALAR BASE ELEMENT IN THE SAME STRUCTURE. N ASSUMED FOR EXTENT.

Example:

```
 1. DCL 1 A BASED, 2 B(X REFER(C):10),
    2 C;
    -base element C follows the REFER
    item.

 2. DCL 1 A BASED, 2 B, 3 C(X REFER
    (B):10);
    -B is not a base element in
    structure A.

 3. DCL 1 A, 2 B, 3 C;
    DCL 1 D BASED, 2 E(X REFER(C):10);
    -C is not a base element in
    structure D.
```

IEL0459I I D TREATED AS NOT 'CONNECTED'.

ARRAY PARAMETER D TREATED AS NOT 'CONNECTED'. OPTIMIZATION MAY BE INHIBITED.

Example:

```
P: PROC(X,Y);
   DCL (X,Y CONNECTED, Z) (10,10);

 1. Z = X; (compiled as a do-group)
    Z = Y; (compiled as a single move
    instruction)

 2. V = X(6,3); (compiled as subscript
    calculation to obtain offset)
    V = Y(6,3); (offset is calculated
    at compile-time, no further
    calculation required)
```

Explanation: If the attribute CONNECTED is added to the declaration of the array, the subscript calculations will be optimized as shown.

IEL0460I E DEFAULT 'BUILTIN' OR 'GENERIC' FOR D IGNORED.

DEFAULT ATTRIBUTE 'BUILTIN' OR 'GENERIC' SPECIFIED FOR D CONFLICTS WITH USE OF IDENTIFIER IN IMPLICIT DECLARATION. ATTRIBUTE IGNORED.

Example:

```
DEFAULT RANGE (P) BUILTIN CHAR STATIC;
DCL A CHAR DEFINED (P);
```

IEL0461I S AGGREGATES INVALID IN GENERIC DESCRIPTOR LIST FOR T.

COMPILER RESTRICTION. AGGREGATES INVALID IN DESCRIPTOR LIST FOR MEMBER T IN 'GENERIC' SPECIFICATION. MEMBER IGNORED.

Example:

```
DCL (E1,E2,E3) ENTRY;
DCL G GENERIC (E1 WHEN(,(*),1),
    E2 WHEN (FIXED,FLOAT),
E3  WHEN (,1,2,2));

(E2 is a valid member, E1 and E3
are ignored)
```

IEL0462I S INITIALIZATION INVALID FOR STATIC LABEL D.

INITIALIZATION INVALID FOR 'STATIC' ENTRY VARIABLE D. INITIALIZATION IGNORED.

Example:

```
DCL EV ENTRY VARIABLE STATIC INIT(EV1);
```

IEL0463I S ENTRY NAME T INVALID IN 'GENERIC' SPECIFICATION.

COMPILER RESTRICTION. 'BASED' 'DEFINED' OR SUBSCRIPTED ENTRY NAME T INVALID IN 'GENERIC' SPECIFICATION. ENTRY NAME IGNORED.

Example:

```
DCL E1 ENTRY, E2 ENTRY BASED(P);
DCL G GENERIC (E1 WHEN (FIXED),
    E2 WHEN (FLOAT)); (E2 ignored)
```

IEL0464I S D IS NOT 'BASED'.

D IN 'LOCATE' STATEMENT NOT 'BASED'. STATEMENT IGNORED.

Example:

```
P: PROC;
   LOCATE FRED FILE(F);
   END P;
FRED (is not declared based)
```

IEL0465I S D IS NOT LEVEL ONE.

D IN 'ALLOCATE' STATEMENT NOT LEVEL ONE. THIS AND ANY FOLLOWING ITEMS IGNORED.

Example:

```
DCL (A,1 X) CTL, 2(Y,Z)
ALLOCATE A,Y,X;

(X and Z are ignored)
```

Explanation: A minor structure cannot be allocated independently of its containing level 1 structure.

IEL0466I S D IS NOT 'BASED' OR 'CONTROLLED'.

D IN 'ALLOCATE' STATEMENT NOT 'BASED' OR 'CONTROLLED'. THIS AND ANY FOLLOWING ITEMS IGNORED.

Example:

```
DCL X, (Y,Z) CTL;
ALLOCATE Y,X,Z;

(X and Z will not be allocated)
```

Explanation: Only based or controlled variables can be allocated storage by means of the ALLOCATE statement.

IEL0467I E FINAL MEMBERS MISSING FROM STRUCTURE.

FINAL MEMBERS MISSING FROM STRUCTURE SPECIFICATION IN 'ALLOCATE' STATEMENT. DECLARATION USED FOR MISSING MEMBERS.

Example:

```
DCL 1 X CTL, 2 (Y,Z) CHAR(3);
ALLOCATE 1 X, 2 Y CHAR(4);
```

Explanation: The member 2 Z is assumed to be included in the ALLOCATE statement with the declare attributes CHAR(3).

IEL0468I E LEVEL NUMBER PRECEDING D IGNORED.

Example:

```
DCL X CTL
ALLOCATE 1 X;

(the level '1' is ignored)
```

Explanation: A level number is only required in an ALLOCATE statement for a structure where members of that structure are specified explicitly in the statement.

IEL0469I E DIMENSIONS ATTRIBUTE MISSING FOR D.

DIMENSIONS ATTRIBUTE MISSING FOR STRUCTURE MEMBER D IN 'ALLOCATE' STATEMENT. DECLARED DIMENSIONS ASSUMED.

Example:

```
DCL 1 X CTL, 2 Y(10), 2 Z;
ALLOCATE 1 X,2 Y, 2 Z;
```

Explanation: Except for level 1 identifiers, those identifiers declared with dimensions must, when given in an ALLOCATE statement, be specified with dimensions.

IEL0470I S WRONG NUMBER OF DIMENSIONS FOR D.

WRONG NUMBER OF DIMENSIONS FOR D IN 'ALLOCATE' STATEMENT. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL X(10) CTL;
ALLOCATE X(5,2);
```

Explanation: An identifier declared with dimensions, when given in an ALLOCATE statement, must be specified with the same number of dimensions, although the bounds of a particular dimension can differ from those given in the declaration.

IEL0471I S CONFLICTING ATTRIBUTE T FOR D IGNORED.

CONFLICTING ATTRIBUTE T FOR D IN 'ALLOCATE' STATEMENT. ATTRIBUTE IGNORED.

Example:

```
DCL X CHAR(6) CTL;
ALLOCATE X BIT(6);
```

Explanation: The attribute of an identifier given in an ALLOCATE statement should not conflict with the attribute given in the declaration of the identifier. Note that string lengths and the upper and lower bounds of dimensions can differ between the declaration and the ALLOCATE statement.

IEL0472I E INVALID ATTRIBUTE T FOR D IGNORED.

INVALID ATTRIBUTE T FOR D IN 'ALLOCATE' STATEMENT. ATTRIBUTE IGNORED.

Explanation: Only the following attributes can be used in an ALLOCATE statement: BIT, CHARACTER, AREA, and INITIAL.

IEL0473I E LEVEL NUMBER FOR T REPLACED BY ONE.

INVALID LEVEL NUMBER SPECIFIED FOR T. LEVEL ONE ASSUMED.

Example:

```
ALLOCATE 2 X;
```

Explanation: The first identifier in an ALLOCATE statement must be a level 1 identifier.

IEL0474I E STRUCTURING ERROR FOLLOWING D.

ERROR IN SPECIFICATION OF STRUCTURING FOLLOWING D. DECLARED STRUCTURING ASSUMED FOR FINAL MEMBERS OF STRUCTURE.

Example:

```
DCL  1 A CTL, 2 B, 3 C CHAR(8);
ALLOCATE 1 A, 3 C CHAR(4);
```

(structure member B is assumed to be included in the ALLOCATE statement)

Explanation: If any members of a structure appear in an ALLOCATE statement, all the members of that structure must appear.

IEL0475I E ATTRIBUTES FOR 'BASED' VARIABLE D IGNORED.

ATTRIBUTES FOR BASED VARIABLE D INVALID ON 'ALLOCATE' STATEMENT. ATTRIBUTES IGNORED.

Example:

```
DCL X BASED (P);
ALLOCATE X INIT(3);
INIT(3) ignored
```

Explanation: Based variables cannot be given attributes when allocated.

IEL0476I E 'SET' OR 'IN' INVALID FOR 'CONTROLLED' D.

'SET' OR 'IN' OPTION INVALID IN 'ALLOCATE' STATEMENT FOR 'CONTROLLED' VARIABLE D. OPTION IGNORED.

Example:

```
DCL X CTL, Y BASED;
ALLOCATE X IN (A); (invalid)
ALLOCATE Y IN (A); (valid)
```

Explanation: The object of the SET or IN options must be a based variable.

IEL0477I E 'CHAR' 'BIT' OR 'AREA' WITHOUT EXTENT.

'CHARACTER' OR 'BIT' OR 'AREA' SPECIFIED WITHOUT EXTENT IN 'ALLOCATE' STATEMENT. ATTRIBUTE IGNORED.

Example:

```
DCL X CHAR(3) CTL;
ALLOCATE X CHAR;
'ALLOCATE X;' (assumed)
```

IEL0478I W D HAS STRING OVERLAY DEFINING.

D HAS STRING OVERLAY DEFINING AND MAY BE INCOMPATIBLE WITH THE PL/I F COMPILER.

Example:

```
DCL X(10) PICTURE '9999',
    A(10) PICTURE '9' DEFINED X;
```

Explanation: In the above example the F compiler would have given correspondence defining but the compiler will give string overlay defining.

IEL0479I S STRING OR AREA SIZE REDUCED TO N.

COMPILER RESTRICTION. CHAR OR BIT OR GRAPHIC OR AREA SIZE REDUCED TO COMPILER MAXIMUM.

Example:

```
     DCL A AREA(2234567890),
         B BIT (40000),
         C CHAR(40000);
```

Explanation: The maximum size allowed by this compiler is 2147483631 (2**31 - 1) for an area, 32767 for character and bit strings, and 16383 for graphic strings. Even so, these sizes might exceed the available main storage when the program is run.

IEL0480I S D DEFINED ON 'DEFINED' OR 'BASED'.

D IS DECLARED AS 'DEFINED' ON A BASE WHICH ALSO HAS THE 'DEFINED' OR 'BASED' ATTRIBUTE. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL A DEFINED B, B DEFINED C;
```

IEL0481I S D 'ISUB' 'DEFINED' ON CROSS-SECTION.

D IS DECLARED AS 'DEFINED' WITH AN 'ISUB' VARIABLE ON THE CROSS-SECTION OF A BASE. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL B(2,5), D(2,4) DEFINED B (*,1SUB);
```

IEL0482I S D 'DEFINED' WITH WRONG NUMBER OF SUBSCRIPTS.

D IS DECLARED AS 'DEFINED' WITH AN 'ISUB' VARIABLE ON A BASE WITH THE WRONG NUMBER OF SUBSCRIPTS. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL A(10) FIXED BIN(31),
    B BIT(32) DEFINED A(1SUB);
```

IEL0483I S D 'DEFINED' WITH 'ISUB' AND 'POSITION' ATTRIBUTE.

D IS DECLARED AS 'DEFINED ' WITH AN 'ISUB' VARIABLE AND HAS 'POSITION' ATTRIBUTE. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL B (10,10),
D (6) DEFINED B(1SUB,6) POS(3);
```

IEL0484I S MAPPING OF DEFINED ITEM D CONFLICTS WITH BASE.

MAPPING OF ELEMENT D OF ISUB-DEFINED ARRAY CONFLICTS WITH THAT OF BASE. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL 1 B(10), 2 C, 3 D;
DCL 1 X(5,2) DEFINED B(1SUB + 2SUB), 2 Y, 2 Z;
```

IEL0485I E CONFLICT BETWEEN DEFINED ITEM D AND BASE ATTRIBUTES IGNORED.

ATTRIBUTES OF ITEM D 'DEFINED' WITH AN 'ISUB' VARIABLE CONFLICT WITH THOSE OF BASE. CONFLICT IGNORED.

IEL0486I E SIMPLE DEFINING ASSUMED AS ATTRIBUTES OF D CONFLICT WITH BASE.

ATTRIBUTES OF 'DEFINED' ITEM D CONFLICT WITH THOSE OF BASE. SIMPLE DEFINING ASSUMED.

Example:

```
DCL B POINTER,
A FIXED BINARY(31,0) DEFINED B;
```

Explanation: Simple defining is assumed only if the two items have matching size, alignment, and dimensionality. String lengths or bounds are ignored.

IEL0487I S D 'DEFINED' ON UNCONNECTED AGGREGATE.

D IS STRING OVERLAY 'DEFINED' ON AN AGGREGATE WHICH IS NOT 'CONNECTED'. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL 1 B(10),
          2 C CHAR(2),
          2 F,
          A CHAR(20) DEFINED C;
```

Explanation: An aggregate used as the base in string overlay defining must occupy a contiguous area of storage.

IEL0488I S ATTRIBUTES OF 'DEFINED' ITEM D CONFLICT WITH BASE.

ATTRIBUTES OF 'DEFINED' ITEM D CONFLICT WITH THOSE OF BASE. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL A OFFSET DEFINED B,
    1 B, 2 (C,D) CHAR;
```

Explanation: The mapping of the defined and base items differ and the defined item is a level 1 offset.

IEL0489I S 'POSITION' VALUE FOR D LESS THAN ONE OR EXCEEDS N.

COMPILER RESTRICTION. D IS DECLARED WITH 'POSITION' VALUE LESS THAN ONE OR GREATER THAN N. 'POSITION' ATTRIBUTE IGNORED.

Example:

```
DCL B CHAR,
   A CHAR DEFINED B POS(-5);
```

IEL0490I E INVALID 'DEFINED' FOR D.

INVALID USE OF 'DEFINED' IN DECLARATION OF D. COMPILER WILL ATTEMPT TO ASSUME STRING OVERLAY DEFINING.

Example:

```
DCL B CHAR(5),
   D BIT(80) DEF(B);
```

Explanation: If the defined and base items do not match, both must be nonvarying, unaligned and either picture or character or both bit strings. If these rules are infringed, the defining will be accepted provided that the base item occupies contiguous storage.

IEL0491I S 'DEFINED' BASE FOR D IS AMBIGUOUS.

BASE REFERENCE OF 'DEFINED' ATTRIBUTE IN DECLARATION OF D IS AMBIGUOUS. UNDEFINED SELECTION MADE.

Example:

```
DCL 1 A, 2 B, 3 B,
      D DEFINED B;
```

(the identifier B is ambiguous)

IEL0492I S 'DEFINED' BASE FOR D IS NOT ACCEPTABLE.

D IS 'DEFINED' ON A BASE WHICH IS NOT ACCEPTABLE. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
1.     P: PROC;
       DCL X DEF P;
       END P;

2.     DCL B;
       DCL A DEF B(100);
```

IEL0493I W SIMPLE DEFINING APPLIES FOR D.

SIMPLE DEFINING APPLIES FOR D. IF OVERLAY DEFINING REQUIRED THEN ADD T TO DECLARATION.

Example:

```
DCL 1 A,
        2 B(10) CHAR(3),
        2 C(10) CHAR(2),
    1 D DEF A,
        2 E(5) CHAR(3),
        2 F(5) CHAR(2);
(simple defining will be used for structure D)
```

Explanation: The purpose of this message is to indicate a difference between this implementation and that of the PL/I D and F compilers which can result in the different mapping for the structure.

Programmer Response: The above action should be carried out if the program was originally written for the D or F compilers or if the program is to exchange records to and from D or F programs, when the records are derived from such structures and therefore require identical mapping.

IEL0494I E STRING OVERLAY DEFINING ASSUMED FOR D.

STRING LENGTH IN DEFINED ITEM D IS TOO LONG FOR SIMPLE DEFINING. STRING OVERLAY DEFINING ASSUMED.

Example:

```
DCL 1 A,
        2 B CHAR(1),
        2 C CHAR(79),
    1 D DEF A,
        2 E CHAR(40),
        2 F CHAR(40);
```

Explanation: Simple defining cannot be used where the length of the defined string is greater than the length of the base string. In the above example, string D.E is longer than its corresponding base string A.B.

IEL0495I E MAXIMUM LENGTHS OF DEFINED ITEM D AND BASE DIFFER.

AREA SIZE OR MAXIMUM LENGTH OF VARYING STRING IN SIMPLE DEFINED ITEM D DIFFERS FROM THAT OF THE CORRESPONDING BASE. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL 1 A, 2 B CHAR(3) VAR,
         2 C CHAR(4) VAR,
    1 D DEF A,
         2 E CHAR(2) VAR,
         2 F CHAR(3) VAR;
```

Explanation: If a defined item, for which simple defining is used, is a varying string that is shorter than the corresponding base string which is also varying, an error can occur during the run. A reference to the defined varying string can result in a string that is longer than its declared maximum length.

IEL0496I S T INVALID IN 'CALL' STATEMENT.

BUILTIN FUNCTION T INVALID IN 'CALL' STATEMENT. STATEMENT IGNORED.

Example:

```
P: PROC;
   CALL SIN(X);
   END;
```

IEL0497I S D INVALID IN 'FETCH' OR 'RELEASE'.

D IN 'FETCH' OR 'RELEASE' STATEMENT IS INVALID. STATEMENT IGNORED.

Example:

```
  MAIN: PROC OPTIONS(MAIN);
     DCL X ENTRY EXTERNAL;
     DCL PLIDUMP BUILTIN;

     FETCH X;        /* valid   */
     FETCH PLIDUMP;  /* invalid */
     FETCH INT;      /* invalid */

     INT: PROC; ... END;
```

Explanation: The identifier in a FETCH statement must be the name of an external PL/I procedure or a non-PL/I routine. Internal PL/I procedures cannot be obtained by a FETCH statement.

IEL0498I E INVALID SUBSCRIPTED PREFIX T.

SUBSCRIPTED STATEMENT PREFIX T IS NOT A NONSTATIC LABEL ARRAY. PREFIX IGNORED.

Example:

```
DCL LS(2) LABEL STATIC;
    LS(1):; (ignored)
DCL LA(3) LABEL AUTOMATIC;
    LA(2):; (accepted)
DCL L LABEL;
    L(3):; (ignored)
```

IEL0499I D INITIALIZED BY PREFIX AND DECLARATION.

LABEL VARIABLE D IS INITIALIZED BY STATEMENT PREFIX AND BY DECLARATION. DECLARED 'INITIAL' IGNORED.

Example:

```
DCL LV(3) LABEL INIT(L1,L2,L3);

:
   LV(1):L1:  X = Y/Z;
```

IEL0500I S CONFLICT IN USE OF D AS T.

CONFLICT BETWEEN USE OF D AS T AND ITS DECLARED ATTRIBUTES. STATEMENT IGNORED.

Example:

```
DCL P EVENT;
  :
CALL P;
```

Explanation: This message is produced when an identifier has an explicit declaration that conflicts with its use when the use would constitute a contextual declaration in the absence of the explicit declaration.

IEL0501I E D HAS INVALID ATTRIBUTES. OPTION IGNORED.

ATTRIBUTES FOR D INVALID IN 'ENVIRONMENT' OPTION. OPTION IGNORED.

Example:

```
DCL F FILE ENV(RECSIZE(X) PASSWORD(Y));
DCL (X,Y) FLOAT;
```

Explanation: The attributes for arguments in the ENVIRONMENT option are restricted. In the example, the arguments, X and Y, should be declared as follows:

Example:

```
DCL X FIXED BIN(31,0) STATIC;
DCL Y CHAR STATIC;
```

IEL0502I S USE OF D CONFLICTS WITH PREVIOUS DECLARATION.

USE OF D AS A STATEMENT LABEL PREFIX IS A CONFLICTING OR MULTIPLE DECLARATION. PREFIX IGNORED.

Example:

```
L1: X = 1;
  :
L1: A = B;
```

IEL0503I E T ASSUMED TO BE EXTERNAL ENTRY.

IDENTIFIER T IS NOT DECLARED. EXTERNAL ENTRY ASSUMED.

Example:

```
1.  P1:  PROC;
         CALL FRED;
         END;

2.  P2:  PROC;
         BERT = FRED(6);
         END;
```

Explanation: In the first example above, FRED is contextually declared BUILTIN. It is not however a recognized built-in function. In the second example, FRED is contextually declared BUILTIN in the absence of an explicit or default declaration as an array.

IEL0504I S T ASSUMED TO BE AN ARRAY.

IDENTIFIER T IN 'BUILTIN' CONTEXT IS INVALID. ASSUMED TO BE AN ARRAY.

Explanation: When a contextual declaration for an identifier as BUILTIN conflicts with a default declaration for the same identifier as an array, the contextual declaration is superseded by the default declaration.

IEL0505I S CONFLICT BETWEEN ATTRIBUTES OF D AND USE AS T.

CONFLICT BETWEEN DECLARED ATTRIBUTES OF D AND ITS USE AS T IN BOUNDS SPECIFICATION. BOUNDS OF N TO 10 ASSUMED.

Example:

```
DCL P, (P is float dec)
    X BASED,
    A (P-> X);
```

IEL0506I E CONFLICT BETWEEN ATTRIBUTES OF D AND USE AS T.

CONFLICT BETWEEN DECLARED ATTRIBUTES OF D AND ITS USE AS T IN LOCATOR QUALIFICATION. QUALIFICATION IGNORED.

Example:

```
DCL P FLOAT,
    A BASED (P);
```

IEL0507I S CONFLICT BETWEEN ATTRIBUTES OF D AND USE AS T.

CONFLICT BETWEEN DECLARED ATTRIBUTES OF D AND ITS USE AS T IN ADJUSTABLE STRING OR AREA SPECIFICATION. DEFAULT EXTENT ASSUMED.

Example:

```
DCL P DECIMAL,
    X BASED,
    A AREA (P-> X),
    B BIT (P-> X),
    C CHAR (P->X);
```

Explanation: The attributes assumed by default are AREA(1000), BIT (1), and CHAR(1).

IEL0508I S CONFLICT BETWEEN ATTRIBUTES OF D AND USE AS T.

CONFLICT BETWEEN DECLARED ATTRIBUTES OF D AND ITS USE AS T IN 'DEFINED' 'POSITION' OR 'INITIAL' ATTRIBUTE. ATTRIBUTE IGNORED.

Example:

```
DCL P DECIMAL,
    Q(10) DECIMAL,
    X BASED,
    A DEFINED (Q(P->X));
```

Explanation: Invalid INITIAL and POSITION attributes are ignored. The storage class AUTOMATIC is assumed for an invalid DEFINED attribute.

IEL0509I E CONFLICT BETWEEN ATTRIBUTES OF D AND USE AS T.

CONFLICT BETWEEN DECLARED ATTRIBUTES OF D AND ITS USE AS T. CONTEXTUAL ATTRIBUTES ASSUMED.

Example:

```
P: PROC (F);
   READ FILE (F) INTO (A);
```

Explanation: If an identifier is explicitly declared one way, but is used in another way, the identifier's attributes will be derived from how it is used, rather than how it was declared.

IEL0510I E CONFLICT BETWEEN ATTRIBUTES OF D AND USE AS T.

CONFLICT BETWEEN DECLARED ATTRIBUTES OF D AND ITS USE AS T IN 'SET' OR 'IN' OPTION. OPTION IGNORED.

Example:

```
DCL X BASED,
   (A,P) DECIMAL;
ALLOCATE X IN (A) SET (P);

('ALLOCATE X;' assumed)
```

IEL0511I S D INVALID IN TARGET POSITION.

D IS NOT A VARIABLE AND IS IN A TARGET POSITION. STATEMENT IGNORED.

Example:

```
P:  PROC;
    P = 1;
```

Explanation: A target position can be one of the following:

1. The left-hand side of an assignment statement
2. A DO-loop control variable
3. Data list in a GET statement
4. INTO option in a READ statement
5. SET option
6. KEYTO option in a READ statement
7. REPLY option

IEL0512I S T IS NOT DECLARED.

QUALIFIED NAME BEGINNING T IS NOT DECLARED. STATEMENT IGNORED.

Example:

```
P:  PROC;
    A.B = 1;
    END;
```

Explanation: Structures must be explicitly declared.

IEL0513I S INVALID USE OF D AS 'BUILTIN'.

D IS DECLARED BUILTIN BUT IS EITHER NOT A BUILTIN FUNCTION NAME OR IS INVALIDLY USED WITHOUT ARGUMENTS. STATEMENT IGNORED.

Example:

```
DCL E ENTRY VARIABLE,
    XYZ BUILTIN,
    SIN BUILTIN;

⋮
    E = XYZ;

⋮
    E = SIN;
```

Explanation: The identifier XYZ is not a built-in function. The built-in function SIN is used without an argument.

IEL0514I S D NOT LABEL KNOWN IN CURRENT BLOCK.

IDENTIFIER D AFTER 'GOTO' IS NOT A LABEL KNOWN IN THE CURRENT BLOCK. STATEMENT IGNORED.

Example:

```
P:  PROC;
    BEGIN;
     L: X = 1;
    END;
    GO TO P; (P is not known at this point)
    GOTO L; (L is not known at this point)
    END;
```

IEL0515I S INVALID USE OF D AS PSEUDO-VARIABLE.

INVALID USE OF D AS PSEUDO-VARIABLE. STATEMENT IGNORED.

Example:

```
  DCL ONCHAR BUILTIN;
  READ FILE (X) INTO (ONCHAR);
```

IEL0516I S D INVALID IN 'FROM' OPTION.

INVALID ITEM D IN 'FROM' OPTION. STATEMENT IGNORED.

Example:

```
WRITE FILE (FRED) FROM (FRED);
```

IEL0517I S D INVALID AS 'DO' CONTROL VARIABLE.

INVALID USE OF D AS CONTROL VARIABLE IN ITERATIVE SPECIFICATION. NONITERATIVE 'DO' ASSUMED.

Example:

```
 I: ; /* STATEMENT LABEL CONSTANT */
 DO I = 1 TO 10;
 END;
```

IEL0518I W T IS NOT IMPLICITLY 'BUILTIN'.

T IS THE NAME OF A BUILTIN FUNCTION BUT ITS IMPLICIT DECLARATION DOES NOT IMPLY 'BUILTIN'.

Example:

```
 X = DATE;
```

Explanation: A built-in function that does not require an argument must be declared BUILTIN. The declaration can be explicit, contextual, or implicit. (A contextual declaration is obtained by including a nonexecuting CALL statement for the built-in function name, and an implicit declaration is obtained by using the built-in function name with a null argument list.)

IEL0519I S IDENTIFIER BEGINNING T AMBIGUOUS.

IDENTIFIER BEGINNING T IS AN AMBIGUOUS REFERENCE TO A STRUCTURE MEMBER. UNDEFINED SELECTION MADE.

Example:

```
DCL 1 A, 2 B, 3 C, 2 D, 3 C;
   ⋮
A.C = 1;
```

Explanation: If a name is an incomplete qualification of more than one identifier, but does not completely qualify any identifier, it is in error.

IEL0520I S TOO MANY SUBSCRIPTS FOR D.

'ENTRY' VARIABLE 'A.B.C' HAS TOO MANY SUBSCRIPTS. STATEMENT IGNORED.

Example:

```
DCL 1 A(10), 2 B(3), 3 C ENTRY(FIXED,FLOAT);
   ⋮
X = B(9,2). C(5)(P);
```

Explanation: Subscripts in a qualified entry name must agree in number with the subscripts given in the declaration of the containing aggregate so that the argument list can be correctly distinguished.

IEL0521I S WRONG NUMBER OF ARGUMENTS FOR ENTRY.

WRONG NUMBER OF ARGUMENTS SPECIFIED IN REFERENCE TO ENTRY NAME. STATEMENT IGNORED.

Example:

```
P:  PROC(X);
END;
CALL P(Y,Z);
```

IEL0522I S INVALID 'GOTO' IN ITERATIVE 'DO' GROUP.

'GOTO' STATEMENT SPECIFIES INVALID BRANCH INTO AN ITERATIVE 'DO' GROUP. STATEMENT IGNORED.

Example:

```
P: PROC;
   DO I = 1 to 10;
   L: A = A + 1;
      END;
   GOTO L;
END P;
```

IEL0523I S INVALID 'GOTO' TO 'FORMAT' STATEMENT.

'GOTO' STATEMENT SPECIFIES INVALID BRANCH TO A FORMAT STATEMENT. STATEMENT IGNORED.

Example:

```
R: FORMAT (SKIP,COLUMN(2),A);
   GOTO R;
```

IEL0524I S AREA EXPRESSION SPECIFIED FOR RETURNED OFFSET.

COMPILER RESTRICTION. AREA SPECIFIED FOR OFFSET IN 'RETURNS' SPECIFICATION IS NOT A SIMPLE AREA NAME. AREA EXPRESSION IGNORED.

Example:

```
X:  ENTRY RETURNS(OFFSET(P->A));
  CALL X;
```

Explanation: An area expression in a RETURNS option must be a single identifier that is an area name.

IEL0525I S INVALID 'INITIAL' ATTRIBUTE IGNORED.

INVALID INITIAL SPECIFICATION FOR SCALAR. 'INITIAL' ATTRIBUTE IGNORED.

Example:

```
DCL A INIT((10)0);
```

IEL0526I S PSEUDO-VARIABLE INVALID AS CONTROL VARIABLE.

SPECIFIC PSEUDO-VARIABLE NOT ALLOWED AS CONTROL VARIABLE IN ITERATIVE SPECIFICATION. NONITERATIVE 'DO' ASSUMED.

Example:

```
DO COMPLEX(A,B) = M TO N;
```

IEL0527I U STATEMENT TOO LARGE. COMPILATION TERMINATED IN PHASE P.

COMPILER RESTRICTION. STATEMENT TOO LARGE. COMPILATION TERMINATED IN PHASE P.

Explanation: The amount of main storage available for the compiler determines the maximum length of a source statement. If the storage exceeds the maximum available, the maximum possible statement length can be used. This message can be produced also by a statement containing many nonstatic arrays with the

INITIAL attribute, particularly if these arrays are controlled or are arrays of structures.

Programmer Response: Either increase the amount of main storage for the compiler by using the SIZE compile-time option, or divide the statement into smaller statements. If neither of the above apply, check that the statement does not contain an unmatched quote character or comment delimiter. If due to array initialization, attempt to separate some of the initialization code by means of dummy BEGIN blocks or by using separate ALLOCATE statements. If this fails, initialize the arrays by assignment. If the TOTAL option is in use and the program contains many record I/O statements close together, break up the sequence of these statements by inserting BEGIN...END around half of them.

IEL0528I S D INVALID AS REMOTE FORMAT ITEM.

D NOT VALID AS REMOTE FORMAT ITEM. STATEMENT IGNORED.

Example:

```
    DCL L(10) LABEL, X;
    PUT FILE(F) EDIT(X) (R(L(1))); (valid)
    PUT FILE(F) EDIT(X) (R(L1)); (valid)
    PUT FILE(F) EDIT(X) (R(X)); (invalid)
L1: FORMAT (F(5,2));
```

Explanation: This message is produced if the remote format item is neither a label on a FORMAT statement, nor a label variable, nor a function reference that returns a label.

IEL0529I S D IS NOT 'BASED' OR 'CONTROLLED'.

D IN 'FREE' STATEMENT NOT 'BASED' OR 'CONTROLLED'. STATEMENT IGNORED.

Example:

```
    DCL A;
    FREE A;
```

IEL0530I S INVALID USE OF 'STRING' PSEUDO-VARIABLE.

COMPILER RESTRICTION. INVALID USE OF 'STRING' PSEUDO-VARIABLE. STATEMENT IGNORED.

Example:

```
GET STRING(STRING(A)); (invalid)
PUT STRING(STRING(A)); (invalid)
DISPLAY (B) REPLY(STRING(A)); (invalid)
READ FILE(F) INTO(X) KEYTO(STRING(A)); (invalid)
STRING(A) = C; (valid)
```

Explanation: The STRING pseudovariable can only be used in an assignment statement.

IEL0531I S STRING LENGTH EXCEEDS N.

COMPILER RESTRICTION. STRING LENGTH EXCEEDS N. REPETITION FACTOR OF ONE ASSUMED.

Example:

```
  A = (32768)'A'; /* BECOMES 'A' */
  A = (16384)'AA'; /* BECOMES 'AA' */
```

Explanation: An attempt has been made to produce a character or bit string with a length exceeding 32767 or a graphic string with a length exceeding 16383, using a repetition factor. A repetition factor of one is assumed.

IEL0532I S D NOT LABEL CONSTANT KNOWN IN CURRENT BLOCK.

IDENTIFIER D AFTER 'LEAVE' IS NOT A LABEL CONSTANT KNOWN IN THE CURRENT BLOCK. LABEL IGNORED.

IEL0533I I NO 'DECLARE' STATEMENT(S) FOR D,D,D...

Explanation: Identifiers in the list D,D,D...have not been explicitly declared.

IEL0534I I NO 'DECLARE' STATEMENT(S) FOR PARAMETER(S) D.

IEL0537I S EXTERNAL ENVIRONMENT NAME T IS ONLY SUPPORTED FOR OPTIONS(ASSEMBLER) ENTRY CONSTANTS.

EXTERNAL ENVIRONMENT NAME T IS ONLY SUPPORTED FOR OPTIONS(ASSEMBLER) ENTRY CONSTANTS. ENVIRONMENT NAME IGNORED.

Example:

```
DCL SUM_IT ENTRY EXT('W23A44');                 /* invalid */
DCL STRTBL CHAR  EXT('S22Z53');                 /* invalid */
DCL ASM    ENTRY EXT('AXEZ11') OPTIONS(ASM);    /* valid   */
```

Explanation: The external environment name may only be specified for an entry constant. The entry constant must have the OPTIONS ASSEMBLER attribute. The external environment name may not be specified for an entry variable.

IEL0538I S CHAR OR BIT OR GRAPHIC OR AREA SIZE INVALID.

CHAR OR BIT OR GRAPHIC OR AREA SIZE SPECIFIED AS NEGATIVE. ZERO IS ASSUMED.

Example:

```
DCL A CHAR(-4);
```

Explanation: A character, bit, or graphic string has a negative length specified. Zero is assumed.

IEL0539I E T IS NOT A BUILTIN FUNCTION NAME.

T EXPLICITLY DECLARED BUILTIN, BUT IS NOT A BUILTIN FUNCTION NAME. DECLARATION DELETED.

IEL0540I W EXTENDED FLOAT ARITHMETIC WILL BE USED.

EXTENDED FLOAT ARITHMETIC WILL BE USED IN THIS PROGRAM BECAUSE IT CONTAINS ITEMS WITH EXTENDED PRECISION.

Explanation: The message is given as a warning that expressions can be evaluated using extended precision even though they do not contain variables declared with extended precision. The same expressions would be evaluated using long float precision if no variables in the source program were declared using extended precision. Although the use of long float can mean loss of precision, it avoids the performance degradation of using extended float.

IEL0541I I 'ORDER' MAY INHIBIT OPTIMIZATION.

'ORDER' OPTION APPLIES TO THIS BLOCK. OPTIMIZATION MAY BE INHIBITED.

Example:

```
P: PROC;
   A: PROC REORDER;
      B: PROC;
         END;
      END;
         C: PROC;
            D: PROC ORDER;
               E: PROC;
                  END;
               END;
            END;
END;
```

Explanation: The message is produced for procedures P, C, D, and E. Procedure P has the ORDER option by default; procedure C inherits the ORDER option from procedure P; procedure D has the ORDER option declared explicitly; and procedure E inherits the ORDER option from procedure D. Procedure A has the REORDER option declared explicitly, and procedure B inherits the REORDER option from procedure A. This message is produced only when the OPT(TIME) option is specified for the compilation of blocks to which the ORDER option applies.

IEL0542I S AREA SPECIFIED FOR OFFSET IN ENTRY DECLARATION.

COMPILER RESTRICTION. AREA SPECIFIED FOR OFFSET IN 'ENTRY' DECLARATION IS IGNORED.

Example:

```
DCL E ENTRY (OFFSET(A));
```

is assumed to be:

```
DCL E ENTRY (OFFSET);
```

IEL0543I S STRUCTURE TERMINATED AFTER N MEMBERS.

COMPILER RESTRICTION. STRUCTURE TERMINATED AFTER N ITEMS.

Explanation: The structure has too many separately identifiable items. (Items include all minor structures and elements.)

IEL0544I W 'BUILTIN' SUBROUTINE WILL NOT BE USED FOR D.

D DECLARED AS EXTERNAL ENTRY REQUIRES PROVISION OF SUBROUTINE BY USER PROGRAM. 'BUILTIN' SUBROUTINE WILL NOT BE USED.

Example:

```
    DCL PLIDUMP ENTRY;
    CALL PLIDUMP ('HB','P');
```

Explanation: Built-in subroutines such as PLIDUMP are contextually declared to be built-in by their appearance in a CALL statement.

IEL0545I W 'ASSEMBLER' OPTION INVALID.

USE OF 'ASSEMBLER' OPTION INVALID ON 'PROCEDURE' OR 'ENTRY' STATEMENT. OPTION IGNORED.

Example:

```
P: PROC OPTIONS(ASSEMBLER);
```

Explanation: The ASSEMBLER option is valid only in an ENTRY declaration.

IEL0548I W PARAMETER TO MAIN PROCEDURE NOT VARYING CHARACTER STRING. 'NOEXECOPS' HAS BEEN ASSUMED.

PARAMETER TO PRIMARY ENTRY POINT OF MAIN PROCEDURE IS NOT VARYING CHARACTER STRING. 'NOEXECOPS' HAS BEEN ASSUMED.

Example:

```
P: PROC(X) OPTIONS(MAIN);
      DCL X FLOAT;
```

Explanation: OS passes arguments in the form of PL/I varying character strings, which comprise a 2-byte length field followed by the string data. If the parameter to the main procedure does not have the attributes

'VARYING CHARACTER', then at execution, NOEXECOPS is defaulted, the argument is passed as is, and run-time options, if passed, are ignored. One result of this is that PLITEST cannot be invoked by passing a run-time option and therefore may not get control.

IEL0549I E CONFLICT IN USE OF D AS T.

CONFLICT BETWEEN USE OF D AS T AND ITS DECLARED ATTRIBUTES. BIT VALUE ONE ASSUMED IN WHEN CLAUSE.

Example:

```
DCL E EVENT;
SELECT (CODE);
WHEN (E->B);
OTHERWISE;
END;
```

Explanation: In example identifier E is explicitly declared with the attribute EVENT. Its contextual use as a pointer (E->B) conflicts with the explicitly declared attribute EVENT.

IEL0550I E INVALID PREFIX(ES) SPECIFIED ON 'WHEN' OR 'OTHERWISE' CLAUSE.

INVALID PREFIX(ES) SPECIFIED ON 'WHEN' OR 'OTHERWISE' CLAUSE. PREFIX(ES) IGNORED.

Example:

```
SELECT(I);
LAB1: WHEN(A);
END;
```

or

Example:

```
SELECT(I);
WHEN(A);
(ZERODIVIDE): OTHERWISE;
END;
```

IEL0551I S NULL OR INVALID 'SELECT' EXPRESSION.

NULL OR INVALID 'SELECT' EXPRESSION. EXPRESSION IGNORED.

IEL0552I S DUPLICATE INITIALIZATION OF ELEMENT OF LABEL ARRAY D.

SUBSCRIPTED STATEMENT PREFIX SPECIFIES A DUPLICATE INITIALIZATION OF AN ELEMENT OF LABEL ARRAY T. PREFIX IGNORED.

Example:

```
DCL  L(10) LABEL;
  :
L(1): X = Y;
  :
L(1): A = B;
```

Explanation: The second appearance of L(1) is in error.

IEL0553I U END OF SOURCE TEXT IN DOUBLE-BYTE CHARACTER STRING OR STATEMENT TOO LONG AND CONTAINS UNMATCHED QUOTE.

END OF SOURCE TEXT FOUND IN DOUBLE-BYTE CHARACTER STRING OR STATEMENT LENGTH TOO LONG AND STATEMENT CONTAINS UNMATCHED GRAPHIC QUOTE. COMPILATION TERMINATED.

Explanation: The compiler has reached the end of the source program or the maximum statement length and has not found an ending quotation mark. The compiler will terminate.

Programmer Response: Check whether there is a quotation mark missing or the source program is incomplete.

IEL0554I S CONVERSION OF GRAPHIC EXPRESSION INVALID.

CONVERSION OF GRAPHIC EXPRESSION INVALID. STATEMENT IGNORED.

Explanation: No conversions are made by the compiler for graphic data.

IEL0557I S DBCS IDENTIFIER '<kkkk>' CANNOT BE EXTERNAL

Example:

```
DCL <kkkk> CHAR(5) EXT;
```

Explanation: DBCS names (non-EBCDIC) can be used as internal names but not as external names.

IEL0558I E OPTIONS(BYVALUE) ASSUMED FOR SYSTEM(CICS).

OPTIONS(BYADDR) IS INVALID WITH SYSTEM(CICS) COMPILER OPTION. OPTIONS(BYVALUE) ASSUMED.

Example:

```
*PROCESS SYSTEM(CICS);
 T2: PROC(E,C) OPTIONS(BYADDR MAIN);  /* MSG IEL0558 */
     DCL (E,C) POINTER;
```

Explanation: A MAIN procedure newly compiled with PL/I VSE can only receive parameters BYVALUE if the SYSTEM(CICS) compiler option is in effect. Receiving parameters BYADDR is not permitted for programs newly compiled (or re-compiled) with PL/I VSE.

Programmer Response: Remove the OPTIONS(BYADDR) specification. Alternatively, change the SYSTEM compiler option to VSE.

IEL0559I E OPTIONS(BYADDR) ASSUMED FOR SYSTEM(DLI).

OPTIONS(BYVALUE) IS INVALID WITH SYSTEM(DLI) COMPILER OPTION. OPTIONS(BYADDR) ASSUMED.

Example:

```
*PROCESS SYSTEM(DLI);
 T3: PROC(E,C) OPTIONS(BYVALUE MAIN);  /* MSG IEL0559 */
     DCL (E,C) POINTER;
```

Explanation: A MAIN procedure newly compiled with PL/I VSE can only receive parameters BYADDR if the SYSTEM(DLI) or SYSTEM(DL1) compiler option is in effect. Receiving parameters BYVALUE is not permitted for programs newly compiled (or re-compiled) with PL/I VSE.

Programmer Response: Remove the OPTIONS(BYVALUE) specification. Alternatively, change the SYSTEM compiler option to VSE.

IEL0560I W EXTERNAL ENTRY NAME BEGINS 'IHE'.

EXTERNAL ENTRY NAME BEGINS 'IHE'. POSSIBLE PL/I F COMPILER BUILTIN SUBROUTINE.

Example:

```
CALL IHESRTA(A,B,C,D,E); /*SORT ROUTINE*/
```

Explanation: F compiler subroutines commence with the characters "IHE," and therefore it is likely that the program has not been correctly converted for use with the compiler.

IEL0561I I DUPLICATE D IN PARAMETER LIST.

D APPEARS MORE THAN ONCE IN THE PARAMETER LIST. ONLY THE FIRST OCCURRENCE IS USED.

Explanation: A parameter should not be specified more than once in a parameter list. The compiler ignores subsequent occurrences.

IEL0562I S MORE THAN N PARAMETERS SPECIFIED.

MORE THAN N PARAMETERS SPECIFIED FOR THE CURRENT BLOCK.

Explanation: The combined number of unique parameters specified in the procedure statement and all its entry statements cannot exceed 255.

Programmer Response: Either reduce the number of unique parameters or restructure the program by dividing the procedure into smaller ones with fewer parameters.

IEL0563I W STATEMENT NUMBER/LEVEL/NEST LISTING DETAILS MAY BE INCOMPLETE.

STATEMENT NUMBER/LEVEL/NEST DETAILS MAY BE INCOMPLETE IN SOURCE LISTING DUE TO PREVIOUSLY DETECTED INVALID SYNTAX.

Explanation: The compiler has noted an invalid syntax condition (message IEL0327I). During subsequent source analysis, several lines might be printed without statement number, level, or nest details. (Incomplete details might continue until the compiler encounters the next line containing a quotation mark.)

Programmer Response: Correct the syntax error noted by message IEL0327I.

IEL0564I E T CONFLICTS WITH T.

T OPTION CONFLICTS WITH THE T OPTION AND IS IGNORED.

Example:

```
GET STRING(S1) EDIT(S2) (A(10)) COPY;
```

Explanation: The COPY option can appear only in a GET FILE statement.

IEL0565I U TOO MANY 'DEFAULT' SPECIFICATIONS.

COMPILER RESTRICTION. TOO MANY 'DEFAULT' SPECIFICATIONS. PROCESSING TERMINATED.

Explanation: The maximum number of default specifications allowed in a PL/I program depends on the length of the sequence of letters specified in the (default) range and can vary from 12 to 112. This message is issued when this maximum number is exceeded.

Programmer Response: Reduce the number of default specifications or merge them together.

IEL0566I S '*' USED AS ARGUMENT TO D WITHOUT 'OPTIONAL' PARAMETER DESCRIPTOR ATTRIBUTE.

'*' USED AS ARGUMENT TO ENTRY D BUT 'OPTIONAL' PARAMETER DESCRIPTOR ATTRIBUTE IS MISSING. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL X ENTRY OPTIONS(ASSEMBLER);
CALL X(*);    /* invalid */

DCL Y ENTRY(OPTIONAL) OPTIONS(ASSEMBLER);
CALL Y(*);    /* valid   */
```

Explanation: The use of asterisk, '*', to specify an omitted argument is only permitted in a CALL statement if the OPTIONAL attribute has been specified in the associated parameter descriptor list of the OPTIONS(ASSEMBLER) ENTRY declaration.

Programmer Response: Add the missing OPTIONAL attribute or remove the use of asterisk from the CALL statement.

IEL0567I S 'OPTIONAL' IS ONLY SUPPORTED FOR OPTIONS(ASSEMBLER) ENTRIES.

'OPTIONAL' ATTRIBUTE IS ONLY SUPPORTED FOR OPTIONS(ASSEMBLER) ENTRIES. 'OPTIONAL' ATTRIBUTE IGNORED FOR ENTRY D.

Example:

```
DCL X ENTRY(OPTIONAL);                    /* invalid */

DCL Y ENTRY(POINTER OPTIONAL) OPTIONS(ASM); /* valid */

DCL Z ENTRY(* OPTIONAL) OPTIONS(ASM);       /* valid */
```

Explanation: The use of the OPTIONAL attribute requires that OPTIONS ASSEMBLER be specified.

The OPTIONAL attribute is only supported within the parameter descriptor list of OPTIONS ASSEMBLER entries declared using the ENTRY attribute. OPTIONAL is not allowed in the DEFAULT statement or as a generic-descriptor attribute.

IEL0569I W PARAMETER TO MAIN PROCEDURE CONFLICTS WITH THE SYSTEM COMPILER OPTION SPECIFICATION

Explanation: The format of the run-time parameters passed to the program do not match the format that was expected. The SYSTEM(...) compile option is most likely incompatible with the run-time system.

Note that MAIN procedures newly compiled by PL/I VSE now default/require:

OPTIONS(BYVALUE) if the SYSTEM(CICS) compiler option is in effect

OPTIONS(BYADDR) if the SYSTEM(DLI) or SYSTEM(DL1) compiler option is in effect

Compiler implementation note: this message is issued even if the parameter is contextually declared with the correct data attributes.

IEL0570I S 'BYVALUE' PARAMETER D MUST BE POINTER OR REAL FIXED BINARY(31,0).

'BYVALUE' PARAMETER D MUST BE EITHER SCALAR POINTER OR SCALAR REAL FIXED BINARY(31,0).

Example:

```
P: PROC(A,B,C,D) OPTIONS(BYVALUE);
   DCL A    FIXED BIN(31) COMPLEX;    /* invalid */
   DCL B(4) FIXED BIN(31);            /* invalid */
   DCL C    CHAR(4);                  /* invalid */
   DCL D    FLOAT;                    /* invalid */
```

Explanation:

Procedures specifying OPTIONS(BYVALUE) are only permitted to have parameters with either the POINTER or REAL FIXED BINARY(31,0) data types. Arrays and structures are not permitted.

Note that MAIN procedures newly compiled by PL/I VSE now default/require OPTIONS(BYVALUE), if the SYSTEM(CICS), SYSTEM(DLI), or SYSTEM(DL1) compiler option is in effect. Receiving parameters BYADDR is not permitted for programs newly compiled (or re-compiled) with PL/I VSE.

Compiler implementation note: this message is issued even if the parameter is contextually declared with the correct data attributes.

Programmer Response: Correct the data type attributes.

IEL0571I S 'BYVALUE' PARAMETER D MUST NOT BE CONTROLLED.

'BYVALUE' PARAMETER D MUST NOT HAVE CONTROLLED ATTRIBUTE.

Example:

```
P: PROC(K)   OPTIONS(BYVALUE);
   DCL K FIXED BIN(31) CONTROLLED;  /* invalid */
```

Explanation: Procedures specifying OPTIONS(BYVALUE) may not have parameters with the CONTROLLED attribute.

Programmer Response: Remove the CONTROLLED attribute.

IEL0572I S ARGUMENT(S) FOR 'BYVALUE' ENTRY D MUST BE POINTER OR REAL FIXED BINARY(31,0).

ARGUMENT(S) FOR OPTIONS(BYVALUE) ENTRY D MUST BE EITHER SCALAR POINTER OR SCALAR REAL FIXED BINARY(31,0).

Example:

```
DCL X ENTRY OPTIONS(BYVALUE);
DCL A    FIXED BIN(31) COMPLEX;
DCL B(4) FIXED BIN(31);
DCL C    CHAR(4);
DCL D    FLOAT;
CALL X(A,B,C,D);   /* all arguments are invalid */
```

Explanation: Calls to entries declared with OPTIONS(BYVALUE) must pass arguments that have the attributes of either POINTER or REAL FIXED BINARY(31,0). Arrays and structures are not allowed.

Programmer Response: Change the arguments so that they have allowable attributes.

IEL0573I S 'CONTROLLED' PARAMETER IS INVALID FOR 'BYVALUE' ENTRY D.

'CONTROLLED' PARAMETER DESCRIPTOR IS INVALID FOR OPTIONS(BYVALUE) ENTRY D.

Example:

```
DCL Y ENTRY(POINTER CONTROLLED) OPTIONS(BYVALUE);  /* invalid */
```

Explanation: OPTIONS(BYVALUE) ENTRY declarations must not have parameter descriptors that specify or default the CONTROLLED attribute.

Programmer Response: Remove the CONTROLLED attribute.

IEL0574I S 'OPTIONAL' AND 'BYVALUE' CONFLICT FOR ENTRY D.

'OPTIONAL' ATTRIBUTE AND 'BYVALUE' OPTION CONFLICT. 'OPTIONAL' ATTRIBUTE IS IGNORED FOR ENTRY D.

Example:

```
DCL X ENTRY(* OPTIONAL) OPTIONS(BYVALUE ASM);  /* invalid */
```

Explanation: A declared entry may not specify both OPTIONS(BYVALUE) and the OPTIONAL parameter descriptor.

Programmer Response: Remove one of the conflicting options.

IEL0575I S RETURNED VALUE FOR 'BYVALUE' PROCEDURE MUST BE POINTER OR REAL FIXED BINARY(31,0).

DATA TYPE OF RETURNED VALUE FOR OPTIONS(BYVALUE) PROCEDURE MUST BE EITHER POINTER OR REAL FIXED BINARY(31,0).

Example:

```
P: PROC  OPTIONS(BYVALUE) RETURNS(FLOAT);   /* invalid */
   DCL F FLOAT;
   RETURN(F);
```

Explanation: Procedures specifying OPTIONS(BYVALUE) are only permitted to return either the POINTER or the REAL FIXED BINARY(31,0) data type.

Programmer Response: Change the RETURNS data type to an allowable type. Alternatively, omit the RETURN statement that returns a value and invoke the procedure not as a function reference but with a CALL statement.

IEL0576I S PARAMETER(S) FOR 'BYVALUE' ENTRY D MUST BE POINTER OR REAL FIXED BINARY(31,0).

PARAMETER DESCRIPTOR(S) FOR OPTIONS(BYVALUE) ENTRY D MUST BE EITHER SCALAR POINTER OR SCALAR REAL FIXED BINARY(31,0).

Example:

```
DCL X ENTRY(FLOAT) OPTIONS(BYVALUE);  /* invalid */
DCL Y ENTRY(CHAR ) OPTIONS(BYVALUE);  /* invalid */
```

Explanation: OPTIONS(BYVALUE) ENTRY declarations must have parameter descriptors that specify or default to either POINTER or REAL FIXED BINARY(31,0). Arrays and structures are not allowed.

Programmer Response: Correct the data type attributes.

IEL0577I S RETURNED VALUE FOR 'BYVALUE' ENTRY D MUST BE POINTER OR REAL FIXED BINARY(31,0).

DATA TYPE OF RETURNED VALUE FOR OPTIONS(BYVALUE) ENTRY D MUST BE EITHER POINTER OR REAL FIXED BINARY(31,0).

Example:

```
DCL X ENTRY OPTIONS(BYVALUE) RETURNS(BIT(32));  /* invalid */
DCL B BIT(32);
B = X();
```

Explanation: Entries declared with OPTIONS(BYVALUE) are only permitted to return either the POINTER or the REAL FIXED BINARY(31,0) data type.

Programmer Response: Change the RETURNS data type to an allowable type.

IEL0578I S D HAS BOUND GREATER THAN 2147483647

COMPILER RESTRICTION. D DECLARED WITH ARRAY BOUND GREATER THAN 2147483647. 2147483647 ASSUMED FOR BOUND.

Explanation: When CMPAT(V2) is used, larger arrays are allowed. The upper bound cannot be larger than 2**31 - 1.

IEL0579I S D HAS BOUND LESS THAN -2147483648.

COMPILER RESTRICTION. D DECLARED WITH ARRAY BOUND LESS THAN -2147483648. -2147483648 ASSUMED FOR BOUND.

Explanation: With PL/I VSE, larger arrays are allowed. The lower bound cannot be less than -2**31.

IEL0580I E INVALID INITIALIZATION FOR 'STATIC' LABEL D.

INITIALIZATION INVALID FOR 'STATIC' LABEL VARIABLE D. INITIALIZATION ACCEPTED.

Example:

```
1.  DCL LV LABEL STATIC INIT(LAB);
    LAB: ;

2.  DCL L(10) LABEL STATIC;
    L(1): ;
```

Explanation: The compiler allows the illegal language shown above, but for the program to run successfully, the OPT(TIME) compiler option must be specified, and the number of elements in the array must not exceed 511.

IEL0581I S INVALID BIT AGGREGATE DEFINING IGNORED.

COMPILER RESTRICTION. INVALID USE OF 'DEFINED' FOR BIT AGGREGATE D. 'DEFINED' ATTRIBUTE IGNORED.

Example:

```
DCL 1 B1(10),
      2 B2 BIT(1),
      2 B3 BIT(1),
      2 B4 BIT(2);

1.  DCL 1 D11(10) DEF B1,
          2 D2 BIT(2),
          2 D3 BIT(3);
    (this declaration is valid)

2.  DCL 1 D12 DEF B1(2),
          2 D2 BIT(2),
          2 D3 BIT(3);
    (this declaration is invalid)

3.  DCL 1 D13 (10) DEF B1 POS(X),
          2 D2 BIT(1),
          2 D3 BIT(1);
    (this declaration is also invalid)
```

Explanation: Defining on a bit aggregate is not allowed by this compiler when either the defined item is subscripted or the expression in the POSITION attribute is not an integer constant.

IEL0582I E MORE THAN 64 PARAMETERS.

MORE THAN 64 PARAMETERS IN CALL OR FUNCTION STATEMENT.

Explanation: The compiler limits the number of parameters on a CALL or function reference to 64.

Programmer Response: If some (or all) of the parameters can be collected into a structure then the structure can be passed as a single parameter.

IEL0583I E THE NUMBER OF DIGITS IN THE 'X' OR 'GX' CONSTANT T IS INVALID. THE CONSTANT WILL BE PADDED WITH HEXADECIMAL ZEROES.

Example:

```
C = '123'X;
G = '438743'GX;
```

Explanation: The hexadecimal characters in the constant string was not a multiple of 2('X') or a multiple of 4('GX'). An X constant represents bytes of storage and, therefore, must contain two hex digits for each byte. Similarly, a GX constant represents pairs of bytes of storage.

IEL0584I S A CHARACTER IN THE 'X' OR 'GX' OR 'BX' CONSTANT T IS INVALID. THE CONSTANT IS IGNORED.

Example:

```
C = '6FG3'X;
```

Explanation: Characters within X, GX or BX constants must be digits (0-9) or hex characters (A-F).

IEL0585I S THE 'BX' CONSTANT T IS TOO LONG. IT IS IGNORED.

Explanation: The BX bit string constant is too long. Each hexadecimal digit will be converted to four (4) bits, so a BX constant of length 1024 is equivalent to a bit constant of length 4096.

Programmer Response: Use concatenation or replication to build the long string from several shorter ones.

IEL0586I S THE SOURCE RECORD CONTAINS AN INVALID USE OF A SHIFT-IN OR SHIFT-OUT. T IS IGNORED.

Example:

```
GG = <kk<kk>>;
```

Explanation: An input data record was received that did not use shift codes properly. The example shows "nested" DBCS characters which is not allowed. This message is also produced if a shift-in was encountered following an SBCS character.

IEL587I W THE SYNTAX T(1) HAS BEEN ACCEPTED FOR SOURCE COMPATIBILITY BUT HAS BEEN TREATED AS T(2).

Example:

```
DCL NSA FILE RECORD INPUT
    ENVIRONMENT (MEDIUM(SYS020),FBS);
```

Explanation: Option T1 has been accepted for source compatibility with other versions of PL/I, but has been treated as T2 by this compiler. In the above example ENVIRONMENT option FBS is accepted but treated as FB.

IEL0588I W THE SYNTAX T HAS BEEN ACCEPTED FOR SOURCE COMPATIBILITY BUT IS IGNORED.

Example:

```
DCL NSA FILE RECORD INPUT
    ENVIRONMENT (MEDIUM(SYS020),NOLABEL);
```

Explanation: The syntax T is not recognized by this compiler. It is accepted for source compatibility but is functionally ignored. In the above example the NOLABEL option is accepted but ignored by the compiler.

IEL0589I S A CHARACTER IN THE CHARACTER CONSTANT T IS INVALID. THE CONSTANT IS IGNORED.

Example:

```
C = 'A<kk>';
```

Explanation: The character string constant contains a DBCS data item having no SBCS equivalent. If non-EBCDIC DBCS characters are to be included in a string constant, the mixed string constant must be used.

Programmer Response: The example above can be corrected by writing C = 'A<kk>'M;.

IEL0590I S A BLANK REPLACED AN INVALID CHARACTER IN AN IDENTIFIER. T IS NOW T.

Explanation: The identifier contains an invalid DBCS character. The invalid character is replaced by a blank.

IEL0591I I THE 'NOEXECOPS' OPTION IS ONLY VALID FOR THE MAIN PROCEDURE. 'NO EXECOPS' IS IGNORED.

Explanation: NOEXECOPS cannot appear with a PROCEDURE statement that is internal to the program. It can appear only with the first PROCEDURE statement; the one that contains the MAIN suboption.

IEL0592I S THE SOURCE RECORD VIOLATES DOUBLE-BYTE CHARACTER CONTINUATION RULES. THE RECORD IS IGNORED.

Explanation: A shift-out was detected in the right-most statement position. DBCS continuation is not defined for this situation.

Programmer Response: Move the statement left or right if possible. If a constant is involved you might be able to break it into several parts and concatenate the parts.

IEL0593I S THE SOURCE RECORD ENDS IN DOUBLE-BYTE MODE. THE RECORD IS IGNORED.

Explanation: All PL/I source program records must end with either an SBCS character or a shift-in code.

IEL0594I S THE SHIFT CODES ARE MISSING IN THE GRAPHIC CONSTANT T. THE CONSTANT IS IGNORED.

Example:

```
G = 'kk'G;
```

Explanation: The shift codes were omitted from the graphic string constant.

Programmer Response: The example above can be corrected by writing G = '<kk>'G;

IEL0595I S THE NUMBER OF DIGITS IN THE GRAPHIC CONSTANT T IS INVALID. THE CONSTANT IS IGNORED.

Explanation: Graphic constants must contain pairs of bytes.

IEL0596I S A DOUBLE-BYTE ITEM OVERLAPS THE MARGINS. THE RECORD IS IGNORED.

Example:

```
DCL GG GRAPHIC(40) INIT('<jj...m m>'G);
                                |
                              margin
```

Explanation: The right margin terminates a statement between the two bytes of a double-byte character. The same thing can happen when the left margin splits a double-byte character.

IEL0597I W THE ENVIRONMENT OPTION 'INDEXED' IS ACCEPTED BUT TREATED AS 'VSAM'.

Explanation: ISAM files are not supported by PL/I VSE. The ENVIRONMENT option INDEXED has been accepted for source compatibility, but is treated as if the ENVIRONMENT option VSAM was coded.

IEL0599I W D IS NOT THE SAME AS THAT SPECIFIED OR IMPLIED BY THE OFFSET ATTRIBUTE.

D IS NOT THE SAME AS THAT SPECIFIED OR IMPLIED BY THE OFFSET ATTRIBUTE. THE RESULTS OF EXECUTION ARE UNDEFINED UNLESS THE FORMER IS CONTAINED IN OR CONTAINS THE LATTER.

Example:

```
DCL A AREA(30);
DCL D FIXED BINARY(31) BASED;
DCL 1 B BASED(P),
    2 OFF OFFSET(A),
    2 C AREA(10);
ALLOCATE B IN(A) SET(P);
ALLOCATE D IN(C) SET(OFF);
```

Explanation: A run error can occur if the OFFSET variable is not contained in an area as specified in the IN/SET option of the ALLOCATE statement. (See the rules for the ALLOCATE statement in the *PL/I VSE Language Reference.)*

IEL0600I S [PROLOGUE CODE.] LOCATOR QUALIFICATION OF BUILTIN FUNCTION T.

[PROLOGUE CODE.] LOCATOR QUALIFICATION OF BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL TIME BUILTIN;
 T = P->TIME;
```

Explanation: Locators can only qualify based variables. Built-in functions cannot be based.

IEL0601I S INVALIDLY DECLARED VARIABLE. STATEMENT IGNORED.

INVALID DECLARATION OF A VARIABLE USED IN THIS STATEMENT. STATEMENT IGNORED.

Example:

```
DCL X BASED (A.B);
  :
X = 1;
```

Explanation: A variable which has been incorrectly declared and for which a message will have been issued has been used elsewhere. The message is issued because the compiler was unable to complete the declaration of the variable.

IEL0602I S [PROLOGUE CODE.] LOCATOR QUALIFICATION OF NON-BASED D.

[PROLOGUE CODE.] LOCATOR QUALIFICATION OF NON-BASED VARIABLE D. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL P POINTER, B FIXED;
A = P -> B;
```

Explanation: Locators (pointers and offsets) can only qualify based variables.

IEL0603I S STRUCTURE D DEPENDS ON A VARIABLE WITHIN STRUCTURE.

MAJOR STRUCTURE D ALLOCATION DEPENDS ON A VARIABLE DEFINED WITHIN THE STRUCTURE. [RESULTS OF EXECUTION UNDEFINED.]

IEL0604I S [PROLOGUE CODE.] AGGREGATE D INVALID AS LOCATOR QUALIFIER.

[PROLOGUE CODE.] USE OF AGGREGATE D FOR LOCATOR QUALIFICATION IS INVALID.

STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL P(10) POINTER;
        P -> X = Y;
```

Explanation: A locator qualifier must be an element and cannot be an unsubscripted or unqualified reference to an aggregate containing locators.

IEL0605I S [PROLOGUE CODE.] LEVEL OF LOCATOR QUALIFICATION EXCEEDS N.

COMPILER RESTRICTION. [PROLOGUE CODE.] LOCATOR QUALIFICATION IS RECURSIVE OR NUMBER OF LEVELS EXCEEDS N. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL Q OFFSET(P),
    P AREA BASED (Q);
DCL X AREA;
Q->P = X;
```

IEL0606I S [PROLOGUE CODE.] NO LOCATOR QUALIFICATION FOR BASED VARIABLE D.

[PROLOGUE CODE.] BASED VARIABLE D IS REFERENCED WITHOUT LOCATOR QUALIFICATION. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL B BASED;
    A = B;
```

Explanation: A based variable declared without an implicit pointer qualifier must be referred to with an explicit pointer qualifier.

IEL0607I W [PROLOGUE CODE.] T INVALID AS LOCATOR QUALIFIER.

[PROLOGUE CODE.] INVALID USE OF BUILTIN FUNCTION T AS LOCATOR QUALIFIER. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Explanation: A built-in function cannot be used as a locator qualifier.

IEL0608I S [PROLOGUE CODE.] ENTRY D INVALID AS LOCATOR QUALIFIER.

[PROLOGUE CODE.] INVALID USE OF ENTRY D AS A LOCATOR QUALIFIER. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Explanation: An entry name cannot be used as a locator qualifier.

IEL0609I W [PROLOGUE CODE.] EXPRESSION INVALID AS ARGUMENT TO 'STRING'.

[PROLOGUE CODE.] INVALID USED OF EXPRESSION AS ARGUMENT TO 'STRING' BUILTIN FUNCTION. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
A = STRING(B + C);
```

Explanation: The argument to the STRING built-in function must be an expression representing string data.

IEL0610I S [PROLOGUE CODE.] INVALID ARGUMENT TO 'STRING'.

[PROLOGUE CODE.] ELEMENTS OF ARGUMENT TO 'STRING' BUILTIN FUNCTION MUST BE EITHER ALL CHARACTERS OR ALL BIT. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL 1 S,
      2 B BIT(1),
      2 C CHAR(6);
        A = STRING(S);
```

Explanation: The argument to the STRING built-in function must consist of string data that is either all BIT or all CHARACTER.

IEL0611I S [PROLOGUE CODE.] NO ARGUMENTS PASSED TO T.

[PROLOGUE CODE.] NO ARGUMENTS PASSED TO BUILTIN FUNCTION OR PSEUDO-VARIABLE T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

IEL0612I S INVALID ARGUMENT TO T.

EXPRESSION OR CONSTANT INVALID AS ARGUMENT TO PSEUDO-VARIABLE T. STATEMENT IGNORED.

Example:

```
SUBSTR (A + B,I,J) = C;
```

Explanation: The argument to the pseudovariable must be an element variable.

IEL0613I S DATA TYPE OF ARGUMENT D INVALID FOR T.

[PROLOGUE CODE.] DATA TYPE OF ARGUMENT D INVALID FOR BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULT OF PROLOGUE UNDEFINED.]

Example:

```
DCL E FIXED BINARY;
   I = STATUS (E);

(E should be an event variable.)
```

IEL0614I S [PROLOGUE CODE.] INCORRECT 'AREA' SPECIFIED FOR OFFSET D.

[PROLOGUE CODE.] INCORRECT 'AREA' SPECIFIED OR DECLARED FOR OFFSET D. STATEMENT IGNORED. [RESULT OF PROLOGUE UNDEFINED.]

Example:

```
DCL (A,B) AREA,
    C FIXED,
    S BASED(P),
    O OFFSET(C);
ALLOCATE S IN(A) SET(O);
```

IEL0615I W RESULTS MAY BE UNDEFINED IN USE OF 'REFER' VARIABLE D.

COMPILER RESTRICTION. RESULTS MAY BE UNDEFINED IF LOCATOR QUALIFIER FOR D OR 'REFER' EXTENTS CHANGED IN LOOP.

Example:

```
DCL 1 N BASED, 2 NO, 2 NV(I REFER(NO)), 2 NP;
Q = P(1);
DO I = 1 BY 1 WHILE (Q -> NP<4);
Q = P(I + 1);
END;
```

Explanation: Mapping to refer variables appearing in WHILE expressions is performed once only outside the loop so that the expression is reevaluated without taking account of any changes of generation or adjustability. If the generation to the refer variable is changed in the loop by an ALLOCATE or FREE statement, by an assignment to a locator qualifying the refer variable, or if the extents of the refer variable are changed in the loop, unexpected results might occur.

IEL0616I W VARIABLE IN 'INITIAL' FOR D MAY BE UNINITIALIZED.

INITIAL SPECIFICATION FOR VARIABLE D MAY CONTAIN AN UNINITIALIZED VARIABLE. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL M, N INIT(M);
```

Explanation: This is a possible error detected in compiling the prologue routine to the program block which contains the erroneous initial specification. Consequently, the statement number given in this message is that of the PROCEDURE or BEGIN statement for the block.

Programmer Response: The program might contain a preceding declaration which uses the INITIAL CALL form of the INITIAL attribute to invoke a procedure that assigns a value to the identifier used in the subsequent INITIAL specification. If so, this message can be ignored. Otherwise, the program should be modified to ensure that the identifier will be initialized before it is used in the INITIAL attribute.

IEL0617I S T NOT LEVEL ONE.

D IN 'FREE' STATEMENT NOT LEVEL ONE. STATEMENT IGNORED.

Example:

```
DCL 1 A BASED,
    2 B, 2 C;

:
FREE B;
```

Explanation: A free statement cannot be used to free storage occupied by a part of a based or controlled item.

IEL0618I S [PROLOGUE CODE.] 'DCL'OR 'DFT' STATEMENT CONTAINS INVALID EXPRESSION.

[PROLOGUE CODE] 'DECLARE' OR 'DEFAULT' STATEMENT CONTAINS AN INVALID EXPRESSION. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

IEL0619I S [PROLOGUE CODE.] CONSTANT ARGUMENT TO T.

[PROLOGUE CODE.] CONSTANT IS INVALID ARGUMENT TO BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
   DCL P POINTER;

⋮

   P = ADDR(27);

⋮

L: P = ADDR(L);
```

Explanation: A constant in PL/I is not considered to be associated with a particular location in storage. It cannot, therefore, have a storage address.

IEL0620I S [PROLOGUE CODE.] ARGUMENT N TO D IS NOT AN ARRAY.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D IS NOT AN ARRAY BUT THE CORRESPONDING PARAMETER HAS A '*' BOUND. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
   DCL J;
       CALL E(J);
   E: PROC(P);
   DCL P(*);
```

Explanation: A parameter with an adjustable (*) bound is assumed to be an array that obtains the value for the bound from the associated argument. Consequently, the argument must also be an array.

IEL0621I W [PROLOGUE CODE.] AGGREGATE ARGUMENT D INVALID FOR ELEMENT PARAMETER.

[PROLOGUE CODE.] PARAMETER CORRESPONDING TO AGGREGATE ARGUMENT D IS AN ELEMENT. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
1.  DCL E ENTRY(FLOAT),
         ARR(8) FLOAT;
    CALL E(ARR);

2.  DCL ARR(8) FLOAT;
         CALL E(ARR);
         E:  PROC (PARAM);
           DCL PARAM FLOAT;
```

Explanation: An aggregate argument cannot be passed to a parameter that is not an aggregate.

IEL0622I W RECORD VARIABLE D NOT 'CONNECTED'.

RECORD VARIABLE D IS NOT 'CONNECTED'. STATEMENT IGNORED.

Example:

```
DCL 1 A (4),
      2 B CHAR (3),
      2 C CHAR (7);
READ FILE(F) INTO (B);
```

Explanation: The INTO or FROM option of a record-oriented input/output statement must refer to an identifier that represents a contiguous area of storage.

IEL0623I S [PROLOGUE CODE.] ARGUMENT N TO D INVALID FOR 'CONTROLLED' PARAMETER.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D INVALID FOR 'CONTROLLED' PARAMETER. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
   DCL X(10);
   CALL E(X);
   E: PROC(C);
   DCL C(10) CTL;
```

Explanation: An argument corresponding to a controlled parameter must be a level 1 unsubscripted variable with the CONTROLLED attribute. Other attributes must also match those of the parameter so that the argument need not be converted and assigned to a temporary argument.

IEL0624I S [PROLOGUE CODE.] ARGUMENT N TO D HAS TOO MANY DIMENSIONS.

COMPILER RESTRICTION. [PROLOGUE CODE.]RESULT OF EXPRESSION IN ARGUMENT NUMBER N TO ENTRY D HAS TOO MANY DIMENSIONS. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL 1 A,
      2 B,
      2 C(2, 2, 2, 2, 2, 2, 2, 2, 2, 2);
CALL X(A + C);
```

Explanation: The expression (A+C) results in a temporary argument that is an array of structures, the first structure element having 10 dimensions, and the second having 20 dimensions. The maximum number of dimension allowed is 15. If an argument contains both an array and a structure and there is no parameter descriptor, the temporary argument is created in the form of an array of structures.

IEL0625I S [PROLOGUE CODE.]'*' USED AS ARGUMENT TO D.

[PROLOGUE CODE.]'*' USED AS ARGUMENT TO D. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
1. CALL X(*);
   X: PROC(N); ... END;

2. A = HBOUND (*,1);
```

Explanation: An asterisk, which can be used in a subscript list to indicate a cross-section of an array, is meaningless in an argument list. The error might have occurred because an array declaration has been omitted.

IEL0626I S [PROLOGUE CODE.] STRUCTURING OF D DOES NOT MATCH PARAMETER.

[PROLOGUE CODE.] STRUCTURING OF ARGUMENT D DOES NOT MATCH THAT OF THE PARAMETER. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL 1 S, 2 S1, 2 S2;
CALL P(S);
P: PROC(F);
DCL 1 F, 2 F1, 2 F2, 3 F3;
END P;
```

Explanation: A structure passed as an argument should match the corresponding parameter exactly. (However, a parameter that is a structure can correspond to an argument that is not a structure.)

IEL0627I S NUMBER OF DIMENSIONS IN ARGUMENT 'H' DOES NOT MATCH THAT OF PARAMETER.STATEMENT IGNORED.

[PROLOGUE CODE.] NUMBER OF DIMENSIONS IN ARGUMENT D DOES NOT MATCH THAT OF THE PARAMETER. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL H(10);
CALL Q(H);
Q:PROC(G);
DCL G(10,10);
```

Explanation: An array passed as an argument must match the corresponding array parameter for dimensions. (However, a parameter that is an array can correspond to an argument that is not an array.)

IEL0628I S [PROLOGUE CODE] BOUNDS OF D DO NOT MATCH PARAMETERS.

[PROLOGUE CODE] BOUNDS OF ARGUMENT D DO NOT MATCH THOSE OF PARAMETERS. STATEMENT IGNORED.[RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL S(4,12);
CALL P(S);
P: PROC(F);
DCL F(4,10);
```

Explanation: An argument with fixed bounds must match the corresponding parameter at all levels.

IEL0629I S [PROLOGUE CODE] USE OF CROSS-SECTION OF STRUCTURE D IS INVALID.

COMPILER RESTRICTION. [PROLOGUE CODE] USE OF CROSS-SECTION OF STRUCTURE D IS INVALID. STATEMENT IGNORED.[RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL 1 S(4,4), 2 S1, 2 S2;
CALL X(S(2,*));
```

Explanation: A cross-section of an array of structures cannot be given as an argument. The reference must be either fully subscripted with an asterisk for each dimension or unsubscripted.

IEL0630I S [PROLOGUE CODE] SUBSCRIPT CONTAINING D IS NOT AN ELEMENT.

SUBSCRIPT CONTAINING D IS NOT AN ELEMENT EXPRESSION. STATEMENT IGNORED.[RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL A(10,10);
A(2,A) = 1;
```

Explanation: An array subscript must be an expression that represents the value of a single integer.

IEL0631I S [PROLOGUE CODE.] WRONG NUMBER OF SUBSCRIPTS FOR D.

[PROLOGUE CODE.] WRONG NUMBER OF SUBSCRIPTS FOR D. [RESULTS OF PROLOGUE UNDEFINED.] [STATEMENT IGNORED.]

Example:

```
1. DCL A(5,5);
         X = A(2);

2. DCL A;
         X = A(2);
```

Explanation: A reference to an array must contain the same number of subscripts as given in its declaration.

IEL0632I S [PROLOGUE CODE.] STRUCTURE IS INVALID ARGUMENT TO T.

COMPILER RESTRICTION. [PROLOGUE CODE.] STRUCTURE IS INVALID ARGUMENT TO BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL 1 S,
      2 S1 CHAR,
      2 S2 CHAR(4);
S = SUBSTR(S,1,3);
```

Explanation: The only built-in functions that accept structures as arguments are ALLOCATION, ADDR, and STRING. All other operations on structures by built-in functions must be specified individually for each element.

IEL0633I S [PROLOGUE CODE.] EXPRESSION OR 'ISUB' ARRAY INVALID ARGUMENT TO T.

[PROLOGUE CODE.] EXPRESSION OR ISUB-DEFINED ARRAY IS INVALID ARGUMENT TO BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
I = ALLOCATION(A + B);
```

Explanation: Operational expressions are not allowed as arguments to the built-in functions ALLOCATION, ADDR, and STRING.

IEL0634I S [PROLOGUE CODE.] ELEMENT IS INVALID ARGUMENT TO T.

[PROLOGUE CODE.] ELEMENT IS INVALID ARGUMENT TO BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL X;
I = HBOUND(X,1);
```

Explanation: Array built-in functions cannot have element arguments.

IEL0635I E [PROLOGUE CODE.] NON-CONNECTED ARGUMENT TO 'ADDR' INVALID.

[PROLOGUE CODE.] NON-CONNECTED ARGUMENT TO BUILTIN FUNCTION 'ADDR' INVALID. ARGUMENT ACCEPTED.

Example:

```
DCL A(10,10), P POINTER;
P = ADDR(A(*,1));
```

Explanation: The argument to the built-in function ADDR occupies non-connected storage. The value returned by the function is the address of the first byte of the argument. Care must be exercised when using this pointer to refer to a based variable, because it is probable that the based variable will be mapped over storage occupied not only by the argument, but by some other variable as well.

IEL0636I S EXPRESSION OR 'ISUB' ARRAY IN GET/PUT DATA.

EXPRESSION OR ISUB-DEFINED ARRAY USED IN GET/PUT DATA. STATEMENT IGNORED.

Example:

```
1. DCL A(10), B(5) DEF A(2*1SUB);
   GET DATA(B);

2. DCL C(6) CHAR(8);
   PUT DATA(SUBSTR(C,3));
```

Explanation: PL/I does not allow expressions in GET DATA or PUT DATA statements, and the compiler does not implement the transmission of ISUB defined arrays by these statements.

IEL0637I S [PROLOGUE CODE.] SECOND ARGUMENT TO T IS AGGREGATE.

[PROLOGUE CODE.] SECOND ARGUMENT TO BUILTIN FUNCTION T IS AN AGGREGATE. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL ARR(10), T;
T = HBOUND(ARR,ARR);
```

Explanation: With the exception of the POLY built-in function, the array built-in functions that have two arguments must have an element expression as the second argument.

IEL0638I S [PROLOGUE CODE.] ARGUMENT N TO 'POLY' HAS MORE THAN ONE DIMENSION.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO BUILTIN FUNCTION 'POLY' HAS MORE THAN ONE DIMENSION. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL ARR(6,6);
X = POLY(ARR,X);
```

IEL0639I E [PROLOGUE CODE.] ARGUMENT TO 'ADDR' MAY HAVE WRONG ALIGNMENT.

[PROLOGUE CODE.] ARGUMENT TO BUILTIN FUNCTION 'ADDR' MAY HAVE WRONG ALIGNMENT. ARGUMENT ACCEPTED.

Example:

```
DCL 1 S UNALIGNED,
      2 T BIT(3),
      2 U BIT(8),
      2 P PTR;
P = ADDR(U);
```

Explanation: This implementation uses byte addresses for locator values and does not provide bit addressing mechanisms for them. Consequently, if the argument to the ADDR built-in function does not lie on a byte boundary, the address returned will be that of the containing byte.

IEL0640I W ARGUMENT N TO GENERIC ASSUMED TO MATCH AGGREGATE PARAMETER.

ARGUMENT NUMBER N TO GENERIC FUNCTION IS ASSUMED TO MATCH ITS CORRESPONDING AGGREGATE PARAMETER.

Example:

```
DCL G GENERIC
      (G1 WHEN(FIXED),
       G2 WHEN(FLOAT)),
    (G1,G2) ENTRY,
    ARR(10) FLOAT;
    CALL G(ARR);
```

Explanation: Matching of arguments and parameters is not performed on aggregate arguments to generic functions. Consequently, a mismatch will not be detected and a run-time error could result.

IEL0641I S NESTING OF FUNCTIONS EXCEEDS MAXIMUM.

COMPILER RESTRICTION. LEVEL OF NESTING OF FUNCTIONS EXCEEDS MAXIMUM. STATEMENT IGNORED.

Explanation: The nominal limit on the number of nested functions in a source module is 50. However, this limit can vary according to the length of the labels prefixed to the PROCEDURE statements. If the average length of the labels exceeds eight characters, the maximum number of nesting levels will be less than 50.

IEL0642I S ARRAY D IN ELEMENT ASSIGNMENT.

INVALID USE OF ARRAY D IN ELEMENT ASSIGNMENT. STATEMENT IGNORED.

Example:

```
DCL A(8,8);
I = A + J;
```

Explanation: An unsubscripted array reference cannot appear on the right-hand side of an assignment to an element variable.

IEL0643I S STRUCTURE D IN ARRAY OR ELEMENT ASSIGNMENT.

INVALID USE OF STRUCTURE D IN ARRAY OR ELEMENT ASSIGNMENT. STATEMENT IGNORED.

Example:

```
DCL 1 A, 2 B, 2 C;
DCL (X,Y)(5);
I = A + J; (invalid)
X = Y + A; (also invalid)
```

Explanation: A structure cannot be used in an assignment to an array or to an element variable.

IEL0644I S AGGREGATE D USED WHERE ELEMENT REQUIRED.

AGGREGATE D USED WHERE ELEMENT EXPRESSION IS REQUIRED. STATEMENT IGNORED.

Example:

```
DCL A(10), B(10);
DO I = (A + B) TO 10;
END;
```

Explanation: A structure has been used where the language requires an element expression.

IEL0645I S DIMENSIONS OF D DO NOT MATCH FIRST AGGREGATE.

NUMBER OF DIMENSIONS IN AGGREGATE D DOES NOT MATCH THE FIRST AGGREGATE IN EXPRESSION. STATEMENT IGNORED.

Example:

```
DCL A(6,6), B(6), C(6);
PUT EDIT (A + B + C) )A)5));
```

Explanation: In an expression involving more than one aggregate, all the aggregates involved must have identical dimensions.

IEL0646I S BOUNDS OF D DO NOT MATCH FIRST AGGREGATE.

BOUNDS OF AGGREGATE D DO NOT MATCH THE FIRST AGGREGATE IN EXPRESSION. STATEMENT IGNORED.

Explanation: In an expression involving more than one aggregate, all the aggregates involved must have identical dimensions.

IEL0647I S STRUCTURING OF D DOES NOT MATCH FIRST STRUCTURE.

STRUCTURING OF D DOES NOT MATCH THE FIRST STRUCTURE IN EXPRESSION. STATEMENT IGNORED.

Example:

```
DCL 1 A, 2 B, 2 C,
    1 X, 2 Y, 2 Z, 3 U;
    PUT LIST(A + X);
```

Explanation: In an expression involving more than one structure, all the structures involved must have identical structuring.

IEL0648I S AGGREGATE D USED IN EXTENT SPECIFICATION IN BLOCK.

AGGREGATE D USED FOR EXTENT SPECIFICATION IN 'DECLARE' OR 'DEFAULT' STATEMENT FOR BLOCK BEGINNING AT THIS STATEMENT. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL 1 S,
      2 P,
      2 Q;
DCL 1 A,
      2 B CHAR(5),
      2 C CHAR(S);
```

Explanation: The implementation will assume that the entire content of the aggregate is to be used as the length specification. This can result in a run-time error.

IEL0649I E TARGET OF 'BYNAME' ASSIGNMENT NOT A STRUCTURE.

TARGET OF ASSIGNMENT CONTAINING 'BYNAME' OPTION IS NOT A STRUCTURE. OPTION IGNORED.

Example:

```
DCL (A,B) FIXED;
A = B, BY NAME;
```

Explanation: The BY NAME option can only be used in a structure assignment.

IEL0650I S NO STRUCTURE IN SOURCE OF 'BYNAME' ASSIGNMENT.

NO STRUCTURE IN SOURCE OF 'BYNAME' ASSIGNMENT. STATEMENT IGNORED.

Example:

```
DCL 1 A, 2 OR, 3 RE, 3 GR,
    1 B, 2 OR, 3 RE, 3 BL;
    A = 5,BY NAME;
```

Explanation: The BY NAME option has been used to qualify the assignment of a value that is not a structure.

IEL0651I E D HAS WRONG STRUCTURE ORGANIZATION.

STRUCTURE ORGANIZATION OF D IS NOT THE SAME AS TARGET. ASSIGNMENT MAY NOT BE PERFORMED.

Explanation: Structures used in BYNAME assignment contain base elements with identical names but attributes which do not match.

IEL0652I S INVALID USE OF D IN ARRAY 'INITIAL' IN THIS BLOCK.

INVALID USE OF AGGREGATE D IN ARRAY 'INITIAL' IN THIS BLOCK. 'INITIAL' ATTRIBUTE IGNORED.

Example:

```
DCL ARRAY1 (8,9),
    ARRAY2 (8,9) INIT(ARRAY1),
    ARRAY3 (8) INIT(ARRAY1(*,1));
```

Explanation: The INITIAL attribute for an array can specify initial values for the array elements on an individual basis only. The type of initialization attempted above can be achieved by an assignment statement.

IEL0653I W RESULTS MAY BE UNDEFINED IN ASSIGNMENT TO 'REFER' STRUCTURE D.

ASSIGNMENT TO STRUCTURE D DECLARED WITH 'REFER' OPTION. RESULTS UNDEFINED IF 'REFER'EXTENTS CHANGED BY ASSIGNMENT.

Example:

```
DCL 1 A BASED(P1),2 B,2 C(X REFER B);
    1 S BASED(P2),2 P,2 C(Y REFER P);
A=S;
This becomes
A.B = S.P; (ignored by the compiler
A.C = S.C;  for mapping of C in
           this assignment)
```

Explanation: The values of the bounds or extents of the REFER items in both source and target structures are taken from the target before assignment. If these values do not match in source and target, the values of these extents or bounds in the target will be altered by the assignment, and will not correspond to the REFER items assigned to the target. Therefore, in any subsequent references, the target is undefined.

Programmer Response: If the bounds or extents differ, they should be made to match *prior* to the assignment of the REFER items. The use of the BY NAME option can be convenient in the structure assignment once the REFER bounds or extents have been correctly set up.

IEL0654I S DIMENSIONS OF D DO NOT MATCH TARGET.

NUMBER OF DIMENSIONS OF AGGREGATE D DOES NOT MATCH THE TARGET OF THE ASSIGNMENT OR DUMMY ARGUMENT. STATEMENT IGNORE.

Example:

```
DCL A(5,6), B(5,6), C(5);
A = B + C;
```

Explanation: The number of dimensions of an aggregate to be assigned must match the number of dimensions of the target aggregate.

IEL0655I S BOUNDS OF D DO NOT MATCH TARGET.

BOUNDS OF AGGREGATE D DO NOT MATCH THE TARGET OF THE ASSIGNMENT OR DUMMY ARGUMENT. STATEMENT IGNORED.

Example:

```
DCL A(3,3), B(4,4), C(2:5,-3:-1);
A = A + B; (incorrect)
A = B + C; (also incorrect)
```

Explanation: The bounds for each dimension of an aggregate to be assigned must match the bounds for each dimension of the target aggregate.

IEL0656I S STRUCTURING OF D DOES NOT MATCH TARGET.

STRUCTURING OF D DOES NOT MATCH THE TARGET OF THE ASSIGNMENT OR DUMMY ARGUMENT. STATEMENT IGNORED.

Example:

```
DCL 1 A, 2 B, 2 C;
    1 P, 2 Q, 2 R, 2 S;
    A = P;
```

Explanation: Structures in a structure assignment must have identical structuring.

IEL0657I S AGGREGATE D USED IN EXTENT SPECIFICATION.

AGGREGATE D USED FOR EXTENT SPECIFICATION IN 'ALLOCATE' STATEMENT. STATEMENT IGNORED.

Example:

```
DCL X(*) CTL,
    1 A, 2 B, 2 C;
ALLOCATE X(A);
```

IEL0658I S NO MATCHING IDENTIFIERS FOR 'BYNAME' ASSIGNMENT.

NO MATCHING IDENTIFIERS AT CORRESPONDING LEVELS IN THE STRUCTURES IN 'BYNAME' ASSIGNMENT. STATEMENT IGNORED.

Example:

```
DCL 1 A, 2 B, 2 C,
    1,X, 2 Y, 2 Z,
    1 P, 2 Q, 2 R;
A = X, BY NAME; (incorrect)
A = P + X, BY NAME; (also incorrect)
```

Explanation: In order to use the BY NAME option in a structure assignment, the structure should have matching names at corresponding levels, otherwise no assignment can take place.

IEL0659I U TOO MANY ACTIVE QUALIFIED REFERENCES.

COMPILER RESTRICTION. TOO MANY QUALIFIED REFERENCES ACTIVE IN THIS STATEMENT. PHASE P.

Example:

```
DCL (A,B,C,...Z) (10);
A,B,C,...Z = A;
```

Explanation: A *qualified reference* can result from the use of any of the following:

1. An item declared BASED
2. An item declared DEFINED
3. The first argument of the SUBSTR built-in function or pseudovariable
4. A subscripted item or array expression
5. A multiple concatenation operation
6. SUBSCRIPTRANGE checking

Explanation: A qualified reference is active only for the statement that contains it, unless it is the control variable of a DO-loop, when it is active throughout the scope of the loop. The limit to the number of active qualified references is 32; this limit will be exceeded only if the statement with a qualified reference appears in a nest of DO-loops with qualified control variables, or if the statement is a multiple assignment with many qualified references as targets, or if the statement is a stream I/O statement containing more than 32 items requiring active qualified references.

Programmer Response: Either simplify a multiple assignment or change DO-loop control variables that are qualified references to non-qualified references.

IEL0660I S [PROLOGUE CODE.] NON-CONNECTED ARGUMENT TO T.

[PROLOGUE CODE.] NON-CONNECTED ARGUMENT TO BUILTIN FUNCTION T INVALID. STATEMENT IGNORED.

Explanation: STORAGE and CURRENTSTORAGE built-in functions are only defined for variables which could legally appear in the INTO or FROM option of a record-oriented input/output statement. The INTO or FROM option of a record-oriented input/output statement must refer to an identifier that represents a contiguous area of storage.

IEL0669I S BUILTIN T REQUIRES LANGLVL(SPROG) COMPILER OPTION.

BUILTIN FUNCTION T REQUIRES LANGLVL(SPROG) COMPILER OPTION.

Explanation: The built-in functions POINTERADD, BINARYVALUE and POINTERVALUE require that the LANGLVL(SPROG) compiler option be in effect.

Programmer Response: Specify the LANGLVL(SPROG) compiler option or remove these built-ins from your program.

IEL0670I W THE ADDRESS-MODE OF NON-AUTOMATIC ARGUMENT N MAY CONFLICT WITH FETCHED ENTRY D.

Explanation: The argument flagged in the message is either BASED, CONTROLLED, or STATIC, which might be residing above the 16-megabyte line. If this procedure runs in AMODE(31) and the fetched procedure runs in AMODE(24), the parameter will not be correctly interpreted and access to the parameter can cause unpredictable failure.

Programmer Response: Check the fetched procedure to ensure that it can run AMODE(31). If the called procedure must be run with AMODE(24), declare the argument as AUTOMATIC, or ensure that the argument is below the 16-megabyte line. For more detailed information, refer to the *PL/I VSE Programming Guide.*

IEL0671I W [PROLOGUE CODE.] DUMMY CREATED FOR ARGUMENT N TO ENTRY D.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D DOES NOT MATCH ITS CORRESPONDING PARAMETER OR IS AN ISUB-DEFINED ARRAY. A DUMMY ARGUMENT HAS BEEN CREATED.

Example:

```
DCL X ENTRY (FIXED);
CALL X(A);
```

Explanation: Whenever an argument does not match its parameter, the compiler generates a dummy argument that does match the parameter. On invocation of the entry point, the value of the argument is converted and assigned to the dummy argument. Similarly, when the argument is in ISUB-DEFINED array, the compiler generates a dummy argument. On invocation of the entry point, the value of the argument is assigned to the dummy argument.

IEL0672I W [PROLOGUE CODE.] ARGUMENT N TO T IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO BUILTIN FUNCTION T IS NOT REQUIRED FOR FLOATING POINT RESULT. ARGUMENT IGNORED.

Example:

```
A = DIV(B,C,5,2);
```

Explanation: A superfluous argument has been given in a reference to a built-in function.

IEL0673I I [PROLOGUE CODE.] ARGUMENT N TO T IS NOT 'COMPLEX'.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO BUILTIN FUNCTION T NOT COMPLEX. ZERO IMAGINARY PART ASSUMED.

Example:

```
DCL A REAL;
    A = REAL(A);
```

IEL0674I S INVALID ELEMENT EXPRESSION IN 'DO' OR 'IF'.

INVALID SPECIFICATION OF ELEMENT EXPRESSION IN 'DO' OR 'IF' STATEMENT. VALUE ONE ASSUMED FOR EXPRESSION.

Example:

```
DCL SAM FILE;
IF SAM = PTR THEN...;
```

Explanation: The expression in a DO or IF statement must be a valid element expression which can be evaluated by the compiler.

IEL0675I S INVALID ITERATIVE SPECIFICATION.

INVALID ITERATIVE SPECIFICATION. NON-ITERATIVE 'DO' ASSUMED.

Example:

```
LABEL: DO I = 1 TO LABEL;
```

IEL0676I I [PROLOGUE CODE.] ARGUMENT N TO D ASSUMED TO BE ALIGNED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D IS OF TYPE 'ENTRY' AND IS ASSUMED TO BE ALIGNED.

IEL0677I S [PROLOGUE CODE] SOURCE OF ASSIGNMENT DOES NOT MATCH TARGET D.

[PROLOGUE CODE] ATTRIBUTES OF SOURCE OF ASSIGNMENT STATEMENT CONFLICT WITH TARGET D. [STATEMENT IGNORED.] [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL LV LABEL VARIABLE,
    FV FILE VARIABLE;
LV = FV;
```

Explanation: The variables LV and FV have unlike and unresolvable attributes.

IEL0680I S ATTRIBUTES OF 'REPEAT' EXPRESSION CONFLICT WITH THE CONTROL VARIABLE.

ATTRIBUTES OF 'REPEAT' EXPRESSION CONFLICT WITH THE CONTROL VARIABLE. NON-ITERATIVE 'DO' ASSUMED.

Example:

```
DCL P POINTER;
DCL I FIXED BINARY;
DO I = 1 REPEAT(P);
```

IEL0681I S [PROLOGUE CODE.] ATTRIBUTES OF ARGUMENT N TO ENTRY D CONFLICT WITH PARAMETER.

[PROLOGUE CODE.] ATTRIBUTES OF ARGUMENT NUMBER N TO ENTRY D CONFLICT WITH THE CORRESPONDING PARAMETER. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL A FILE,
    X ENTRY (FLOAT);
    CALL X(A);
```

Explanation: The compiler has detected a conflict between the attributes of an argument and its parameter which cannot be resolved by creating a dummy argument and performing a conversion.

IEL0682I S [PROLOGUE CODE.] ARGUMENT N TO ENTRY D IS NOT 'CONTROLLED'.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D IS NOT 'CONTROLLED' BUT THE CORRESPONDING PARAMETER IS. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL A,
    E ENTRY (CONTROLLED);
CALL E(A);
```

Explanation: A parameter with the CONTROLLED attribute must correspond to an argument with the CONTROLLED attribute.

IEL0683I S [PROLOGUE CODE.] WRONG NUMBER OF ARGUMENTS TO T.

[PROLOGUE CODE.] WRONG NUMBER OF ARGUMENTS TO BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
J = SUBSTR(A,B,C,D);
```

Explanation: A built-in function with either too few or too many arguments has been detected.

IEL0684I S [PROLOGUE CODE.] INVALID DATA TYPE FOR ARGUMENT N TO T.

[PROLOGUE CODE.] ARGUMENT NUMBER N HAS INCORRECT DATA TYPE FOR BUILTIN FUNCTION T. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL F FILE;
A = SIN(F);
```

IEL0685I S [PROLOGUE CODE.] MODE OF ARGUMENT N TO T IS INCORRECT.

[PROLOGUE CODE.] THE MODE OF ARGUMENT NUMBER N TO BUILTIN FUNCTION T IS INCORRECT. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL A COMPLEX;
B = CEIL(A);
```

IEL0686I S [PROLOGUE CODE.] ARGUMENT N TO T IS NOT INTEGER CONSTANT.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO BUILTIN FUNCTION T IS NOT AN INTEGER CONSTANT. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
A = DECIMAL(B,C,D)
```

IEL0687I S [PROLOGUE CODE.] CONSTANT OR FUNCTION OR TEMPORARY RESULT HAS INVALID ATTRIBUTES FOR EXPRESSION.

[PROLOGUE CODE.] CONSTANT OR FUNCTION OR TEMPORARY RESULT HAS INVALID ATTRIBUTES FOR EXPRESSION. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
L:  A = 1;
    B = 2 + L;
EC: PROC(Z) RETURNS(OFFSET);
    B = 2 + EC(1);
```

Explanation: This message is issued for incorrect usage of based or subscripted variables. If the pointer arithmetic is utilized, the LANGLVL(SPROG) compiler option must be specified.

IEL0688I S [PROLOGUE CODE.] ASSIGNMENT TO CONSTANT.

[PROLOGUE CODE.] TARGET OF ASSIGNMENT IS A CONSTANT. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Explanation: The target of an assignment can never be a constant; it must always be a variable.

IEL0689I S [PROLOGUE CODE.] SOURCE OF ASSIGNMENT DOES NOT MATCH TARGET.

[PROLOGUE CODE.] ATTRIBUTES OF SOURCE OF ASSIGNMENT CONFLICT WITH TARGET. [STATEMENT IGNORED.] [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL O OFFSET (A),
    A AREA BASED (P);
O->A = X;
```

Explanation: The variables have unlike, and unresolvable, attributes.

IEL0690I S [PROLOGUE CODE.] OPERAND D INVALID IN ELEMENT EXPRESSION.

[PROLOGUE CODE.] INVALID USE OF OPERAND D IN AN ELEMENT EXPRESSION. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL A(10), F FILE;
a.  READ FILE(F) SET(P) KEY(A + B);
b.  B = F + C;
```

Explanation: An element expression cannot refer to a structure or unsubscripted array. Arithmetic operations can never involve nonarithmetic data such as files or events. Pointer arithmetic is allowed under the LANGLVL(SPROG) compiler option.

IEL0691I U LEVEL OF NESTING FOR 'DO', OR 'IF', OR 'SELECT' STATEMENT EXCEEDS N.

COMPILER RESTRICTION. LEVEL OF NESTING FOR 'DO', OR 'IF', OR 'SELECT' STATEMENT EXCEEDS N.

IEL0692I S [PROLOGUE CODE.] ARGUMENT N TO ENTRY D INVALID.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D INVALID. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

IEL0694I E [PROLOGUE CODE.] NO SELECTION POSSIBLE FOR 'GENERIC' NAME.

[PROLOGUE CODE.] NO SELECTION POSSIBLE FOR 'GENERIC' NAME. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
DCL X FLOAT,
    I FIXED BINARY,
    E ENTRY,
    G GENERIC(E WHEN (FLOAT));
X = G(I);
```

Explanation: A reference to a generic name should contain arguments with attributes that match the generic descriptor list for one of the generic entry constants.

IEL0695I S [PROLOGUE CODE.] OPERANDS OF COMPARE CONFLICT.

[PROLOGUE CODE.] ATTRIBUTES OF OPERANDS IN AN EQUAL OR NOT-EQUAL OPERATION CONFLICT. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

Example:

```
    DCL F FILE;
L:  IF L = F THEN...;
```

IEL0697I S [PROLOGUE CODE] SOURCE OF ASSIGNMENT DOES NOT MATCH TARGET D.

[PROLOGUE CODE.] ATTRIBUTES OF SOURCE OF ASSIGNMENT STATEMENT CONFLICT WITH THE TARGET D. STATEMENT IGNORED. [RESULTS OF PROLOGUE UNDEFINED.]

IEL0702I W 'NOMAP' SPECIFIED. MAPPING OF PARAMETER N TO D MAY DIFFER IN T.

MAPPING OF PARAMETER N TO ENTRY D MAY DIFFER IN PL/I AND T BUT DUMMY PARAMETER NOT CREATED BECAUSE OF 'NOMAP' OPTION.

Programmer Response: Ensure either that the parameter and its corresponding argument are mapped identically in the two language implementations or that differences in mapping are allowed for in the descriptions (or declarations) used in the two languages.

IEL0703I W 'NOMAP' SPECIFIED. MAPPING OF ARGUMENT N TO D MAY DIFFER IN T.

MAPPING OF ARGUMENT NUMBER N TO ENTRY D MAY DIFFER IN PL/I AND T BUT DUMMY ARGUMENT NOT CREATED BECAUSE OF 'NOMAP' OPTION.

Programmer Response: Ensure either that the argument and its corresponding parameter are mapped identically in the two language implementations or that differences in mapping are allowed for in the descriptions (or declarations) used in the two languages.

IEL0704I S MORE THAN N ARGUMENTS TO T ENTRY D.

COMPILER RESTRICTION. NUMBER OF ARGUMENTS TO T ENTRY D EXCEEDS N. EXCESS ARGUMENTS IGNORED.

Explanation: The maximum number of arguments that can be passed to an ASSEMBLER or COBOL routine in a single invocation is 64.

Programmer Response: Eliminate the excess number of arguments. If necessary and feasible, make these arguments known in both the invoking and invoked routines by declaring them STATIC EXTERNAL in PL/I and the equivalent to this in the invoked routine.

IEL0705I S EXTENTS OF PARAMETER N TO T ENTRY D NOT FIXED.

EXTENTS OF PARAMETER N TO T ENTRY D ARE NOT FIXED. RESULTS OF EXECUTION UNDEFINED.

Example:

```
C:  ENTRY(M) OPTIONS (COBOL);
    DCL M CHAR(*);              /* invalid */
```

Explanation: All bounds and extents of parameters to entry points invoked from COBOL must be specified as decimal integer constants.

IEL0706I I T MAPPING USED FOR DUMMY ARGUMENT N TO D.

T MAPPING USED FOR DUMMY ARGUMENT NUMBER N TO ENTRY D.

Explanation: COBOL mapping has been used for a dummy argument that has been created for an argument that is to be passed to a COBOL routine.

IEL0707I I PL/I MAPPING USED FOR DUMMY ARGUMENT N TO D.

PL/I MAPPING USED FOR DUMMY ARGUMENT NUMBER N TO ENTRY D.

Example:

```
DCL FF ENTRY OPTIONS(COBOL,
             NOMAPIN(ARG1)),
             A(10,10);
             CALL FF(A + I);
```

Explanation: A dummy argument is created for this argument according to normal PL/I rules with the NOMAPOUT option. The explicit use of the NOMAPIN option will combine with NOMAPOUT to produce the effective specification of the NOMAP option.

IEL0708I I DUMMY CREATED FOR ARGUMENT N TO T ENTRY D.

MAPPING OF ARGUMENT NUMBER N TO ENTRY D MAY DIFFER IN PL/I AND T. DUMMY ARGUMENT CREATED.

Example:

```
DCL 1 A,
    2 B CHAR(1),
    2 C FIXED BIN(31,0),
    X(10,10),
    COB ENTRY OPTIONS(COBOL);
CALL COB(A);
(message produced for A)
```

IEL0709I I DUMMY CREATED FOR PARAMETER N TO T ENTRY D.

MAPPING OF PARAMETER N TO ENTRY D MAY DIFFER IN PL/I AND T. DUMMY PARAMETER CREATED.

Example:

```
C:  ENTRY(A) OPTIONS(COBOL);
   DCL 1 A,
         2 B CHAR(1),
         2 C FIXED BIN(31,0);
```

IEL0710I E D CONTAINS DATA INVALID FOR 'COBOL'.

RECORD VARIABLE D FOR 'COBOL' FILE CONTAINS 'AREA' OR 'BIT' DATA WITH NO EQUIVALENT IN 'COBOL'. PL/I MAPPING ASSUMED FOR VARIABLE.

Explanation: A PL/I data type specified for a COBOL FILE has no equivalent in COBOL.

IEL0711I S T IGNORED FOR 'CALL' WITH TASKING OPTION.

T OPTION IGNORED FOR 'CALL' WITH TASKING OPTION.

Example:

```
DCL A ENTRY OPTIONS(COBOL);
CALL A TASK;
```

Explanation: Interlanguage subroutines cannot be tasks.

IEL0712I W PL/I MAPPING ASSUMED FOR ARRAY RECORD VARIABLE.

RECORD VARIABLE IS AN ARRAY. PL/I MAPPING ASSUMED FOR VARIABLE.

Example:

```
DCL 1 A(8),
    2 B,
    2 C,
    3 D(5);
DCL F FILE ENV(COBOL) RECORD;
READ FILE (F) INTO (A);
```

Explanation: A PL/I data type specified for a COBOL file has no equivalent in COBOL.

IEL0713I S 'COBOL' FILE D INVALID IN ASSIGNMENT OR AS ARGUMENT.

USE OF COBOL FILE D IN ASSIGNMENT OR AS AN ARGUMENT IS INVALID. 'COBOL' OPTION WILL NOT APPLY TO TARGET.

Example:

```
DCL PROC ENTRY (FILE),
    COBFIL FILE ENV(COBOL...);
    CALL PROC(COBFILE);
```

IEL0714I W D CONTAINS DATA INVALID FOR 'COBOL'.

RECORD VARIABLE FOR 'COBOL' FILE CONTAINS ELEMENT WITH NO DIRECT EQUIVALENT IN 'COBOL'. 'COBOL' MAPPING ASSUMED FOR VARIABLE.

Explanation: "A" has fractional precision which is not available for fixed binary (COMPUTATIONAL) variables in COBOL.

Programmer Response: If appropriate, correct the data type attributes to one of those supported by the interlanguage communication definition. For further details see the *LE/VSE Programming Guide.*

IEL0715I E STATEMENT INVALID FOR 'COBOL' FILE D.

STATEMENT INVALID FOR COBOL FILE D. PL/I MAPPING ASSUMED FOR RECORD.

Example:

```
DCL F FILE ENV(COBOL);
DELETE FILE(F);
```

IEL0716I E 'SET' OPTION INVALID FOR 'COBOL' FILE D.

'SET' OPTION ON 'READ' STATEMENT INVALID FOR COBOL FILE D. PL/I MAPPING ASSUMED FOR RECORD.

Explanation: Locate mode input/output is not allowed for a COBOL file. Move mode must be used.

IEL0717I 'EVENT' OPTION INVALID FOR 'COBOL' FILE D.

'EVENT' OPTION INVALID FOR COBOL FILE D WHEN PL/I AND COBOL MAPPING MAY DIFFER. PL/I MAPPING ASSUMED FOR RECORD.

Example:

```
DCL F FILE ENV(COBOL),
    1 R,
    2 S CHAR(1)
    2 T FIXED BIN(31,0);
 READ FILE (F) INTO (R) EVENT (EV);
```

Explanation: The EVENT option is allowed only if it can be deduced at compile-time that the mapping of the record will be the same in PL/I and COBOL.

IEL0720I E ARGUMENT N TO D CONTAINS DATA INVALID FOR T.

ARGUMENT N TO ENTRY D CONTAINS 'AREA' OR 'BIT' DATA WITH NO EQUIVALENT IN T. PL/I MAPPING ASSUMED FOR ARGUMENT IF AGGREGATE.

Example:

```
DCL I BIT (10), E ENTRY
EXTERNAL OPTIONS (COBOL);
CALL E(I);
```

Explanation: An argument which has no direct equivalent in COBOL has been encountered in a CALL statement or function reference to invoke a COBOL routine.

IEL0721I E PARAMETER N TO D CONTAINS DATA INVALID FOR T.

PARAMETER N TO ENTRY D CONTAINS 'AREA' OR 'BIT' DATA WHICH HAS NO EQUIVALENT IN T. PL/I MAPPING ASSUMED FOR PARAMETER IF AGGREGATE.

Example:

```
E: ENTRY (X,Y,Z) OPTIONS (COBOL);
DCL Y BIT(8);
```

Explanation: A parameter which has no direct equivalent in COBOL has been encountered in a PROCEDURE or ENTRY statement invoked from a COBOL routine.

IEL0722I E ARGUMENT N TO 'COBOL' ENTRY D IS AN ARRAY.

ARGUMENT N TO ENTRY D IS AN ARRAY WHICH IS INVALID FOR 'COBOL'. PL/I MAPPING ASSUMED FOR ARGUMENT.

Example:

```
DCL E ENTRY EXTERNAL OPTIONS (COBOL),
   I(8) FIXED BIN;
   CALL E(I);
```

Explanation: COBOL data types do not include the equivalent of PL/I arrays.

IEL0723I E PARAMETER N TO 'COBOL'ENTRY D IS AN ARRAY.

PARAMETER N TO ENTRY D IS AN ARRAY WHICH IS INVALID FOR 'COBOL'. PL/I MAPPING ASSUMED FOR PARAMETER.

Example:

```
E:  ENTRY (A,B,C) OPTIONS (COBOL);
    DCL A(8) FIXED BIN;
```

Explanation: COBOL data types do not include the equivalent of PL/I arrays.

IEL0724I W DATA IN ARGUMENT N TO D INVALID FOR T.

ARGUMENT NUMBER N TO ENTRY D CONTAINS ELEMENT WITH NO DIRECT EQUIVALENT IN T.

Example:

```
DCL E ENTRY EXTERNAL OPTIONS (COBOL);
DCL I FIXED BIN (10,6);
CALL E(I);
```

Explanation: "I" should have the precision (n,0).

Programmer Response: If appropriate, correct the data type attributes to one of those supported by the interlanguage communication definition. For further details see the *LE/VSE Programming Guide.*

IEL0725I W DATA IN PARAMETER N TO D INVALID FOR T.

PARAMETER N TO ENTRY D CONTAINS ELEMENT WHICH HAS NO DIRECT EQUIVALENT IN T.

Example:

```
E:  ENTRY (I,J,K) OPTIONS (COBOL);
    DCL I FLOAT DEC (20);
```

Explanation: COBOL does not implement extended precision floating-point variables, but only variables with short precision (COMPUTATIONAL-1) or long precision (COMPUTATIONAL-2).

Programmer Response: If appropriate, correct the data type attributes to one of those supported by the interlanguage communication definition. For further details see the *LE/VSE Programming Guide.*

IEL0727I S PARAMETER N TO T ENTRY D MUST NOT BE 'CONTROLLED'.

PARAMETER N TO ENTRY D HAS 'CONTROLLED' STORAGE CLASS WITH NO EQUIVALENT IN T. RESULTS OF EXECUTION UNDEFINED.

Example:

```
E:  ENTRY (I,J) OPTIONS (COBOL);
    DCL J CONTROLLED;
```

IEL0730I S ARGUMENT N TO T ENTRY D IS NOT 'CONNECTED'.

ARGUMENT N TO T ENTRY D IS NOT 'CONNECTED' AND THE 'NOMAP' OPTION IS SPECIFIED. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL 1 A(3), 2 B, 3 C, 2 D,
    E ENTRY EXTERNAL OPTIONS (COBOL, NOMAP);
   CALL E (B);
```

Explanation: An argument must occupy a contiguous area of storage when passed as a parameter to an invoked routine.

IEL0731I I ARGUMENT N TO T ENTRY D ASSUMED TO BE 'CONNECTED'.

ARGUMENT PASSED TO UNCONNECTED PARAMETER N OF T ENTRY D IS ASSUMED TO BE CONNECTED.

Example:

```
E:  ENTRY (A,B,D) OPTIONS
   (COBOL, NOMAP);
   DCL 1 A(3), 2 B, 3 C, 2 D,
       3 E;
```

Explanation: A parameter must be a variable that occupies a contiguous area of storage.

IEL0735I W ARGUMENT N TO D NOT CORRECTLY ALIGNED FOR T.

ARGUMENT N TO ENTRY D IS AN ELEMENT WHICH MAY NOT BE CORRECTLY ALIGNED FOR T. NO DUMMY ARGUMENT CREATED.

Example:

```
       DCL 1 A UNALIGNED,
           2 B BIT(5),
           2 C BIT(27),
       FF ENTRY OPTIONS(COBOL);
       CALL FF(C);
    /* C IS AN UNALIGNED ELEMENT */
```

Explanation: Although, according to PL/I rules, a dummy argument is not created for an element argument, the alignment of the argument might not be acceptable as a parameter to a COBOL routine, and an addressing interrupt can occur when the routine is invoked.

IEL0740I S D CANNOT BE A FUNCTION.

'COBOL' OR 'ASSEMBLER' ENTRY D CANNOT BE INVOKED AS A FUNCTION. INTERLANGUAGE OPTION IGNORED.

Example:

```
DCL SUB ENTRY OPTIONS(COBOL);
DCL (A,B);
A = SUB(B);
```

Explanation: A COBOL or ASSEMBLER procedure cannot be invoked as a function by a PL/I program.

IEL0742I U AGGREGATE D EXCEEDS MAXIMUM LENGTH.

COMPILER RESTRICTION. AGGREGATE D EXCEEDS MAXIMUM LENGTH. COMPILATION WILL BE TERMINATED AFTER PHASE 'IQ'.

Example:

```
DCL A (256,256,256,256) FIXED BINARY;
```

Explanation: The maximum number of bytes in an aggregate must be equal to or less than 2**31 - 1 bytes for all but unaligned bit data. For unaligned bit aggregates, the maximum number of bytes is 2**28 - 1. There is a special exception for a structure that contains an unaligned bit array. The position of all elements of the unaligned bit array must be within 2**28 - 1 limit from the start of the structure, independent of the size of the array or structure.

IEL0743I S ARGUMENT N TO D INVALID.

COMPILER RESTRICTION. ARGUMENT N TO ENTRY D IS AN ADJUSTABLE STRING AGGREGATE. RESULTS OF EXECUTION UNDEFINED.

Explanation: If an array or structure expression is used as the argument to a function procedure or subroutine procedure, then the length of any string expression contained in it must be available to the compiler during compilation. Therefore, only constant-length strings can be used or the corresponding parameter descriptor must specify the string length.

IEL0750I U THE PHASE IEL1AE COULD NOT BE DELETED. THE COMPILER IS TERMINATED.

Explanation: The compiler initialization phase IEL1AE was not successfully deleted by the CDDELETE macro, the compiler can not continue because the GETVIS storage occupied by IEL1AE is required to load other compiler phases.

Programmer Response: Rerun the compile. If the problem persists see your systems programmer as there may be an operating system problem.

IEL0751I W CLOSE ERROR ON T FILE.

Explanation: The compiler was not able to successfully close file T. This is a possible operating system problem.

Programmer Response: Rerun the compile. If the problem persists see your systems programmer as there may be an operating system problem.

IEL0752I OPTION 'CMPAT(V1)' IS NOT SUPPORTED. IT IS REPLACED BY 'CMPAT(V2)'.

Explanation: CMPAT(V1) is not supported by this compiler, processing continues as if CMPAT(V2) was coded.

IEL0753I W THE 'MEMBER-TYPE' ON THE %INCLUDE STATEMENT IS GREATER THAN 1 CHARACTER. A 'MEMBER-TYPE' OF 'P' IS ASSUMED.

Example:

```
%INCLUDE ZHA(MEMBER);
```

Explanation: The member type on the %INCLUDE is greater than one character. In the above example ZHA is defaulted to P.

Programmer Response: Check the %INCLUDE statement for possible errors in the member type.

IEL0756I W 'CHECK' CONDITION IS NOT SUPPORTED.

'CHECK' CONDITION IS NOT SUPPORTED. 'CHECK' CONDITION IGNORED AT EXECUTION TIME.

Example:

```
 (CHECK(A)):  A = B;
```

Explanation: The CHECK condition appears in a condition prefix, SIGNAL, REVERT or ON statement. This condition is no longer supported by PL/I. APARs related to this condition will not be accepted. The condition is always disabled at execution time.

Programmer Response: See the books that provide information on testing and debugging your program for a replacement function for CHECK. Remove the use of the CHECK condition to avoid performance loss.

If your program is syntactically and semantically correct but compilation or run-time errors occur, remove the use of the CHECK condition from your program.

IEL0758I W ARGUMENT N OF D HAS INVALID VALUE.

COMPILER RESTRICTION. VALUE OF ARGUMENT N OF BUILT-IN FUNCTION 'T' IS OUTSIDE THE PERMITTED RANGE. ARGUMENT REPLACED BY IMPLEMENTATION MAXIMUM PRECISION.

Example:

```
DCL (A,B,C) FIXED DECIMAL (10,0);
C = MULTIPLY (A,B,18,4);
```

Explanation: The implementation maximum precision for fixed decimal is 15; the third argument is replaced by 15.

Programmer Response: Change the source so that the specified argument is not greater than:

15 for FIXED DECIMAL results
31 for FIXED BINARY results
33 for FLOAT DECIMAL results
109 for FLOAT BINARY results

IEL0759I S INVALID USE OF FETCHED ENTRY CONSTANT D.

INVALID USE OF FETCHED ENTRY CONSTANT D. FETCHED PROCEDURES MAY BE SPECIFIED ONLY IN 'FETCH'OR 'RELEASE' STATEMENTS OR AS ENTRIES IN FUNCTION OR SUBROUTINE REFERENCES.

Example:

```
DCL F1 EXT ENTRY;
DCL F2 ENTRY INIT (F1);
FETCH F1;
F2 = F1;
```

Explanation: F1 has been used invalidly in the declaration of F2 and in the assignment to F2. The assignment will be ignored and the INIT value ignored. (This leaves F2 uninitialized.)

IEL0760I E BIT VALUE ZERO ASSUMED IN 'UNTIL' EXPRESSION.

VARIABLE IN 'UNTIL' EXPRESSION CANNOT BE CONVERTED TO BIT STRING. BIT CONSTANT OF LENGTH ONE AND VALUE ZERO ASSUMED.

Example:

```
DCL P POINTER;
  DO UNTIL(P);
```

Explanation: In the example, pointer P cannot be converted to a bit string.

IEL0761I E VARIABLE IN CONDITIONAL EXPRESSION CANNOT BE CONVERTED TO BIT STRING.

VARIABLE IN CONDITIONAL EXPRESSION CANNOT BE CONVERTED TO BIT STRING. BIT CONSTANT OF LENGTH AND VALUE ONE ASSUMED.

Example:

```
1.  DCL F FILE;
    IF F THEN X = Y;

2.  DCL F FILE;
    SELECT;
    WHEN(F) X = Y;

3.  DCL F FILE;
    DO I = 1 WHILE(F);
```

Explanation: Only string and arithmetic element variables and constants are allowed in the conditional clause of an IF or WHEN statement or WHILE or UNTIL expression.

IEL0762I W TOO FEW ARGUMENTS IN CALL TO D.

FEWER ARGUMENTS THAN PARAMETERS FOR CALL TO 'ASSEMBLER' PROCEDURE D.

Example:

```
DCL P ENTRY(FIXED,FIXED)
  OPTIONS(ASSEMBLER);
CALL P(A);
```

Explanation: An assembly language external procedure has been invoked with fewer arguments than the number of parameters to the corresponding DECLARE statement.

IEL0763I E NEGATIVE SECOND ARGUMENT TO 'BIT' OR 'CHAR'.

NEGATIVE SECOND ARGUMENT TO 'BIT' OR 'CHAR' BUILTIN FUNCTION. ZERO ASSUMED.

Example:

```
PUT LIST(BIT(I,-3));
```

Explanation: The second argument to the 'BIT' or 'CHAR' built-in function specifies the string length for the converted first argument. This length cannot be negative.

IEL0764I E LENGTH OF STRING OPERATION RESULT EXCEEDS N.

COMPILER RESTRICTION. LENGTH OF RESULT OF STRING OPERATION EXCEEDS N. LENGTH OF N ASSUMED.

Example:

```
DCL (A,B) CHAR(32767);
C = A||B;
```

Explanation: A character string cannot exceed 32767 characters in length and a graphic string cannot exceed 16383 graphics in length.

IEL0765I S NON-CONSTANT VALUE IN STATIC INITIAL FOR D IN THIS BLOCK.

NON-CONSTANT VALUE IN A STATIC 'INITIAL' SPECIFICATION FOR VARIABLE D IN THE BLOCK BEGINNING WITH THIS STATEMENT. 'INITIAL' SPECIFICATION IGNORED.

Example:

```
DCL X(2) STATIC INITIAL(Y);
```

Explanation: Only constants can appear in the INITIAL attribute for STATIC variables.

IEL0766I E 'REPEAT' STRING RESULT EXCEEDS MAXIMUM LENGTH.

COMPILER RESTRICTION. STRING RESULT FROM 'REPEAT' BUILTIN FUNCTION GREATER THAN ALLOWED MAXIMUM LENGTH. ZERO REPETITION FACTOR ASSUMED.

Example:

```
A = REPEAT('XXX',40000);
```

Explanation: The result should not produce a string greater than 32,767 characters (or bits) in length.

IEL0767I E NEGATIVE REPETITION FACTOR FOR 'REPEAT'.

NEGATIVE REPETITION FACTOR SPECIFIED FOR 'REPEAT' BUILTIN FUNCTION. ZERO REPETITION FACTOR ASSUMED.

Example:

```
A = REPEAT('XXX',-2);
```

IEL0768I W CONSTANT SPECIFIED WHERE EXPRESSION EXPECTED.

CONSTANT SPECIFIED WHERE EXPRESSION EXPECTED. FLOW OF CONTROL WILL BE UNCONDITIONAL.

Example:

```
     IF 1 THEN ...;
          ELSE ...;

(The THEN clause will always be run.)

     IF 0 THEN ...;
          ELSE ...;

(The ELSE clause will always be run.)

     DO WHILE(1);
     END;

(The loop will always be run and might be
 permanent).

     DO WHILE(0);
     END;

(The loop will never be run.)

     DO UNTIL(1);
     END;

(The loop will be run once and only once.)

     DO UNTIL(0);
     END;

(The loop will be run repeatedly and might
 be permanent.)

     SELECT;
     WHEN(1)...;
     OTHERWISE...;
     END;

(The WHEN unit will be run.)

     SELECT;
     WHEN(0)...;
     OTHERWISE...;
     END;

(The OTHERWISE unit will be run.)
```

Explanation: A constant has been supplied in an IF statement, a WHILE or UNTIL expression, or in a WHEN statement (there being no SELECT expression). Running the statement can result in one flow of control only.

IEL0769I S AREA VARIABLE FOR OFFSET D INVALID.

COMPILER RESTRICTION. SPECIFICATION OF AREA VARIABLE ASSOCIATED WITH OFFSET D NOT VALID IN THIS STATEMENT.

Explanation: See the relevant section of the language reference manual for this compiler for an explanation of the language restrictions concerning the use of qualified AREA variables.

IEL0770I W THIRD ARGUMENT TO 'MPSTR' BUILTIN FUNCTION IS NEGATIVE. ZERO ASSUMED.

Explanation: The third argument is used to determine the length of the resulting string. It cannot be less than zero.

IEL0771I W THE DBCS ORDERING FACILITY WAS NOT FOUND OR THE FACILITY WAS NOT LOADED DUE TO INSUFFICIENT STORAGE. BINARY SEQUENCE ASSUMED.

Explanation: The DBCS ordering program is not available. DBCS identifiers cannot be put in the XREF portion of the listing according to a natural sequence. However, the DBCS identifiers will be put in the listing using the identifier's binary sequence.

IEL0775I W CONSTANT ITERATION FACTOR EXCEEDS 65,535.

COMPILER RESTRICTION. COMPILER ITERATION FACTOR IN INITIAL ATTRIBUTE FOR STATIC ARRAY CANNOT EXCEED 65,535. FIRST 65,535 ELEMENTS INITIALIZED.

Example:

```
DCL 1 A(4800) STATIC,
     2 B(20) FIXED DEC (1,0)
         INIT ((96000)1);
```

Explanation: The iteration factor in an INITIAL list for an array cannot exceed 65,535. Only the first 65,535 elements of the array will be initialized.

Programmer Response: Multiple iteration factors can be specified, each less than 65,535. For example, the above INITIAL attribute can be coded: INIT ((64000)1,(32000)1);

IEL0776I E CONSTANT SUBSCRIPT OF D OUT OF RANGE.

VALUE OF CONSTANT SUBSCRIPT FOR ARRAY D IS OUT OF RANGE BUT HAS NOT BEEN REPLACED.

Example:

```
(SUBSCRIPTRANGE):PIG:PROC;
                 DCL A(2,3);
                 A(6,3) = 1;
                 END;
```

IEL0777I W TOO MANY ITEMS IN 'INITIAL' LIST FOR D.

TOO MANY ITEMS IN 'INITIAL' LIST FOR ARRAY D. REDUNDANT ITEMS IGNORED.

Example:

```
DCL A(2) INIT(1,2,3);
```

IEL0778I S 'ISUB' VARIABLE FOR D OUT OF RANGE.

'ISUB' VARIABLE FOR DEFINED ARRAY D OUT OF RANGE. RESULTS OF EXECUTION UNDEFINED.

IEL0779I S INVALID REPETITION FACTOR IN 'INITIAL' FOR D.

ZERO OR NEGATIVE REPETITION FACTOR IN 'INITIAL' ATTRIBUTE FOR ARRAY D. REPETITION FACTOR IGNORED.

Example:

```
DCL A(10) INIT((0)1,(-2)1);
```

IEL0787I W INITIAL VALUE OF ITERATIVE SPECIFICATION OUT OF RANGE.

INITIAL VALUE OF ITERATIVE SPECIFICATION IS OUTSIDE THE RANGE OF THE 'BY' AND 'TO' EXPRESSIONS. LOOP WILL NOT BE EXECUTED.

Example:

```
1.  DO I = 1 BY -2 TO 10;

2.  DO I = 1 BY 3 TO -10;
```

IEL0792I W TITLE OPTION REQUIRED ON OPEN STATEMENT FOR FILE '<kkkk>'.

Example:

```
OPEN FILE(<kkkk>);
```

Explanation: The non-EBCDIC DBCS file name cannot be used as an external name. The TITLE option must be added to supply an EBCDIC name for the host system.

IEL0798I S INVALID IDENTIFIER IN CHECK LIST.

INVALID IDENTIFIER IN CHECK LIST. IDENTIFIER IGNORED.

Example:

```
(CHECK(F)): P:PROC;
        DCL F FILE;
```

Explanation: The identifier in the check list must be a variable, or a label or entry constant.

Programmer Response: Remove the invalid identifier from the name list in the CHECK condition prefix.

IEL0799I W AREA ASSOCIATED WITH OFFSET D MAY BE INVALID FOR LOCATOR CONVERSION IN 'RETURN'.

COMPILER RESTRICTION. AREA ASSOCIATED WITH OFFSET D INVALID FOR LOCATOR CONVERSION FOR 'RETURN' STATEMENT. 'RETURN' EXPRESSION WILL BE IGNORED IF THE INVALID COMBINATION OF 'RETURN' AND 'ENTRY' IS USED.

Example:

```
     P: PROC RETURNS (OFFSET(A));
     Q: ENTRY RETURNS (POINTER);
        DCL A AREA BASED,
            PTR POINTER,
        RETURN (PTR);
```

Explanation: If locator conversion is required in a RETURN statement, the offset must have an associated area. The area must be unsubscripted, it cannot be defined; if it is based it must be based on an explicit non-based, non-defined unsubscripted pointer.

Programmer Response: Ensure that the combination of return expression and entry type never requires locator conversion to be performed.

IEL0800I S INVALID SPECIFICATION IN 'WAIT'.

INVALID SPECIFICATION OF NUMBER OF EVENTS IN 'WAIT' STATEMENT. SPECIFICATION IGNORED.

Example:

```
DCL (E1,E2) EVENT, F FILE;
  ⋮
WAIT (E1,E2)(F);
```

Explanation: The number of events specification in the WAIT statement must be convertible to a FIXED BINARY integer.

IEL0801I S INVALID EXPRESSION IN 'DELAY'.

INVALID EXPRESSION IN 'DELAY' STATEMENT. ZERO ASSUMED.

Example:

```
DCL F FILE;
DELAY (F);
```

Explanation: The expression in the DELAY statement must be convertible to a FIXED BINARY integer.

IEL0802I S INVALID EXPRESSION IN 'RETURN'.

INVALID EXPRESSION IN 'RETURN' STATEMENT. EXPRESSION IGNORED.

Example:

```
DCL C CONDITION;
RETURN (C);
```

Explanation: The expression in a RETURN statement must be problem data or locator, area, label, event, file, or task program control data.

IEL0803I S INVALID 'DISPLAY' EXPRESSION.

'DISPLAY' EXPRESSION IS NOT A VALID DATA TYPE OR ELEMENT EXPRESSION. STATEMENT IGNORED.

Example:

```
DCL F FILE;
DISPLAY (F);
```

Explanation: The argument of a DISPLAY statement must be an element expression that can be converted to character form.

IEL0804I W 'DISPLAY' STRING LENGTH EXCEEDS 72.

STRING LENGTH FOR 'DISPLAY' EXCEEDS 72 CHARACTERS. TERMINAL MAY NOT SUPPORT THIS [FIRST N CHARACTERS USED].

Example:

```
DCL A CHAR (150);
DCL B CHAR(80);
DISPLAY (A);
DISPLAY (B);
```

Explanation: The first 126 characters of a DISPLAY expression will always be *transmitted* to the terminal, but if the terminal cannot display more than 72 bytes, and cannot continue the record on a succeeding line, only the first 72 characters will be displayed.

If the length of the DISPLAY expression is more than 126, only the first 126 characters will be displayed, even if there is no restriction on the terminal used by the system.

In the example above, DISPLAY(A) will display the first 126 bytes, although they might be on multiple lines. DISPLAY(B) will display 80 bytes, and perhaps all of the message text, although the text might be on multiple lines.

IEL0805I S 'REPLY' CONTAINS NON-ELEMENT EXPRESSION.

'REPLY' EXPRESSION IS NOT A CHARACTER STRING EXPRESSION. 'REPLY' OPTION IGNORED.

Example:

```
DCL ABC(2,3) FIXED BINARY;
DISPLAY('MESSAGE') REPLY(ABC(1,1));
```

Explanation: The expression in the REPLY statement is not a character string variable. The REPLY option is ignored. If an EVENT option is present, this too is ignored.

IEL0806I E 'REPLY' STRING LENGTH EXCEEDS N.

COMPILER RESTRICTION. STRING LENGTH FOR 'REPLY' TOO LONG. FIRST N CHARACTERS USED.

Example:

```
DCL R CHAR(100);
DISPLAY('MESSAGE') REPLY(R);
```

Explanation: The length of the REPLY expression is more than 72 bytes. Only the first 72 bytes of the message are accepted.

IEL0808I W T NOT ENABLED. 'SIGNAL' IGNORED.

T CONDITION NOT ENABLED. STATEMENT IGNORED.

Example:

```
(NOZERODIVIDE):PIG:PROC;
                 SIGNAL ZERODIVIDE;
                 END;
```

Explanation: The SIGNAL statement for a disabled condition acts as a null statement.

IEL0809I W 'CHECK' NOT ENABLED FOR D.

'CHECK' CONDITION NOT ENABLED FOR VARIABLE D IN SIGNAL STATEMENT. VARIABLE IGNORED.

Example:

```
(CHECK(A,B)):PIG:PROC;
              SIGNAL CHECK(A,B,C);
              END;
```

Explanation: The syntax of the CHECK condition is still analyzed at compile time; however, the CHECK condition is no longer supported and is always disabled at run time.

IEL0810I S INVALID EXPRESSION IN 'PRIORITY' OPTION.

INVALID EXPRESSION IN 'PRIORITY' OPTION. OPTION IGNORED.

Explanation: The expression in the PRIORITY option must represent an integer value with the precision (15,0).

IEL0811I S INVALID SPECIFICATION OF 'IGNORE' EXPRESSION.

INVALID SPECIFICATION OF 'IGNORE' EXPRESSION. VALUE ONE ASSUMED.

Example:

```
DCL F FILE;
READ FILE(F) IGNORE(F);
```

Explanation: The expression in the IGNORE option must represent an arithmetic integer value.

IEL0812I S 'KEY' SPECIFICATION INVALID.

'KEY' SPECIFICATION INVALID. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL F FILE, A(5) FIXED;
READ FILE(F) INTO(A) KEY(A);
```

Explanation: The expression in the KEY option must represent a valid key derived from a character string or an arithmetic variable.

IEL0816I S ATTRIBUTES OF D CONFLICT WITH USE.

CONFLICT BETWEEN ATTRIBUTES OF FILE D AND ITS USE IN THIS STATEMENT. STATEMENT IGNORED.

Example:

```
DCL F FILE OUTPUT;
READ FILE(F) INTO(A);
```

IEL0817I S RECORD VARIABLE INVALID.

RECORD VARIABLE INVALID. STATEMENT IGNORED.

Example:

```
DCL 1 A, 2 B BIT(M) UNALIGNED,
         2 C BIT(N) UNALIGNED;
DCL F FILE;
READ FILE(F) INTO (B);
```

Explanation: The variable named in the INTO or FROM option cannot be an unaligned and non-varying bit string that is also based, defined, a parameter, or contained in an aggregate. Neither can it be any minor structure that starts or ends with an unaligned, non-varying bit string.

IEL0818I S INVALID SET OF OPTIONS.

INVALID SET OF OPTIONS ON RECORD I/O STATEMENT. STATEMENT IGNORED.

Example:

```
REWRITE FILE(F) EVENT(E);
```

Explanation: The input/output statement has an invalid or incomplete set of options. In the example above, the FROM option is missing.

IEL0819I I RECORD I/O FUNCTION OPTIMIZED.

RECORD I/O FUNCTION OPTIMIZED. NO LIBRARY SUBROUTINE CALL REQUIRED.

Explanation: Under certain circumstances, the compiler will generate inline code for record-oriented I/O statements.

IEL0820I S ATTRIBUTES OF D CONFLICT WITH THOSE ON 'OPEN'.

ATTRIBUTES ON 'OPEN' STATEMENT CONFLICT WITH THOSE DECLARED FOR FILE D. 'UNDEFINEDFILE' CONDITIONS MAY BE RAISED ON ATTEMPT TO OPEN THE FILE.

Example:

```
DCL F FILE INPUT;
   OPEN FILE(F) OUTPUT;
```

IEL0827I E 'PAGESIZE' OR 'LINESIZE' CONFLICT WITH ATTRIBUTES OF D.

SPECIFICATION OF 'PAGESIZE' OR 'LINESIZE' OPTIONS CONFLICTS WITH ATTRIBUTES OF FILE D.

Explanation: The PAGESIZE option can only be specified for PRINT files. The LINESIZE option can only be specified for STREAM OUTPUT files (including PRINT files).

IEL0828I S INVALID 'TITLE' 'PAGESIZE' OR 'LINESIZE' FOR D.

INVALID SPECIFICATION OF 'TITLE' 'PAGESIZE' OR 'LINESIZE' OPTION FOR FILE D. OPTION IGNORED.

Explanation: Arguments to the option PAGESIZE and LINESIZE must be expressions that represent an arithmetic value. The argument to the TITLE option must be an expression that represents a character-string value.

IEL0830I S NO ACCESS 'ENV' OPTION FOR 'DIRECT' FILE D.

'DIRECT' ATTRIBUTE BUT NO ACCESS TYPE 'ENVIRONMENT' OPTION FOR FILE D. 'UNDEFINEDFILE' MAY BE RAISED ON ATTEMPT TO OPEN THE FILE.

Example:

```
DCL D FILE DIRECT;
```

Explanation: INDEXED, REGIONAL, or VSAM must be specified in the ENVIRONMENT option for a file with the DIRECT attribute.

Programmer Response: Specify access type in ENVIRONMENT option.

IEL0834I S ATTRIBUTES AND ENVIRONMENT OPTIONS FOR D CONFLICT.

ATTRIBUTES AND 'ENVIRONMENT' OPTIONS FOR FILE D CONFLICT. 'UNDEFINEDFILE' MAY BE RAISED ON ATTEMPT TO OPEN THE FILE.

Example:

```
DCL A FILE DIRECT ENV(CONSECUTIVE...);
```

IEL0835I S INVALID ATTRIBUTES FOR D IGNORED.

INVALID ATTRIBUTE(S) FOR FILE D IGNORED.

IEL0836I S T FOR D CONFLICTS WITH PREVIOUSLY DECLARED ATTRIBUTES.

ATTRIBUTE T IN FILE D CONFLICTS WITH ONE PREVIOUSLY DECLARED AND IS IGNORED.

Example:

```
DCL F FILE STREAM;
OPEN FILE(F) PRINT INPUT;
```

IEL0837I S INVALID 'ENVIRONMENT' OPTION(S) FOR D IGNORED.

INVALID 'ENVIRONMENT' OPTION(S) FOR FILE D IGNORED.

IEL0838I S T FOR D CONFLICTS WITH PREVIOUSLY DECLARED OPTIONS.

'ENVIRONMENT' OPTION T IN FILE D CONFLICTS WITH ONE PREVIOUSLY DECLARED AND IS IGNORED.

Example:

```
DCL F FILE ENV (INDEXED F RECSIZE(80) CONSECUTIVE...);
                |                     |
                |---------------------|
                |    these conflict   |
```

IEL0840I S INVALID OPTION(S) ON 'CLOSE' FOR D.

INVALID 'ENVIRONMENT' OPTION(S) ON 'CLOSE' STATEMENT FOR FILE D. OPTION(S) IGNORED.

Explanation: The only option allowed in a CLOSE statement is the LEAVE option.

IEL0848I S INVALID 'GET' OR 'PUT' STATEMENT IGNORED.

'GET' OR 'PUT' STATEMENT REFERENCES A 'RECORD' FILE. STATEMENT IGNORED.

Explanation: A GET or PUT statement can only reference a file with the STREAM attribute.

IEL0849I E INVALID 'E' OR 'F' FORMAT ITEM.

COMPILER RESTRICTION. THE NUMBER OF DIGITS AFTER THE DECIMAL POINT IN AN 'E' OR 'F' FORMAT ITEM IS GREATER THAN N. N IS ASSUMED.

IEL0850I E INVALID 'E' OR 'F' FORMAT ITEM.

THE NUMBER OF DIGITS AFTER THE DECIMAL POINT IN AN 'E' OR 'F' FORMAT ITEM IS NEGATIVE. ZERO IS ASSUMED.

IEL0851I E CONFLICTING OPTIONS. T IGNORED.

CONFLICTING OPTIONS IN 'PUT' STATEMENT. OPTION T IGNORED.

Example:

```
PUT FILE(A) SKIP(3) PAGE;
PUT FILE(B) LINE(3) SKIP(1);
```

Explanation: The only legal combination of PAGE, SKIP, and LINE is PAGE and LINE.

IEL0852I E T VALID ONLY FOR PRINT FILES.

T VALID ONLY FOR PRINT FILES. OPTION IGNORED.

Example:

```
DCL A FILE STREAM INPUT;
OPEN FILE (A);
PUT FILE (A) LINE(3) LIST(B,C);
```

Explanation: The options LINE and PAGE are allowed only on statements referring to STREAM OUTPUT PRINT files.

IEL0853I W D NOT ARITHMETIC OR STRING.

DATA LIST ITEM D NOT ARITHMETIC OR STRING. ITEM IGNORED.

Example:

```
DCL (D,E) CHAR(8), F AREA;
GET LIST (D,E,F);
```

Explanation: Elements in a data list must have arithmetic or string data type, that is they must be problem data.

IEL0854I S CONSTANT IN 'GET' OR 'PUT' DATA LIST.

CONSTANT INVALID IN DATA LIST IN 'GET' OR 'PUT' 'DATA' STATEMENT. DATA ITEM DELETED.

Example:

```
DCL C FILE,
    (D,E) ENTRY EXT;
    PUT FILE (C) SKIP DATA (D,E);
```

IEL0855I S INVALID STRING OPTION OR GRAPHIC ITEM.

'STRING' OPTION DOES NOT CONTAIN A CHARACTER STRING VARIABLE OR GRAPHIC ITEM IS NOT ALLOWED IN 'GET' OR 'PUT' STRING. STATEMENT IGNORED.

Example:

```
DCL A FIXED BIN;
PUT STRING(A) LIST(B,C,D);
```

Explanation: The variable referred to by the STRING option must be a character string variable; or a graphic item cannot be used in a GET STRING or PUT STRING statement.

IEL0856I E NO DATA ITEM IN FORMAT LIST.

NO DATA FORMAT ITEM IN FORMAT LIST. FORMAT LIST WILL BE USED ONLY ONCE. DATA LIST IGNORED.

Example:

```
PUT EDIT(A) (X(4));
```

Explanation: Data items cannot be transmitted unless a data format item is given in the format list. No assumptions are made.

IEL0857I S CONTROL FORMAT ITEM(S) INVALID WITH 'STRING' OPTION.

INVALID CONTROL FORMAT ITEM(S) IN 'GET' OR 'PUT' STATEMENT WITH 'STRING' OPTION. FORMAT ITEM(S) IGNORED.

Example:

```
DCL (NAME, A, B) CHAR;
PUT STRING(NAME) EDIT(A,B) (F(5),PAGE,F(5));
```

Explanation: Control format items SKIP, LINE, PAGE, and COLUMN are not allowed in GET STRING or PUT STRING statements.

IEL0858I S INVALID 'A' OR 'B' FORMAT ITEM.

INVALID 'A' OR 'B' FORMAT ITEM IN 'GET' STATEMENT. 'A(1)' OR 'B(1)' ASSUMED.

Example:

```
DCL CHAR1 CHAR(8), BIT1 BIT(2);
GET EDIT(CHAR1,BIT1) (A(),B());
```

Explanation: An A-format item must be specified with an explicit width when used in GET statements.

IEL0859I W 'A' OR 'B' FORMAT ITEM INVALID IF USED BY 'GET'.

'A(1)' OR 'B(1)' ASSUMED FOR 'A' OR 'B' FORMAT ITEM IF FORMAT LIST IS USED BY A 'GET' STATEMENT.

Example:

```
F:  FORMAT (A);
    GET EDIT(CHAR) (R(F));
```

Explanation: An A-format or B-format item must be specified with an explicit width when used in GET statements.

IEL0860I E WIDTH IN FORMAT ITEM EXCEEDS N.

COMPILER RESTRICTION. WIDTH IN FORMAT ITEM IS GREATER THAN N. N ASSUMED.

Example:

```
PUT EDIT(A,B) (F(5),X(43000),F5));
```

Explanation: An A-format item cannot have a width that is greater than 32767.

IEL0861I W 'E' FORMAT ITEM WIDTH TOO SMALL FOR NEGATIVE VALUES.

'E' FORMAT ITEM HAS WIDTH TOO SMALL FOR MINUS SIGN TO BE PRINTED.

Example:

```
A = -3.57;
PUT EDIT(A) (E(8,2,3));
```

Explanation: In the above example, the resultant output should be "-3.57E+00" which is nine characters whereas the width allowed in the PUT statement is only eight characters. This will cause the minus sign to be lost.

IEL0862I S 'E' FORMAT ITEM WIDTH TOO SMALL FOR DATA.

'E' FORMAT ITEM HAS WIDTH TOO SMALL FOR COMPLETE OUTPUT OF THE ITEM. ITEM IGNORED.

Example:

```
A = -3,57;
PUT EDIT(A) (E(7,2,3));
```

Explanation: In the above example, the resultant output should be "3.57E+00" which is nine characters whereas the width allowed in the PUT statement is only seven characters. This would cause the minus sign and the most significant digit to be lost; therefore, the complete item is ignored.

IEL0863I S INVALID ARGUMENT TO 'E' OR 'F' FORMAT ITEM.

INVALID ARGUMENT TO 'E' OR 'F' FORMAT ITEM. FORMAT ITEM IGNORED.

Example:

```
PUT EDIT(A) (E(8,4,3));
```

IEL0864I E 'PAGE' OR 'LINE' IN 'GET' OR 'PUT' IGNORED.

INVALID 'PAGE' OR 'LINE' FORMAT ITEM IN 'GET' OR 'PUT' STATEMENT. FORMAT ITEM IGNORED.

IEL0865I W 'PAGE' OR 'LINE' IGNORED FOR 'GET'.

'PAGE' OR 'LINE' FORMAT ITEM WILL BE IGNORED IF FORMAT LIST IS USED BY A 'GET' STATEMENT.

IEL0866I S SECOND ARGUMENT TO T INVALID.

SECOND ARGUMENT OF BUILTIN FUNCTION T TOO LARGE OR TOO SMALL. VALUE ONE ASSUMED.

Example:

```
DCL A(3,4);
    I = HBOUND(A,3);
```

IEL0867I S INVALID ARGUMENT TO 'ALLOCATION' FUNCTION.

ARGUMENT TO 'ALLOCATION' BUILTIN FUNCTION NOT LEVEL ONE 'CONTROLLED'. FUNCTION RETURNS ZERO VALUE.

Example:

```
a.  DCL A AUTOMATIC;
         I = ALLOCATION(A);
b.  DCL 1 B, 2 C, 2 D;
         I = ALLOCATION(C);
```

Explanation: The argument to the ALLOCATION built-in function must be a level-1 controlled variable.

IEL0868I S INVALID ARGUMENT TO T.

INVALID ARGUMENT TO BUILTIN FUNCTION T. FUNCTION RETURNS NULL VALUE.

Example:

```
DCL A FIXED,
    C AREA,
    D PTR,
    E OFFSET;
D = POINTER(A,C);(1st argument
   invalid)
D = POINTER(E,A);(2nd argument
   invalid)
E = OFFSET(A,C); (1st argument
   invalid)
E = OFFSET(D,A); (2nd argument
   invalid)
```

IEL0869I S ARGUMENT TO T IS NOT A FILE.

ARGUMENT TO BUILTIN FUNCTION T IS NOT A FILE. FUNCTION RETURNS ZERO VALUE.

Example:

```
DCL X FLOAT,
    I FIXED;
    I = COUNT(X);
```

IEL0870I S T USED AS ARGUMENT DOES NOT MATCH PARAMETER DESCRIPTOR.

BUILTIN FUNCTION T USED AS ARGUMENT DOES NOT MATCH CORRESPONDING PARAMETER DESCRIPTOR. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL SIN BUILTIN,
    X ENTRY(ENTRY(FIXED));
CALL X(SIN);
```

Explanation: In the above example, the declaration of X is incorrect. X should be declared ENTRY(FLOAT...) where "..." is the precision and/or the mode. This message also applies to the declaration of a parameter for an internal procedure.

IEL0871I I FIXED POINT ARITHMETIC USED FOR T RESULT.

RESULT OF BUILTIN FUNCTION T WILL BE EVALUATED USING FIXED POINT ARITHMETIC OPERATIONS.

Explanation: This message describes a difference between the compiler implementation and that of the PL/I (F) Compiler. The F compiler converts the arguments of the SUM or PROD built-in functions to floating-point in all cases.

IEL0872I W 'ADDR' BUILTIN FUNCTION POINTS AT STRING LENGTH FIELD.

'ADDR' BUILTIN FUNCTION RETURNS A POINTER TO THE TWO-BYTE LENGTH FIELD PRECEDING THE VARYING STRING VALUE.

IEL0873I S INVALID FORMAT ITEM IGNORED.

INVALID DATA TYPE IN FORMAT ITEM. ITEM IGNORED.

Example:

```
DCL F FILE;
PUT EDIT (A) (A(F));
```

Explanation: Fields in format items are converted to fixed binary. Unless the field specification is arithmetic or string, this conversion cannot take place.

Programmer Response: Change the specification of the format item.

IEL0874I E INVALID 'SKIP' OR 'LINE' OPTION.

INVALID DATA TYPE IN 'SKIP' OR 'LINE' OPTION. VALUE ONE ASSUMED.

Example:

```
DCL F FILE;
PUT SKIP(F);
```

Explanation: The expression in a SKIP or LINE option must be convertible to a fixed decimal integer.

IEL0875I W ITEM NOT ARITHMETIC OR STRING.

DATA LIST ITEM NOT ARITHMETIC OR STRING. ITEM IGNORED.

Example:

```
DCL NULL BUILTIN;
PUT LIST(NULL);
```

Explanation: Elements in a data list must be problem data; that is, they must be arithmetic or string data. This message is produced when the data list contains a reference to a built-in function (such as NULL or OFFSET) or user-defined function returning a pointer value.

Programmer Response: Correct the source program and recompile it.

IEL0879I U COMPILATION TERMINATED IN PHASE P.

COMPILER RESTRICTION. ALL OVERFLOW TEXT PAGES FULL. COMPILATION TERMINATED IN PHASE P.

Explanation: This message can be produced if there is a high concentration of the following statements in the program:

Inline picture conversions
Concatenation
Stream I/O
DECLARE statements for arrays, having INITIAL attribute, for automatic, controlled, or based storage
Interlanguage calls
Record I/O with TOTAL option
Calls having subscripted array of structure as argument

Programmer Response: Reduce the concentration of the statements listed above by putting some of them into a DO group. The effect of this is to reorder the statements internally without changing the running order. Also change the storage class attribute from AUTOMATIC, BASED, or CONTROLLED to STATIC, for large structure declares where many fields are being initialized with the INITIAL attribute.

IEL0881I U TOO MANY SUBSCRIPTED LABELS IN THIS BLOCK.

COMPILER RESTRICTION. TOO MANY SUBSCRIPTED LABELS IN THIS BLOCK. FURTHER LABEL OPTIMIZATION INHIBITED.

Explanation: The compiler has found too many subscripted label variables and/or label prefixes in a block, and further label optimization cannot be performed.

Programmer Response: If full label optimization is required, ensure that there are less than 400 subscripted label variables and/or label prefixes in any block. If necessary, insert dummy BEGIN and END statements in your program to fulfil this condition.

IEL0882I E ARGUMENT TO 'STORAGE' OR 'CURRENTSTORAGE' MAY BE INVALID.

ARGUMENT TO 'STORAGE' OR 'CURRENTSTORAGE' MAY BE INVALID. RESULTS OF EXECUTION UNDEFINED.

Explanation: The variable specified as argument to the STORAGE or CURRENTSTORAGE built-in function is one of the following:

1. An unaligned and non-varying bit string that is also based, defined, a parameter, or contained in an aggregate
2. A minor structure that starts or ends with an unaligned non-varying bit string

Explanation: Such a variable can share delimiting bytes with adjacent bit string variables, and the byte length returned by the built-in function will be undefined.

IEL0885I W ASSIGNMENT OF STRING HAS BEEN OPTIMIZED. ENSURE STRINGS DO NOT OVERLAP.

Example:

```
DCL C CHAR(12) INIT('ABCDEFGHIJKL');
DCL C1 CHAR (8) DEF C POS (1);
DCL C2 CHAR (8) DEF C POS (5);
CALL B(C1,C2);
B:  PROC(M,N);
    DCL M CHAR (8);
    DCL N CHAR (8);
    N = M;
    END B;
```

Explanation: In certain assignments, the compiler is unable to determine whether an assignment can be performed directly without error or whether the assignment must be made via a compiler-generated temporary. When this situation arises, a temporary will always be used when the compiler option NOOPTIMIZE is specified, but the temporary will not be generated if the compiler option OPTIMIZE(TIME) is specified. This message informs the user that no temporary has been generated and that incorrect results could occur if the two variables do in fact overlap. In the example shown, if the assignment has been coded as:

M = N;

then no error would occur if no temporary was used in the assignment.

Note: This message can also be issued for picture assignments.

IEL0886I E SECOND ARGUMENT TO 'SUBSTR' SET TO ONE.

SECOND ARGUMENT OF BUILTIN FUNCTION OR PSEUDO-VARIABLE 'SUBSTR' LESS THAN ONE. VALUE SET TO ONE.

Example:

```
SUBSTRING = SUBSTR(STRING,0,7);
```

Explanation: The second argument of the SUBSTR built-in function must be greater than or equal to 1.

IEL0887I E SECOND ARGUMENT TO 'SUBSTR' TOO LARGE.

SECOND ARGUMENT OF BUILTIN FUNCTION OR PSEUDO-VARIABLE 'SUBSTR' GREATER THAN STRING LENGTH. NULL STRING RETURNED.

Example:

```
DCL STRING CHAR(6);
SUBSTRING = (SUBSTR(STRING,7,J));
```

Explanation: The value of the second argument of the SUBSTR built-in function must be less than or equal to the length of the string in the first argument.

IEL0888I E THIRD ARGUMENT TO 'SUBSTR' NEGATIVE.

THIRD ARGUMENT OF BUILTIN FUNCTION OR PSEUDO-VARIABLE 'SUBSTR' NEGATIVE. NULL STRING RETURNED.

Example:

```
SUBSTRING = SUBSTR(STRING,I,-1);
```

Explanation: The third argument of the SUBSTR built-in function must be greater than or equal to zero.

IEL0889I E THIRD ARGUMENT TO 'SUBSTR' TOO LARGE.

THIRD ARGUMENT TO BUILTIN FUNCTION OR PSEUDO-VARIABLE 'SUBSTR' GREATER THAN STRING LENGTH. RETURNED VALUE TRUNCATED AT END OF SOURCE STRING

Explanation: The third argument of the SUBSTR built-in function must be less than or equal to the length of the string in the first argument.

IEL0890I E ARGUMENTS TO 'SUBSTR' TOO LARGE.

THE SUM OF THE SECOND AND THIRD ARGUMENTS OF BUILTIN FUNCTION OR PSEUDO-VARIABLE 'SUBSTR' IS GREATER THAN THE STRING LENGTH PLUS ONE. RETURNED VALUE TRUNCATED AT END OF SOURCE STRING.

Example:

```
DCL STRING CHAR(6)
SUBSTRING = SUBSTR (STRING,6,2);
```

Explanation: The value of the first argument plus the value of the second argument, less one, must be less than or equal to the length of the string in the first argument.

IEL0892I W RESULT OF STRING OPERATION TRUNCATED.

TARGET STRING SHORTER THAN SOURCE. RESULT TRUNCATED ON ASSIGNMENT.

Example:

```
DCL B1 BIT(5),
    (B2,B3) BIT(7)
B1 = B2;
```

Explanation: This message warns of a possible error caused by the loss of truncated bits when the assignment takes place. If this message is issued, the STRINGSIZE condition will not be raised at run-time, even though it might be enabled by a condition prefix.

IEL0903I S INVALID ARGUMENT TO 'HIGH' OR 'LOW' REPLACED BY '(1)'.

INVALID ARGUMENT TO 'HIGH' OR 'LOW' BUILTIN FUNCTION. '(1)' ASSUMED.

IEL0904I S OPERATOR(S) INVALID FOR 'COMPLEX' DATA.

OPERATOR(S) INVALID FOR 'COMPLEX' DATA. '=' ASSUMED.

Example:

```
DCL (A,B) COMPLEX;
IF A > B THEN GOTO...;
```

Explanation: Operators allowed for use with complex data are limited to "=" and "¬=" (equals and not-equals) operators.

IEL0905I S EXPRESSION IN 'INITIAL' FOR STATIC VARIABLE D.

SPECIFICATION OF 'INITIAL' ATTRIBUTE FOR STATIC VARIABLE D CONTAINS EXPRESSION. 'INITIAL' ATTRIBUTE IGNORED.

Example:

```
DCL A(3) STATIC INIT(1,2,3I);
```

IEL0906I I CONVERSION WILL BE DONE BY SUBROUTINE CALL.

DATA CONVERSION WILL BE DONE BY SUBROUTINE CALL.

Explanation: The program contains one or more conversions that will require a PL/I library subroutine. The message indicates where the program can be more efficient if it is written so the conversion is performed by compiler-generated instructions.

IEL0907I S WRONG NUMBER OF ARGUMENTS FOR ENTRY D.

WRONG NUMBER OF ARGUMENTS SPECIFIED FOR FUNCTION OR CALL D. RESULTS OF EXECUTION UNDEFINED.

Example:

```
DCL P ENTRY(FLOAT,FLOAT) EXTERNAL;
 1. CALL P(A);
 2. A = P(A,A,A);
```

Explanation: A procedure has been referenced with a number of arguments different from the number in the parameter descriptor.

Programmer Response: Correct the source.

IEL0908I W 'RETURN' EXPRESSION MAY CONFLICT WITH ENTRY SPECIFICATION.

DATA TYPE OF RETURNED EXPRESSION CONFLICTS WITH 'RETURNS' OPTION OF AN ENTRY SPECIFICATION IN THIS BLOCK. 'RETURN' EXPRESSION WILL BE IGNORED IF THE INVALID COMBINATION OF 'RETURN' AND ENTRY IS USED.

Example:

```
P: PROC RETURNS(FILE);
E: ENTRY RETURNS(DEC FLOAT);
   DCL F DEC FLOAT,
       G FILE;
   RETURN(F);
   RETURN(G);
```

In this example, the first RETURN statement conflicts with the PROC statement and will be treated as a RETURN without an expression if run during an invocation of P. Similarly, the expression in the second RETURN statement will be ignored if run during an invocation of E.

IEL0909I I DATA VARIABLE USED FOR PROGRAM CONTROL.

BASED REFERENCE TO PROGRAM CONTROL DATA REFERS TO STORAGE USED BY VARIABLE D BUT IS ACCEPTED AS VALID.

Explanation: The global* optimization process must include analysis of all possible values of label variables, entry variables, and pointers in the program before it can attempt to perform move-out and strength-reduction. During the process, the second condition in the example above would be detected. This condition would restrict the global* optimization process, since this process cannot detect all the possible label constant values that might be assigned to FLOAT.

*Global optimization is defined in the explanation for IEL0910I.

IEL0910I W TOO MANY CALLS AND FUNCTION REFERENCES FOR OPTIMIZATION.

COMPILER RESTRICTION. TOO MANY 'CALL' STATEMENTS AND FUNCTION REFERENCES. OPTIMIZATION IS INHIBITED FOR THE PROGRAM.

Explanation: A program that is to be compiled with full optimization (with OPT(TIME) specified) has so many branches of control between blocks that the capacity of the compiler to analyze them has been exceeded. The compilation is completed without global optimization; some local optimization might have been performed. In this context, *local optimization* includes such things as the inline simplification of calculations such as I*3 and A**4, and the matching of data items with format items in edit I/O. Conversely, *global optimization* is concerned with common expression elimination, the moving of invariant expressions from loops, and further simplification of expressions. If full global optimization is performed, then any or all of the types of optimization can be carried out, either within or between flow units (logical divisions of the PL/I source program). If certain compiler limitations are exceeded, then global optimization is restricted to common expression elimination alone. Furthermore, this is performed solely *within* flow units.

The compiler allows up to 256 separate CALL statements and function references involving both entry constants and entry variables. This limit includes each entry constant or entry variable passed as an argument to an external procedure. A further limit of 2048 exists for all possible assignments of entry constants to entry variables.

Programmer Response: To obtain global optimization it is necessary to simplify the program's structure so that the number of branches between begin blocks and between internal procedures is kept within the limits described above.

IEL0911I W TOO MANY LOCATOR LABEL OR ENTRY ASSIGNMENTS FOR OPTIMIZATION.

COMPILER RESTRICTION. TOO MANY LOCATOR LABEL OR ENTRY VARIABLE ASSIGNMENTS. OPTIMIZATION IS INHIBITED FOR THE PROGRAM.

Explanation: A program that is to be compiled with global* optimization (with OPT(TIME) specified) has so many locator and entry variable assignments that the capacity of the compiler to analyze them has been exceeded. The compilation is completed without global optimization.

The compiler allows up to 1360 locator, label, or entry variable assignments without inhibiting optimization.

* Local and global optimization are defined in the explanation for IEL0910I.

Programmer Response: To obtain global optimization it is necessary to reduce the number of locator and entry variable assignments that appear in the source program.

IEL0912I W TOO MANY BASED LOCATOR LABEL OR ENTRY ASSIGNMENTS FOR OPTIMIZATION.

COMPILER RESTRICTION. TOO MANY ASSIGNMENTS WITH BASED LOCATORS LABEL OR ENTRY VARIABLES. OPTIMIZATION IS INHIBITED FOR THE PROGRAM.

Explanation: A program that is to be compiled with global* optimization (with OPT(TIME) specified) has so many based locator, based label, and based entry variable assignments that the capacity of the compiler to analyze them has been exceeded. The compilation is completed without global optimization.

The compiler allows up to 680 based locator, label or entry variable assignments without inhibiting optimization.

* Local and global optimization are defined in the explanation for IEL0910I.

Programmer Response: To obtain global optimization it is necessary to reduce the number of based locator, based label, and based variable assignments that appear in the source program.

IEL0913I W TOO MANY LOCATOR TEMPORARIES ACTIVE FOR OPTIMIZATION.

COMPILER RESTRICTION. TOO MANY LOCATOR TEMPORARIES ACTIVE. OPTIMIZATION IS INHIBITED FOR THE PROGRAM.

Explanation: The compiler creates a "locator temporary" for functions that return locator values. A program that is to be compiled with global* optimization (with OPT(TIME) specified) has so many of these locator temporaries created that the capacity of the compiler to analyze them has been exceeded. The compilation is completed without global optimization.

The compiler allows up to 10 locator temporaries without inhibiting optimization.

* Local and global optimization are defined in the explanation for IEL0910I.

Programmer Response: To obtain global optimization it is necessary to reduce the number of locator values returned by functions that appear in the source program.

IEL0914I W STATEMENT MAY NEVER BE EXECUTED.

Example:

```
P:  PROC;
DCL X, Y CHAR(1);
IF X = '2' THEN Y = '';
   ELSE GOTO L2;
GOTO L2;
L1:  A = 5;
L2:  B = 6;
END P;
```

In this example, the message will be produced for the statement labeled L1, since there is no possibility of control being transferred to it.

The statement "GOTO L2;" in this example can be run. However, the optimization process has modified the THEN clause to branch directly to the label constant L2 rather than to the statement following the ELSE clause. The message is then produced and the redundant statement is eliminated.

Explanation: This message warns that the compiler has detected a statement that can never be run as the flow of control must always pass by it.

IEL0915I W TOO MANY STATEMENT LABEL CONSTANTS FOR OPTIMIZATION.

COMPILER RESTRICTION. TOO MANY STATEMENT LABEL CONSTANTS. OPTIMIZATION IS INHIBITED FOR THE PROGRAM.

Explanation: A program that is to be compiled with global* optimization (with OPT(TIME) specified) has so many statement label constants that the capacity of the compiler to analyze them has been exceeded. The compilation is completed without global optimization.

The compiler allows up to 2048 statement label constants without inhibiting optimization.

* Local and global optimization are defined in the explanation for IEL0910I.

Programmer Response: To obtain global optimization it is necessary to reduce the number of statement label constants used in the source program.

IEL0916I W ITEM(S) D MAY BE UNINITIALIZED.

ITEM(S) D MAY BE UNINITIALIZED WHEN USED IN THIS BLOCK.

Example:

```
P: PROC;
   DCL X;
   Z = X;
   END P;
```

Explanation: This message refers only to variables declared within the block. The flow-analysis stage of optimization checks all possible flow-paths through a program although many of the possible flow-paths might never be used. In doing so, the flow analysis determines flow-paths originating from statements prefixed by label constants that can be branched to from on-units, as well as those that originate from PROCEDURE and ENTRY statements.

It is possible, therefore, that this message is produced for items that are initialized correctly for the flow-paths that will actually be used owing to the presence of other flow-paths that will never be used. This is aggravated by the necessity to consider label constants as external entry points. In the following example, an ON-unit returns control to a block by means of a GOTO statement. The variable X is detected as uninitialized if the block is entered through the label constant Y, although it might have been initialized before the ON-unit was entered.

Example:

```
P: PROC;
   X = 100;
   ON OFL GOTO Y;
   ⋮
Y: A = X;
   ⋮
```

The final value assigned to a static variable in one invocation of a procedure will be the 'initial' value of that variable in a subsequent invocation of that procedure.

IEL0917I W N FLOW UNITS IN BLOCK. GLOBAL OPTIMIZATION RESTRICTED.

BLOCK CONTAINS N FLOW UNITS. GLOBAL OPTIMIZATION PERFORMED ONLY IN DO GROUPS.

Explanation: The block has been split into flow units for the purposes of global* optimization. However, the compiler limit of 255 flow units in a block has been exceeded, and consequently, global optimization is restricted. Before scanning to the next block, the compiler looks for DO-groups in the current block, in the hope that flow analysis (and full global optimization) can be completed for these.

* Local and global optimization are defined in the explanation for IEL0910I.

Programmer Response: If full optimization is required for the block, either simplify the flow of control within the block, or divide the block into two or more simpler blocks.

IEL0918I W GO TO D MAY PASS CONTROL OUT OF BLOCK.

GO TO D MAY CAUSE CONTROL TO BE PASSED OUT OF THE CURRENT BLOCK.

Explanation: D is a label variable declared STATIC and INITIAL. Since the initialization is done at compile time, no environment information can be supplied to the label variable; since it has been detected that control might be passed out of the current block, the GOTO is run by the library. This will cause a run-time error. If this message appears, message IEL0580I (severity E) will have been produced for the specified label variable.

Programmer Response: Redeclare the LABEL variable as AUTOMATIC.

IEL0919I W N VARIABLES IN PROGRAM. GLOBAL OPTIMIZATION RESTRICTED.

N VARIABLES IN PROGRAM. GLOBAL OPTIMIZATION PERFORMED FOR 255 VARIABLES. LOCAL OPTIMIZATION PERFORMED ON REMAINDER.

Explanation: The compiler will consider 255 variables in the program for global* optimization. The remainder are considered solely for local* optimization.

Explicitly declared variables will be considered for global optimization in preference to contextually declared variables, and the latter will in turn be considered in preference to implicitly declared variables. Furthermore, the highest preference will be given to those variables declared in the final DECLARE statements in the outermost block.

If the program contains more than 255 variables, most benefit will be obtained from the global optimization of arithmetic variables, particularly DO loop control variables and subscripting variables. Little or no benefit will be gained from the optimization of string variables or program control data.

Arithmetic variables should not, therefore, be implicitly declared but should be declared in the final DECLARE statements in the outermost block. Further benefits can be obtained if declared but unreferenced variables are eliminated from the program.

* Local and global optimization are defined in the explanation for IEL0910I.

IEL0920I W N FLOW UNITS IN DO GROUP. GLOBAL OPTIMIZATION RESTRICTED.

DO GROUP CONTAINS N FLOW UNITS. GLOBAL OPTIMIZATION IS RESTRICTED.

Explanation: The compiler limit of 255 flow units in a DO-group has been exceeded and full global* optimization is inhibited within the group. Partial global optimization will be performed for flow units within the group.

* Local and global optimization are defined in the explanation for IEL0910I.

IEL0921I E LESS THAN N CHARACTERS OF T IN D PRINTED.

QUALIFIED NAME OF ELEMENT T OF STRUCTURE D WILL BE TRUNCATED TO LESS THAN N CHARACTERS IN DATA DIRECTED I/O.

Example:

```
PUT DATA (PAYROLL);
```

where PAYROLL is declared as a base element of a structure which when fully qualified exceeds 255 characters, including periods.

IEL0923I E D INVALID TYPE IN DATA LIST FOR DATA DIRECTED I/O OR CHECK.

COMPILER RESTRICTION. TYPE OF 'BASED' VARIABLE D IN DATA LIST NOT SUPPORTED FOR DATA DIRECTED I/O OR CHECK. ITEM IGNORED.

Example:

```
DCL 1 STR BASED(P),
    2 LEN FIXED BIN,
    2 TITLE CHAR(N REFER(LEN));
PUT DATA(TITLE);
```

Explanation: The compiler does not allow PUT DATA and GET DATA statements or the CHECK prefix option on certain types of based variables. This based variable in the DATA list or CHECK list will be ignored. The syntax of the CHECK condition is still analyzed at compile time; however, the CHECK condition is no longer supported and is always disabled at run time. See message IEL0756I for additional CHECK condition details. See the *PL/I VSE Language Reference* for information about data-directed I/O.

IEL0924I E D INVALID TYPE IN DATA LIST FOR DATA DIRECTED I/O OR CHECK.

COMPILER RESTRICTION. TYPE OF 'DEFINED' VARIABLE D IN DATA LIST NOT SUPPORTED FOR DATA DIRECTED I/O OR CHECK. ITEM IGNORED.

Example:

```
DCL A CHAR(100),
    B CHAR(10) DEF A POS(N);
PUT DATA(B);
DCL E CTL,
    F DEF E;
PUT DATA(F);
```

Explanation: The compiler does not allow the transmission of the following types of defined variables by means of the PUT DATA statement or the CHECK prefix option:

1. A string-overlay defined item
2. An iSUB-defined item
3. An item defined on a controlled base variable

This defined variable in the DATA list or CHECK list will be ignored.

IEL0925I W GLOBAL OPTIMIZATION RESTRICTED.

FLOW WITHIN BLOCK OR DO GROUP IS TOO COMPLEX. GLOBAL OPTIMIZATION IS RESTRICTED.

Explanation: The block or DO-group has been split into flow units for the purposes of flow analysis and global* optimization. However, the compiler limit of 1024 connections between flow units has been exceeded, and consequently, global optimization has been restricted within the block or group. Partial global optimization will be performed for flow units within the block or group.

* Local and global optimization are defined in the explanation for IEL0910I.

IEL0926I S 'SIZE' RAISED WHEN CONVERTING CONSTANT [TO D].

'SIZE' CONDITION RAISED WHEN CONVERTING CONSTANT [TO D]. RESULT OF CONVERSION UNDEFINED.

Example:

```
DCL A FIXED DECIMAL(2,0);
    A = 999;
```

Explanation: A constant converted at compile-time has raised the SIZE condition.

IEL0927I S 'CONVERSION' RAISED WHEN CONVERTING CONSTANT [TO D].

'CONVERSION' CONDITION RAISED WHEN CONVERTING CONSTANT [TO D]. RESULT OF CONVERSION UNDEFINED.

Example:

```
READ FILE(BERT) IGNORE('JACK AND JIM');
```

Explanation: The IGNORE option should refer to an arithmetic integer value.

IEL0928I U STATIC STORAGE EXCEEDS 16777216 BYTES.

THE SIZE OF STATIC STORAGE REQUIRED FOR THIS PROGRAM EXCEEDS 16777216 BYTES.

Explanation: STATIC storage is limited to 1677216 bytes because STATIC is kept in a CSECT. The maximum size of a CSECT is 1677216 bytes.

This message can be issued when the TEST compile-time option is specified with the (SYM) suboption, since symbol tables for INTERNAL variables and program control constants are built in internal STATIC storage.

Programmer Response: Check the declarations for STATIC arrays and structures, and reduce the size of the extents that have been specified.

If you got this message while using TEST(SYM), you can reduce symbol table space by eliminating unreferenced INTERNAL variables from the program. You can also distribute the variables used by the external procedure among two or more external procedures to this problem.

IEL0929I U AUTOMATIC STORAGE EXCEEDS 16777216 BYTES.

THE SIZE OF AUTOMATIC STORAGE REQUIRED FOR THIS BLOCK EXCEEDS 16777216 BYTES.

Explanation: The size of the initial stack (AUTOMATIC) storage for a block is limited to 16777216 bytes. Stack extensions are also each constrained to 16777216 bytes. This means the size of an AUTOMATIC aggregate, temporary variable or dummy argument may not exceed 16M. Violation of this constraint may result in unpredictable results.

Note that the initial stack storage for a block includes some overhead for internal housekeeping storage. Thus, the maximum stack storage size of 16777216 is not all available for user AUTOMATIC variables.

Programmer Response: Check the declaration for AUTOMATIC arrays and structures in the identified block, and reduce the size of extents that have been specified. Alternatively, change the storage class of the array or structure to BASED or CONTROLLED.

IEL0930I E COMPILER RESTRICTION. PART OF STATIC STORAGE EXCEEDS 64K BYTES.

Explanation: Static storage is divided into a series of regions. Each region contains different categories of either constants or variables. If one of the regions containing constants, for example the region containing symbol tables, exceeds 65535 bytes in size, then it might be impossible for the compiler to address some of the constants.

This message can be issued when the TEST compile-time option with the suboption (SYM) is specified, since additional internal STATIC storage might be required to build DEDs and locators or locator/descriptors for variables.

Programmer Response: If the program contains numerous variables and also contains a GET DATA or PUT DATA statement without any qualifying list, then remove the statement or replace it by statement containing a data list.

Alternatively split the external procedure into two or more external procedures.

If you received this message while using the TEST compile-time option, you can reduce static storage space by eliminating unreferenced INTERNAL variables from the program. You can also distribute the variables used by the external procedure among two or more external procedures to eliminate this problem.

IEL0931I E LENGTH OF D EXCEEDS LENGTH OF 'DEFINED' BASE.

LENGTH OF VARIABLE D EXCEEDS LENGTH OF VARIABLE ON WHICH IT IS DEFINED. THIS DEFINING HAS BEEN ACCEPTED.

Example:

```
DCL A CHAR(6);
DCL B CHAR(10) DEF A;
```

Explanation: The compiler will accept this invalid form of defining to allow running of programs that require it. However, it is possible that an assignment to the defined item will cause storage to be overwritten and an unpredictable error to occur.

Programmer Response: If this defining is required, check that any conditional link-editing and run steps will not be inhibited.

IEL0932I S AGGREGATE DESCRIPTOR FOR STRUCTURE TOO LARGE.

COMPILE RESTRICTION. AGGREGATE DESCRIPTOR FOR STRUCTURE TOO LARGE. RESULTS OF EXECUTION UNDEFINED.

Explanation: An aggregate descriptor is a control block created by the compiler to handle the addressing of the base elements in an aggregate. Its format is described in the *PL/I VSE Diagnosis Guide*. A large number of base elements in a large aggregate has caused the aggregate descriptor to exceed the limit of its internal addressability.

IEL0933I W D INVALID TYPE FOR DATA DIRECTED I/O OR CHECK.

COMPILER RESTRICTION. TYPE OF 'BASED' VARIABLE D NOT SUPPORTED FOR DATA DIRECTED I/O OR CHECK. ITEM IGNORED.

Example:

```
DCL 1 STR BASED(P),
        2 LEN FIXED BIN,
        2 TITLE CHAR(N REFER(LEN));
PUT DATA;
```

Explanation: The compiler does not allow PUT DATA and GET DATA statements or the CHECK prefix option on certain types of based variables. See the *PL/I VSE Language Reference* for details.

IEL0934I W D INVALID TYPE FOR DATA DIRECTED I/O OR CHECK.

COMPILER RESTRICTION. TYPE OF 'DEFINED' VARIABLE D NOT SUPPORTED FOR DATA DIRECTED I/O OR CHECK. ITEM IGNORED.

Example:

```
DCL A CHAR(100), B CHAR(10)
    DEF A POS(N);
DCL C(100,100)CHAR(1);
DCL D(10,10) CHAR(1)
    DEF C(1SUB,2SUB);
DCL E CTL,F DEF E;
PUT DATA;
```

Explanation: The compiler does not allow the transmission of the following types of defined variables by means of the PUT DATA statement or the CHECK prefix option:

1. A string-overlay defined item
2. An iSUB-defined item
3. An item defined on a controlled base variable

IEL0935I U THE SIZE REQUIRED FOR ADDRESSING CONSTANTS IN STATIC STORAGE FOR THIS PROGRAM EXCEEDS 4095 BYTES. COMPILATION TERMINATED.

Explanation: The compiler cannot generate correct addressing code if the addressing constants at the start of static storage occupy more than 4095 bytes. The addressing constants in question consist of three for each procedure, ON-unit, and begin block, and one for each additional entry point, plus address constants for library routines, branching within the object program, and address constants to address the remainder of static storage beyond 4K.

This message can also be issued when the TEST compile-time option is specified, because debugging hooks and the symbol table locator/descriptor initialization code increases the size of the program CSECT. This in turn may increase the number of addressing constants for branching within the program.

Programmer Response: Split the external procedure into two or more external procedures, or change some large static variables to CONTROLLED or AUTOMATIC.

If you received this message while using the TEST compile-time option, you can reduce the program CSECT size by specifying a TEST suboption that causes fewer debugging hooks to be generated. You can also split the external procedure into two or more external procedures to eliminate this problem.

IEL0936I S MULTITASKING FEATURE IS NOT SUPPORTED.

MULTITASKING FEATURE IS NOT SUPPORTED. RESULTS OF EXECUTION UNDEFINED.

Example:

```
CALL X TASK(T);
```

Explanation: The tasking feature is not supported in this release. The compiler provides limited diagnosis and issues this message (only) when:

- A CALL statement with the EVENT, PRIORITY or TASK option is detected.
- The PRIORITY built-in or pseudovariable is used. This includes implicit PRIORITY usage via TASK assignment.

The compiler provides no diagnostic for tasking programs that use task declaration, comparison or procedure options(task).

IEL0940I W T MAY INCREASE EXECUTION TIME.

T CONFLICTS WITH THE OPTIMIZE OPTION. EXECUTION TIME MAY BE INCREASED.

Explanation: The INTERRUPT option and the STRINGRANGE, SUBSCRIPTRANGE, SIZE and STRINGSIZE conditions are program debugging aids causing many extra machine instructions to be generated and executed. Their use is inconsistent with the use of the OPT(TIME) option, which specifies that the compiler is to optimize the generated machine instructions in order that a very efficient program can be produced.

Programmer Response: Remove the INTERRUPT option and/or disable the conditions if the full benefit of optimization is to be obtained.

IEL0950I W 'PLIXOPT' STRING IS INVALID. SEE RELATED RUNTIME MESSAGE T.

'PLIXOPT' STRING IS INVALID. SEE RELATED RUNTIME MESSAGE T.

Explanation: The PLIXOPT INITIAL string could not be parsed.

Programmer Response: Refer to the *LE/VSE Debugging Guide and Run-Time Messages* for an explanation of the equivalent run-time message. Correct the PLIXOPT INITIAL string.

IEL0951I W T IN 'PLIXOPT' STRING IS INVALID. SEE RELATED RUNTIME MESSAGE T.

T IN 'PLIXOPT' STRING IS INVALID. SEE RELATED RUNTIME MESSAGE T.

Explanation: The PLIXOPT INITIAL string contains an item which is not recognized as a valid run-time option.

Programmer Response: Refer to the *LE/VSE Debugging Guide and Run-Time Messages* for an explanation of the equivalent run-time message. Correct the invalid run-time option.

IEL0952I I T IN 'PLIXOPT' STRING IS NOT SUPPORTED. SEE RELATED RUNTIME MESSAGE T.

T IN 'PLIXOPT' STRING IS NOT SUPPORTED. SEE RELATED RUNTIME MESSAGE T.

Explanation: The PLIXOPT INITIAL string contains a run-time option which is not supported by LE/VSE.

Programmer Response: Refer to the *LE/VSE Debugging Guide and Run-Time Messages* for an explanation of the equivalent run-time message. Correct the unsupported run-time option.

IEL0953I I 'SPIE' OR 'STAE' IN 'PLIXOPT' STRING IS NOT SUPPORTED. SEE RELATED RUNTIME MESSAGE T.

'SPIE' OR 'STAE' IN 'PLIXOPT' STRING IS NOT SUPPORTED. SEE RELATED RUNTIME MESSAGE T.

Explanation: The SPIE and STAE options have been replaced by the TRAP option. TRAP(ON) is equivalent to SPIE and STAE; TRAP(OFF) is equivalent to NOSPIE and NOSTAE. The combination SPIE and NOSTAE and the combination NOSPIE and STAE are no longer supported.

Programmer Response: Refer to the *LE/VSE Debugging Guide and Run-Time Messages* for an explanation of the equivalent run-time message. Replace SPIE and STAE with the TRAP option.

IEL0954I W 'PLIXHD' NOT DECLARED AS SCALAR 'CHARACTER' AND 'VARYING'.

'PLIXHD' NOT DECLARED AS SCALAR 'CHARACTER' AND 'VARYING'. RESULTS OF EXECUTION UNDEFINED.

Explanation: A level 1 static external variable called PLIXHD is not declared as scalar and CHARACTER VARYING, rendering it unsuitable for user-identification of REPORT and COUNT output. Such a declaration will give rise to unpredictable results at run time if either REPORT or COUNT output is required.

IEL0955I E 'PLIXOPT' NOT DECLARED AS SCALAR 'CHARACTER' AND 'INITIAL'.

'PLIXOPT' NOT DECLARED AS SCALAR 'CHARACTER' AND 'INITIAL'. RESULTS OF EXECUTION UNDEFINED.

Explanation: A level 1 static external variable called PLIXOPT is not declared as scalar and CHARACTER INITIAL, rendering it unsuitable for run-time options. Such a declaration will give rise to unpredictable results on program initialization at run-time.

IEL0956I W 'PLIXOPT' NOT DECLARED 'VARYING'.

'PLIXOPT' NOT DECLARED 'VARYING' BUT ACCEPTED.

Explanation: The VARYING attribute is omitted from an otherwise suitable declaration of a variable called PLIXOPT for run-time options. However, the variable is accepted for compile-time analysis of the associated initial string.

IEL0957I E 'PLIXOPT' 'INITIAL' STRING HAS LENGTH OUTSIDE PERMITTED RANGE.

'PLIXOPT' 'INITIAL' STRING HAS LENGTH OUTSIDE PERMITTED RANGE. NO EXECUTION OPTIONS PROCESSED.

Explanation: The length of PLIXOPT initial string is outside the allowed range 0< length ‰250. No compile-time analysis of PLIXOPT takes place and message IBM016I—PLIXOPT not a valid execution-time options string—will result on program initialization at run-time.

IEL0958I E NO VALID OPTIONS IN 'PLIXOPT' 'INITIAL' STRING.

NO VALID OPTIONS IN 'PLIXOPT' 'INITIAL' STRING. NO EXECUTION OPTIONS PROCESSED.

Explanation: No valid options have been found during compile-time analysis of PLIXOPT initial string. MESSAGE IBM016I—PLIXOPT not a valid execution-time options string—will result in program initialization at run time.

IEL0959I E ONE OR MORE INVALID OPTIONS IN 'PLIXOPT' 'INITIAL' STRING.

ONE OR MORE INVALID OPTIONS IN 'PLIXOPT' 'INITIAL' STRING. ONLY VALID EXECUTION OPTIONS PROCESSED.

Explanation: One or more invalid options have been found and ignored during compile-time analysis of PLIXOPT initial string. MESSAGE IBM017I—Erroneous option in PLIXOPT has been ignored—will result in program initialization at run time.

IEL0960I W GENERATED EXTERNAL NAMES MAY BE AMBIGUOUS.

COMPILER GENERATED EXTERNAL NAMES MAY BE AMBIGUOUS IF THE PROGRAM IS LINK-EDITED WITH A PROCEDURE OF SIMILAR NAME.

Explanation: The compiler will generate names for internal controlled variables and internal files, if used. These names are processed by the linkage editor. If two external PL/I procedures with similar names, such as ATESTER and BTESTER are to be link-edited together, it is possible for both procedures to have the name /TESTER1 generated for them. However, this cannot occur unless both procedures have at least 36 generated names each.

IEL0961I S STATEMENT TOO LARGE.

COMPILER RESTRICTION. STATEMENT TOO LARGE. RESULTS OF EXECUTION UNDEFINED.

Explanation: The size of the statement can force the compiler to generate a set of instructions that exceeds 4096 bytes of storage. The use of an RX branch instruction does not allow an offset that exceeds 4096. Consequently running of the statement can produce unpredictable errors.

Programmer Response: Divide the statement into two or more smaller statements.

IEL0964I E EXTERNAL NAME D MAY CAUSE ERROR IN EXECUTION.

Explanation: The compiler has detected two or more CSECTs with the same link-edit name for this compilation. The linkage editor resolves all references to this name using the first encountered CSECT with this name. Running yields unpredictable results. (Currently, this message is issued only if the link-edit name is SYSPINT.)

IEL0966I W EXTERNAL NAME D EXCEEDS N CHARACTERS. IT IS SHORTENED TO T.

COMPILER RESTRICTION. EXTERNAL NAME D EXCEEDS N CHARACTERS. IT IS SHORTENED TO T.

Example:

```
DCL ABCDEFGHI FILE...;
```

Explanation: Since external identifiers in PL/I are resolved by the linkage editor, it follows that such names should not exceed the limit imposed by the linkage editor on the length of names. The method of truncation used by the compiler will, in many cases, create unique identifiers so that the compilation can continue, and link-editing and running can be successful.

IEL0967I W D EXCEEDS N CHARACTERS. IT IS TRUNCATED TO T.

COMPILER RESTRICTION. EXTERNAL ENTRY NAME D WITH INTERLANGUAGE OPTION EXCEEDS N CHARACTERS. IT IS TRUNCATED TO T.

Explanation: In PL/I the usual method of truncating external names is to concatenate the first four and last three characters to form a seven-character identifier. External names for COBOL, FORTRAN, and ASSEMBLER routines can be up to eight characters in length, and any truncation of names of greater length than this involves the removal of the excess characters. To allow interlanguage communication, PL/I adopts this technique for identifiers that are associated with COBOL, FORTRAN, or ASSEMBLER routines.

IEL0968I U OVERFLOW CONDITION RAISED WHEN CONVERTING CONSTANT.

OVERFLOW CONDITION RAISED WHEN CONVERTING CONSTANT WHICH IS OUTSIDE ALLOWED RANGE.

Explanation: Floating-point constant is outside the range and cannot be converted to its true value. (Absolute value exceeds 7.23700E75.)

Programmer Response: Verify that constant is in correct range. See the *PL/I VSE Language Reference* for information on floating-point numbers.

IEL0969I E NO LABEL ON 'FORMAT' STATEMENT.

'FORMAT' STATEMENT HAS NO LABEL. STATEMENT IGNORED.

Example:

```
F: ;
FORMAT(A);
```

IEL0970I U COMPILER CANNOT PROCEED. ERROR N DURING PHASE P. CORRECT SOURCE AND RE-COMPILE.

COMPILER ERROR NUMBER N DURING PHASE P. COMPILER UNABLE TO PROCEED. CORRECTION OF SOURCE ERRORS MAY LEAD TO SUCCESSFUL COMPILATION.

Explanation: Errors have prevented successful compilation. A detailed explanation of error number N is given "Error and restriction numbers (0 to 946) for IEL0001I, IEL0230I, and IEL0970I" on page 104 in this chapter.

Programmer Response: Correct the errors indicated by other messages and recompile the program.

IEL0971I W FIRST USE OF OPTION T FOR FILE D IGNORED.

ENVIRONMENT OPTION T SPECIFIED MORE THAN ONCE IN DECLARATION OF FILE D. FIRST USE IGNORED.

Example:

```
DCL A FILE ENV (RECSIZE(20)RECIZE(20));
```

IEL0982I E ARGUMENT TO 'CMDCHN' ENVIRONMENT OPTION FOR FILE D IS INVALID. 'CMDCHN(1)' IS ASSUMED.

Example:

```
DCL TAA ENV(MEDIUM(SYS004) CMDCHN(4));
```

Explanation: The value of CMDCHN must be 1, 2, 13, or 26.

Programmer Response: Specify 2, 13, or 26 if 1 is unacceptable.

IEL0983I W EXTERNAL NAME D EXCEEDS N CHARACTERS.

EXTERNAL NAME D EXCEEDS N CHARACTERS. EXECUTION IS UNDEFINED IF D IS THE SAME AS A COMPILER GENERATED NAME.

Explanation: An 8-character external entry name has been specified. It has been accepted without truncation. However, if the name used is the same as a name that is generated by the compiler during this compilation, then unexpected results can occur.

IEL0985I W EXTERNAL ENVIRONMENT NAME T TRUNCATED TO FIRST 8 CHARACTERS.

EXTERNAL ENVIRONMENT NAME T1 IS TOO LONG. NAME TRUNCATED TO T2 USING FIRST 8 CHARACTERS.

Example:

```
DCL X  ENTRY EXT('A12345678') OPTIONS(ASSEMBLER);
```

Explanation: The external environment name has a maximum permitted length of 8 characters. The compiler truncates the name by using the first 8 characters. The user should ensure that the truncated name is not the same as a name that is generated by the compiler during the compilation. If this occurs, then unexpected results can occur. Names generated by the compiler can be examined by specifying the ESD compiler option.

IEL0989I I RECORD I/O FUNCTION PERFORMED BY SUBROUTINE CALL.

'TOTAL' OPTION SPECIFIED BUT RECORD I/O FUNCTION PERFORMED BY SUBROUTINE CALL.

Example:

```
DCL X RECORD ENV(FB,RECSIZE(N),TOTAL),
    Y CHAR(80),
    N FIXED BINARY(31,0) STATIC;
READ FILE(X) INTO(Y);
```

Explanation: A record I/O statement is performed in-line only if the TOTAL option is specified and all the environment options are known at the time of compilation. In the example shown, the record size of file X was declared as "N", and was thus not known at the time of compilation. Therefore, although the TOTAL option was specified, the READ statement must be performed by a library call. The message can also be produced if the record size and the length of the record variable differ.

Programmer Response: Examine the statement giving rise to the message, and check the file and the variable used in the statement, to determine whether information supplied at run-time could have been made known at compilation time.

IEL0990I E 'PASSWORD' ENVIRONMENT OPTION SPECIFIED WITHOUT 'VSAM'.

'PASSWORD' ENVIRONMENT OPTION SPECIFIED WITHOUT 'VSAM'. ENVIRONMENT OPTION IN FILE D. 'VSAM' ASSUMED.

Explanation: A password can be declared only for a VSAM file.

IEL0991I U PROGRAM TOO LARGE. COMPILATION TERMINATED IN PHASE P.

COMPILER RESTRICTION. PROGRAM TOO LARGE. COMPILATION TERMINATED IN PHASE P.

Explanation: The program contains many source variables or procedure invocations which require aggregate temporaries, or many internal procedure or begin blocks. Information about these is held in the compiler directory whose capacity has been exceeded. The problem is more likely to occur when OPT(TIME) compiler option is used since extra demands are placed on dictionary space. Compilation is terminated as the compiler dictionary has been filled up (and no further information can be held in it).

Programmer Response: Divide the program into two or more parts and compile these separately. Increasing the storage will not correct the problem since the number of pages in the dictionary is fixed, and increasing the storage only increases the page size.

IEL0995I U COMPILER IS UNABLE TO ACCESS RUNTIME ROUTINE T.

COMPILER IS UNABLE TO ACCESS RUNTIME LIBRARY ROUTINE T. COMPILATION TERMINATED.

Explanation: The LE/VSE run-time library is required for compilation of PL/I programs. The compiler utilizes the LE/VSE run-time data conversion routines to perform compile-time conversions. In addition, run-time options specified in PLIXOPT declarations are interpreted using the LE/VSE run-time options parsing routines. If these LE/VSE run-time routines are not accessible to the compiler, the compilation is terminated with the message shown above.

Programmer Response:

Provide access to the LE/VSE PRD2.SCEEBASE run-time library or its equivalent.

Messages IEL2233-IEL2274

IEL2233I E SEMICOLON MISSING. T TO NEXT SEMICOLON IGNORED.

SEMICOLON MISSING IN '%NOTE' STATEMENT. TEXT IGNORED FROM T TO NEXT SEMICOLON.

Example:

```
%NOTE (A);
B = 5;
```

IEL2234I E SEVERITY N INVALID. T ASSUMED.

INVALID SEVERITY CODE N IN '%NOTE' STATEMENT. T ASSUMED.

Example:

```
%NOTE ('XYZ',5);
```

Explanation: The severity code in a %NOTE statement must be 0, 4, 8, 12, or 16.

IEL2235I E MESSAGE TEXT TRUNCATED TO N CHARACTERS.

MESSAGE TEXT IN '%NOTE' STATEMENT TOO LONG. TRUNCATED TO FIRST N CHARACTERS.

Explanation: The message text in a %NOTE statement must not exceed 256 characters in length.

IEL2236I E ARGUMENTS TO T IGNORED.

ARGUMENTS SPECIFIED FOR BUILTIN FUNCTION T. ARGUMENTS IGNORED.

Example:

```
%L = COUNTER(A);
```

IEL2237I E 'COUNTER' EXCEEDS '99999'. RESET.

VALUE OF 'COUNTER' EXCEEDS '99999'. VALUE RESET TO '00000'.

Explanation: The COUNTER built-in function cannot be invoked more than 99999 times.

IEL2238I E LEFT PARENTHESIS ASSUMED.

MISSING LEFT PARENTHESIS ASSUMED IN '%NOTE' STATEMENT.

IEL2239I W LISTING CONTROL STATEMENT SPANS LINES.

PREPROCESSOR RESTRICTION. LISTING CONTROL STATEMENT SPANS LINES. STATEMENT NOT IMPLEMENTED.

Explanation: A listing control statement is not implemented by the preprocessor if it spans lines.

IEL2240I E FIRST SETTING OF PARAMETER T ASSUMED.

PARAMETER T MAY NOT BE SET MORE THAN ONCE. FIRST SETTING ASSUMED.

Example:

```
%P: PROC(A,B) STMT RETURNS(CHAR);
 -
 -
%END P;
%ACT P;
P(X,6) A(Z);
```

Explanation: In a statement-form procedure invocation, an attempt has been made to set the same parameter more than once, either by a positional argument and a keyword argument, or by more than one keyword argument.

IEL2241I E SPECIFICATION T IGNORED.

INVALID SPECIFICATION IN '%PROCEDURE' STATEMENT. T IGNORED.

Explanation: Only the attributes RETURNS and STATEMENT can appear on the %PROCEDURE statement.

IEL2242I E INVALID KEYWORD T AND ANY ARGUMENT IGNORED.

INVALID KEYWORD IN STATEMENT-FORM PROCEDURE INVOCATION. T AND ANY ARGUMENT IGNORED.

Example:

```
%P: PROC(A,B,C) RETURNS(FIXED) STMT;
 -
 -
%END P;
%ACT P;
P C(X) A D(Z);
```

Explanation: A keyword has been specified in a statement-form procedure invocation that is not the name of any of the parameters of the procedure.

IEL2243I E COMMA REPLACED BY BLANK.

INVALID COMMA IN STATEMENT-FORM PROCEDURE INVOCATION REPLACED BY BLANK.

Example:

```
%P: PROC(D,E,F) RETURNS(CHAR) STMT;
 -
 -
%END P;
%DCL P ENTRY;
P E(XYZ), F(ABC);
```

IEL2244I E 'PARMSET' INVOKED IN NON-PREPROCESSOR TEXT.

'PARMSET' BUILTIN FUNCTION INVOKED IN NON-PREPROCESSOR TEXT. NULL STRING RETURNED.

Example:

```
%DCL A CHAR, PARMSET BUILTIN;
C = PARMSET(A);
```

IEL2245I E 'PARMSET' INVOKED OUTSIDE A PROCEDURE.

'PARMSET' BUILTIN FUNCTION INVOKED OUTSIDE A PREPROCESSOR PROCEDURE. BIT VALUE ZERO RETURNED.

Example:

```
%DCL C CHAR, F FIXED;
%F = PARMSET(C);
```

IEL2246I E 'PARMSET' HAS NO ARGUMENT.

'PARMSET' BUILTIN FUNCTION HAS NO ARGUMENT. BIT VALUE ZERO RETURNED.

Example:

```
%DCL D FIXED;
%D = PARMSET;
```

IEL2247I E ARGUMENT TO 'PARMSET' IS NOT A PARAMETER.

ARGUMENT TO 'PARMSET' BUILTIN FUNCTION IS NOT A PARAMETER OF THIS PROCEDURE. BIT VALUE ZERO RETURNED.

IEL2248I E RIGHT PARENTHESIS AND SEMICOLON ASSUMED IN D ARGUMENT LIST.

RIGHT PARENTHESIS AND SEMICOLON ASSUMED AT END OF ARGUMENT LIST FOR PROCEDURE D.

Example:

```
%DCL C CHAR;
%C = 'P E(6';
%P: PROC(E,F) STMT RETURNS(CHAR);
-
-
-
%END;
%ACT P;
C;
```

Explanation: This situation can arise where rescanning and replacement are involved, when the final insertion into the preprocessed text is not done until all replacement is completed. Thus, in the example, C is replaced by an invocation of procedure P (which is erroneous and hence the message) which is in turn replaced by the returned value from the procedure. If further replacements are not possible, this is inserted into the text and the semicolon is then processed.

IEL2249I E SEMICOLON ASSUMED IN D ARGUMENT LIST.

SEMICOLON ASSUMED AT END OF ARGUMENT LIST FOR PROCEDURE D.

Example:

```
%DCL C CHAR;
%PROC1: PROC(A1,B1) STMT RETURNS (CHAR);
-
-
-
%END;
%ACT PROC1;
%C = 'PROC1 B1(25)';
C;
```

Explanation: This situation can arise where rescanning and replacement are involved, when the final insertion into the preprocessed text is not done until all replacement is completed. Thus, in the example, C is replaced by an invocation of procedure PROC1 (which is erroneous and hence the message) which is in turn replaced by the returned value from the procedure. If further replacements are not possible this is inserted into the text and the semicolon is then processed.

IEL2250I severity code

Example:

```
%NOTE ('THIS IS A MESSAGE',8);
gives rise to:
IEL2250I E THIS IS A MESSAGE.
```

Explanation: This message number identifies user-supplied messages generated by the preprocessor %NOTE statement. A severity code (I, E, W, S or U) precedes the %NOTE text in the above example. The second parameter on the %NOTE statement determines which code will appear.

IEL2255I E LEFT PARENTHESIS ASSUMED AFTER 'RETURNS'.

LEFT PARENTHESIS ASSUMED AFTER 'RETURNS' IN '%PROCEDURE' STATEMENT.

IEL2256I E CONFLICTING ATTRIBUTE T IGNORED.

ATTRIBUTE T IN '%PROCEDURE' STATEMENT CONFLICTS WITH A PREVIOUSLY SPECIFIED ATTRIBUTE AND IS IGNORED.

Example:

```
%P:PROC RETURNS(FIXED) RETURNS(CHAR);
```

IEL2257I E INVALID SYNTAX. TEXT IGNORED FROM T.

INVALID SYNTAX IN '%PROCEDURE' STATEMENT. TEXT IGNORED FROM T TO NEXT SEMICOLON.

Example:

```
%B: PROC(A) RETURNS = CHAR;
```

IEL2258I S INVALID SYNTAX. TEXT IGNORED FROM T.

INVALID SYNTAX IN STATEMENT-FORM PROCEDURE INVOCATION. TEXT IGNORED FROM T TO NEXT SEMICOLON.

Example:

```
%Q = PROC(J,K,L) STMT RETURNS(CHAR);
-
-
-
%END Q;
%ACT Q;
Q(A) L(12) K 5;
```

IEL2259I W ARGUMENT N TO T MISSING.

ARGUMENT N TO BUILTIN FUNCTION T MISSING. NULL STRING PASSED.

Explanation: An argument in the function reference is missing. The null string will be converted to fixed zero where a fixed argument is required.

IEL2260I W RESTRICTED VALUE FOUND IN DOUBLE-BYTE CHARACTER.

Explanation: A value between and including X'00' through X'06' was found in either a graphic constant or a graphic string within comments in the input stream. The restricted value is replaced with a blank (X'4040') and processing continues.

IEL2261I S THE SOURCE RECORD CONTAINS AN INVALID USE OF A SHIFT-IN OR A SHIFT-OUT. THE RECORD IS IGNORED.

Example:

```
%PROC1: PROC RETURNS(CHAR);
   DCL GG CHAR;
   GG = '<kk<kk>>';
   RETURN (GG);
%END;
```

Explanation: An input data record was received that did not use shift codes properly. The example shows "nested" DBCS characters which is not allowed. This message is also produced if a shift-in was encountered following an SBCS character.

IEL2262I S THE SOURCE RECORD VIOLATES DOUBLE-BYTE CHARACTER CONTINUATION RULES. THE RECORD IS IGNORED.

Explanation: A shift-out was detected in the right margin. This situation is not allowed with PL/I source programs.

IEL2263I S THE SOURCE RECORD ENDS IN DOUBLE-BYTE MODE. THE RECORD IS IGNORED.

Explanation: All PL/I source program records must end with either an SBCS character or a shift-in code.

IEL2264I S A DOUBLE-BYTE ITEM OVERLAPS THE MARGIN. THE RECORD IS IGNORED.

Example:

```
                GG = '<jj  . . .   kk>
          <mm>';
margins:  |                              |
```

Explanation: The right margin terminated a statement between the two bytes of a double-byte character. The same problem can occur when the left margin splits a double-byte character.

In the example above, both the left and right margins split DBCS characters in a constant character string.

IEL2270I S A CHARACTER IN 'X' CONSTANT STARTING WITH T IS INVALID. THE CONSTANT IS IGNORED.

Example:

```
%C = '6FG3'X;
```

Explanation: Characters within X constants must be digits (0-9) or hex characters (A-F). This message will identify the G as being invalid.

IEL2271I S THE STRING CONSTANT STARTING WITH T IS TOO LONG. THE CONSTANT IS TRUNCATED.

Explanation: The preprocessor string constant is too long; its maximum length is 16384 source bytes.

Programmer Response: The string constant can be corrected by breaking it up into several pieces and using the concatenation operator ('||') to reconnect them.

IEL2272I S AN ODD NUMBER OF CHARACTERS IS SPECIFIED FOR 'X' CONSTANT T. THE CONSTANT IS PADDED WITH A ZERO.

Example:

```
%C = '12345'X;
```

Explanation: The preprocessor hexadecimal string must be an even number of characters. If it is not, a zero will be added to the end of the string to make it an even number of characters.

Programmer Response: If you can add a final zero to the string, and still have an acceptable value, then you can avoid this message by writing:

Example:

```
%C = '123450'X
```

IEL2273I E THE STRING CONSTANT ENDS IN DOUBLE-BYTE MODE. IT IS CORRECTED.

IEL2274I S THE STRING CONSTANT CONTAINS AN INVALID USE OF A SHIFT-IN OR A SHIFT-OUT. THE CONSTANT IS IGNORED.

Error and restriction numbers (0 to 946) for IEL0001I, IEL0230I, and IEL0970I

Error and restriction numbers that are identified in messages IEL0001I, IEL0230I, and IEL0970I are listed below. The phase in which the condition occurred, the probable cause, and possible programmer response are given for each number. The base message for IEL0001I is on page 4, the base message for IEL0230I is on page 21, and the base message for IEL0970I is on page 98.

0 ERROR NUMBER 0 DURING PHASE (any).

Explanation: A program check interrupt has occurred.

1 ERROR NUMBER 1 DURING PHASE (any).

Explanation: The phase specified in an XPST macro statement has not been found. The remainder of the job-step has been canceled.

2 ERROR NUMBER 2 DURING PHASE (any).

Explanation: All pages in main storage are UNMOVABLE. An attempt has been made, in response to a request from the stated phase, to find a page which might be spilled in order to make room for either a new or an existing page. However, since all the pages are marked UNMOVABLE, no such spill candidate could be found.

Programmer Response: If possible, rerun the program with a larger SIZE specification. This will increase the size of the page area, and thus the number of pages in main storage.

3 ERROR NUMBER 3 DURING PHASE (any).

Explanation: A call from the stated phase has been made to the control phase which necessitates either (a) writing a page to the spill file, or (b) reading a page into main storage from the spill file. Prior to the I/O operation, the track address of the page concerned has been found to be invalid. In case (a) , the track address held in the header of the page in main storage has been overwritten, and in case (b) the track address of the requested page is invalid.

Programmer Response: Attempt simplification of the statement referred to in the error message.

4 ERROR NUMBER 4 DURING PHASE (any).

Explanation: An attempt has been made by the stated phase to read into main storage an existing page (specified by its track address) from the spill file. This page, however, has not been spilled, the record at the given track address on the spill file being a dummy record at this stage. When this record is read into main storage, its track address field in the page header, not having been initialized, does not match that of the record.

Programmer Response: Attempt simplification of the statement referred to in the error message.

5 ERROR NUMBER 5 DURING PHASE AIICEIUAIUE DUE TO PREVIOUS ERROR NUMBER n IN PHASE p.

Explanation: A compiler error has occurred which makes it impossible for the compiler to continue.

Programmer Response: If possible, rerun the program with a larger SIZE specification. This will increase the size of the page area, and thus the number of pages in main storage.

81 RESTRICTION NUMBER 81 DURING PHASE EA.

Explanation: The compiler has attempted to correct a series of source errors, and this has had a cumulative effect leading to an "unrecoverable" error.

Programmer Response: Correct the source errors diagnosed before the above error and rerun the program.

100 ERROR NUMBER 100 DURING PHASE (any).

Explanation: Invalid dictionary reference passed to decoding routine XRFAB.

101 RESTRICTION NUMBER 101 DURING PHASE (any).

Explanation: Dictionary full.

103 RESTRICTION NUMBER 103 DURING PHASE (any).

Explanation: An attempt has been made to create a dictionary entry larger than a page.

105 ERROR NUMBER 105 DURING PHASE (any).

Explanation: A phase has requested a page which is said to be in the page area. It is not. This message indicates a logic error in the phase concerned.

151 RESTRICTION NUMBER 151 DURING PHASE GA.

Explanation: Invalid or incorrect specifications have been included in the VALUE option of a DEFAULT statement.

Programmer Response: Avoid the use of, or correct, the relevant VALUE option specification(s) in the statement referred to in the message.

152 RESTRICTION NUMBER 152 DURING PHASE GA.

Explanation: Too deep a parenthesis level has been used in an ENVIRONMENT attribute option-list.

Programmer Response: Remove unnecessary parentheses in ENVIRONMENT attribute option-list arguments.

154 ERROR NUMBER 154 DURING PHASE GA.

Explanation: Error during the processing of the attributes in a DECLARE statement.

201 ERROR NUMBER 201 DURING PHASE GM.

Explanation: An error has been made in statement-label handling.

Programmer Response: Check the syntax of the label prefix of the statement referred to in the error message.

220 RESTRICTION NUMBER 220 DURING PHASE (GAIGEIGIIGM).

Explanation: During the scan of an expression, the semicolon has been found in an apparently incorrect position in the statement.

Programmer Response: Check the syntax of the statement. If this is correct, the statement should be simplified.

221 RESTRICTION NUMBER 221 DURING PHASE IA.

Explanation: An invalid statement type has been found in the secondary input text stream.

222 ERROR NUMBER 222 DURING PHASE IA.

Explanation: Underflow of implicit locator chain stack.

223 RESTRICTION NUMBER 223 DURING PHASE IE.

Explanation: Unqualified REFER item found.

Programmer Response: Avoid using the REFER option in this statement.

261 ERROR NUMBER 261 DURING PHASE IE.

Explanation: Structure element descriptor cannot be found.

Programmer Response: Avoid using structures in this statement.

262 ERROR NUMBER 262 DURING PHASE IE.

Explanation: Dimension entry cannot be found in dimension stack.

Programmer Response: Avoid using arrays in this statement.

263 ERROR NUMBER 263 DURING PHASE IE.

Explanation: End of structure stack found where not expected.

Programmer Response: Avoid use of structures in this statement.

264 ERROR NUMBER 264 DURING PHASE IE.

Explanation: End of dimension stack found when processing array structures.

Programmer Response: Avoid using arrays of structures in this statement.

265 ERROR NUMBER 265 DURING PHASE IE.

Explanation: End of text page found where not expected.

Programmer Response: Avoid array assignments in this statement.

266 ERROR NUMBER 266 DURING PHASE IE.

Explanation: Aggregate assignment marker not followed by dictionary reference.

Programmer Response: Avoid using functions with aggregate arguments in this statement.

281 ERROR NUMBER 281 DURING PHASE II.

Explanation: Main stack underflow.

282 ERROR NUMBER 282 DURING PHASE II.

Explanation: Main stack overflow.

Programmer Response: Simplify the statement involved.

300 RESTRICTION NUMBER 300 DURING PHASE (any).

Explanation: This program requires too many temporary variables.

Programmer Response: Simplify the program to reduce the number of temporary variables it requires. If you use structures with the REFER option, put the items with REFER at the end of the structure.

301 RESTRICTION NUMBER 301 DURING PHASE (any).

Explanation: More than 32 qualified temporaries are currently active.

Programmer Response: Simplify any expressions in the statement involved, particularly any that refer to based or subscripted variables.

302 ERROR NUMBER 302 DURING PHASE (any).

Explanation: The phase has encountered a reference to a qualified temporary without having encountered code for its creation. (Qualified temporaries are used for based and subscripted variables.)

Programmer Response: Simplify any expressions in the statement involved.

303 ERROR NUMBER 303 DURING PHASE KA.

Explanation: The phase has found a reference to a string temporary but has not found code for the creation of such a string temporary.

Programmer Response: Simplify any string expressions in the statement involved.

304 ERROR NUMBER 304 DURING PHASE KA.

Explanation: The phase has found a request for the creation of a string temporary in an operation that should not require one.

Programmer Response: Simplify the use of string expressions in the statement involved.

305 ERROR NUMBER 305 DURING PHASE KA.

Explanation: Too many string temporaries (more than 25) are active.

Programmer Response: Simplify any string expressions in the statement involved. String temporaries are also generated for arrays declared with the REFER option. For example, an INITIAL clause in an array of structures generates one string temporary; a reference to an array containing a REFER option whose source (expression parameter) is the target of a previous REFER option generates at least two string temporaries.

306 ERROR NUMBER 306 DURING PHASE KA.

Explanation: Error has been discovered in the compiler labels generated for the program.

Programmer Response: Rearrange the branching in an IF...THEN...GOTO statement.

321 ERROR NUMBER 321 DURING PHASE IK.

Explanation: An incorrect entry has been found in the sort pages.

Programmer Response: Do not specify either or both of the ATTRIBUTE and XREF compiler options for this program.

322 RESTRICTION NUMBER 322 DURING PHASE IK.

Explanation: An incorrect entry has been found in the ENVIRONMENT attribute option-list for a file.

Programmer Response: Do not specify the ATTRIBUTE compiler option for this program.

341 ERROR NUMBER 341 DURING PHASE IM.

Explanation: The "end of program" marker has been found in error. The marker has been encountered during a text scan before the "end of program" text table has been found.

361 RESTRICTION NUMBER 361 DURING PHASE IQ.

Explanation: For computing the size of a target of a concatenate operation, the phase uses a stack whose maximum depth is 30. The maximum has been exceeded.

Programmer Response: Avoid using more than 30 operands in a concatenate operation.

362 ERROR NUMBER 362 DURING PHASE IQ.

Explanation: Erroneous coding in the phase.

Programmer Response: Avoid built-in functions as operands in concatenate expressions.

371 COMPILER ERROR 371 DURING PHASE KE.

Explanation: The table containing array information has overflowed.

Programmer Response: Simplify the data structure referred to by the statement so that it contains fewer arrays of more than one dimension.

402 ERROR NUMBER 402 DURING PHASE KI.

Explanation: A text-table corresponding to the END statement of a user-written DO-loop cannot be found, owing to incorrect input from a previous phase, probably a syntax checking phase.

441 ERROR NUMBER 441 DURING PHASE KI.

Explanation: Text stack is full - logic error in Phase KL.

461 ERROR NUMBER 461 DURING PHASE KM.

Explanation: Text table stack is full - logic error in Phase KM.

481 ERROR NUMBER 481 DURING PHASE KQ.

Explanation: Text input to Phase KQ does not start with an SL text table.

Programmer Response: Simplify the first statement in the compilation.

482 ERROR NUMBER 482 DURING PHASE KQ.

Explanation: An error has been found during the scan of skeleton text tables in Phase KQ, in the compiler-generated subroutine generation routine.

Programmer Response: Simplify the statement referred to in the error message.

483 ERROR NUMBER 483 DURING PHASE KQ.

Explanation: A FORME text table of unknown type has been encountered by the phase. This is probably due to bad output from Phase II or a logic error in the processing of FORME text tables by Phase KQ.

Programmer Response: Simplify the appropriate stream I/O statement.

485 ERROR NUMBER 485 DURING PHASE KQ.

Explanation: A qualified temporary encountered in a stream I/O text table has not been seen previously in the text.

Programmer Response: Simplify the appropriate stream I/O statement.

488 ERROR NUMBER 488 DURING PHASE KQ.

Explanation: Error in input text - a null operand has been found in a DATAE text table.

Programmer Response: Simplify the stream I/O statement referred to in the error message.

489 ERROR NUMBER 489 DURING PHASE KQ.

Explanation: Text input to Phase KQ contains no text tables for a format list.

Programmer Response: If possible, rewrite the GET|PUT EDIT statement with fewer pairs of data and format lists.

492 ERROR NUMBER 492 DURING PHASE KQ.

Explanation: Input text error. The format list input text to Phase KQ in an edit I/O statement starts with a FITE text table.

Programmer Response: Simplify the format list in the edit I/O statement indicated by the error message.

501 ERROR NUMBER 501 DURING PHASE KV.

Explanation: The phase has encountered an UNSPEC of a picture that should have been replaced by a reference to a character string.

Programmer Response: Avoid UNSPEC, particularly of pictures.

522 ERROR NUMBER 522 DURING PHASE OA.

Explanation: The table containing information about temporary operands has been searched for a temporary which could not be found.

524 ERROR NUMBER 524 DURING PHASE OA.

Explanation: The table containing information about qualified temporaries has been searched for a qualified temporary which could not be found.

529 ERROR NUMBER 529 DURING PHASE OA.

Explanation: The stack of active temporary operands maintained by Phase OA was not empty when a fresh statement was due to be processed.

541 ERROR NUMBER 541 DURING PHASE OE.

Explanation: A GOOB text table has been found in which the third operand is not one of the following:

- A label constant
- A label variable
- A qualified temporary

543 ERROR NUMBER 543 DURING PHASE OE.

Explanation: The table containing information about temporary operands has been searched for a temporary which could not be found.

544 ERROR NUMBER 544 DURING PHASE OE.

Explanation: The table containing information about temporary operands is full; further entries can not be made. This fact should have been detected and acted upon by Phase OA. The occurrence, therefore, of the above error message also indicates that Phase OA did not fully handle the situation.

545 ERROR NUMBER 545 DURING PHASE OE.

Explanation: The table containing information about qualified temporaries has been searched for a qualified temporary that could not be found.

548 ERROR NUMBER 548 DURING PHASE OE.

Explanation: The stack of active temporary operands maintained by Phase OE was not empty when a fresh statement was due to be processed.

602 ERROR NUMBER 602 DURING PHASE KK.

Explanation: Test table stack is full; logic error in Phase KK.

641 ERROR NUMBER 641 DURING PHASE OX.

Explanation: A qualified temporary has been referenced which has not been set.

Programmer Response: If possible, rewrite the statement indicated by the error message.

642 ERROR NUMBER 642 DURING PHASE OX.

Explanation: The qualified temporary stack is full. This happens when previous phases of the compiler have not flagged qualified temporaries correctly on their last use.

Programmer Response: Reduce the number of qualified temporaries.

643 ERROR NUMBER 643 DURING PHASE OX.

Explanation: Input text error. A SELECT, WHEN, or OTHERWISE statement has been encountered with an incorrect value in slot ITSELCT. This can be caused by an array expression in a SELECT or WHEN statement.

644 ERROR NUMBER 644 DURING PHASE OX.

Explanation: SELECT stack is full--logic error in OX or an array expression has been specified in a SELECT or WHEN statement.

645 ERROR NUMBER 645 DURING PHASE OX.

Explanation: SELECT stack contains a bad entry--logic error in OX or an array expression has been specified in a SELECT or WHEN statement.

661 ERROR NUMBER 661 DURING PHASE KX.

Explanation: An invalid conversion, generated by one of the phases II through OX, has been encountered.

681 RESTRICTION NUMBER 681 DURING PHASE PC.

Explanation: Phase PC has been asked to construct a symbol table for an invalid identifier. Variables only can occur in data-directed I/O; variables, label constants, or entry-point constants are allowed in CHECK-condition lists. Any invalid or "unusual" identifiers might not have been detected in earlier compiler phases.

Programmer Response: Check the use of data-directed I/O statements or the CHECK condition. Replace any that might cause trouble.

683 ERROR NUMBER 683 DURING PHASE PC.

Explanation: A pictured operand or PICTURE format item requiring a DED or FED cannot be associated with its correct PICTURE specification, as its dictionary reference has been lost.

Programmer Response: Check the use of PICTURE format items and the passing of pictured variables to library subroutines.

721 ERROR NUMBER 721 DURING PHASE PE.

Explanation: An invalid entry has been found during a scan of the variables dictionary.

722 ERROR NUMBER 722 DURING PHASE PE.

Explanation: An invalid entry has been found during a scan of the storage dictionary.

723 ERROR NUMBER 723 DURING PHASE PE.

Explanation: The compiler has failed to assign correct alignment to a STATIC variable which has been initialized.

Programmer Response: Avoid the use of the INITIAL attribute for STATIC variables.

730 ERROR NUMBER 730 DURING PHASE PI.

Explanation: The phase has encountered a text table operand which holds a bit address, but the operand is not an unqualified temporary; input text error or logic error in this phase.

Programmer Response: Modify the usage of the bit variables in the statement, particularly unaligned bit strings and DEFINED bit strings.

731 **ERROR NUMBER 731 DURING PHASE PI.**

Explanation: The phase has encountered a LADDR or MASSN text table which is resetting a temporary with a bit address. This probably means that a bit address is being used before it was created; possible input text error or logic error in this phase.

Programmer Response: Modify the usage of the bit variables in the statement, particularly unaligned bit strings and DEFINED bit strings.

733 **ERROR NUMBER 733 DURING PHASE PI.**

Explanation: The phase has detected that a bit address temporary is associated with more than one base byte address. That is, an unqualified temporary is being used to hold the bit address of different variables; possible input text error or logic error in this phase.

Programmer Response: Modify the usage of the bit variables in the statement, particularly unaligned bit strings and DEFINED bit strings.

741 **ERROR NUMBER 741 DURING PHASE PI.**

Explanation: On input to PI, a qualified temporary has been referred to without being previously defined.

Programmer Response:

1. Try to simplify the statement involved.
2. Avoid indirect references to variables; that is, BASED, subscripted, POSITION(expression) and SUBSTR.

742 **ERROR NUMBER 742 DURING PHASE PI.**

Explanation: Input to PI indicates need for data element descriptor for a data type which does not require one.

Programmer Response: If a conversion is involved, attempt to avoid conversion.

744 **ERROR NUMBER 744 DURING PHASE PI.**

Explanation: The input to PI tries to take address of an operand that does not have an address.

Programmer Response: Simplify the statement involved.

745 **ERROR NUMBER 745 DURING PHASE PI.**

Explanation: No storage base has been provided for a variable in the input to PI.

746 **RESTRICTION NUMBER 746 DURING PHASE PI.**

Explanation: Too many temporaries alive at the same time.

Programmer Response: Try to simplify the statement involved.

762 **ERROR NUMBER 762 DURING PHASE QI.**

Explanation: A text table that should have been deleted by an earlier phase has been found in the input text stream.

763 **ERROR NUMBER 763 DURING PHASE QI.**

Explanation: Invalid input - addressing vector contains incorrect information.

781 **ERROR NUMBER 781 DURING PHASE QA.**

Explanation: Invalid input has been passed to the phase.

Programmer Response: Modify the statement involved.

782 **RESTRICTION NUMBER 782 DURING PHASE QA.**

Explanation: More registers are required than are available.

Programmer Response: Simplify the statement referred to. For example, perform subscript calculation before the statement.

783 **RESTRICTION NUMBER 783 DURING PHASE QA.**

Explanation: Qualified temporary table full, or missing qualified temporary.

Programmer Response: Simplify the statement involved.

784 **ERROR NUMBER 784 DURING PHASE QA.**

Explanation: All of the storage for register temporaries has been used, probably because preceding phases failed to discard register temporaries.

Programmer Response: Simplify the statement. If the statement is a multiple assignment, ensure that there are not more than 32 targets. If the statement is within a big Do loop (containing more than 256 flow units) and if opt(2) is specified, then recompile with opt(0) or replace the Do loop with another scheme, for example an IF-THEN-GOTO scheme.

785 ERROR NUMBER 785 DURING PHASE QA.

Explanation: A base cannot be found. Either the base was never set up, or it was not set up again after use, or Phase QA has discarded the base too soon.

801 ERROR NUMBER 801 DURING PHASE QE.

Explanation: An unrecognizable text table has been found in the input text stream.

901 ERROR NUMBER 901 DURING PHASE SK.

Explanation: Raised by missing, invalid, or duplicate label.

902 ERROR NUMBER 902 DURING PHASE SK.

Explanation: General register 0 has been used as a base register.

903 RESTRICTION NUMBER 903 DURING PHASE SK.

Explanation: An error has been made in the allocation of region numbers.

Programmer Response: Attempt to break up large EDIT or FORMAT statement.

904 ERROR NUMBER 904 DURING PHASE SK.

Explanation: Untranslated text table - a text table has not been converted to object code by any of the code generation phases.

905 RESTRICTION NUMBER 905 DURING PHASE (KVISK).

Explanation: Too many labels (both user-supplied and compiler-generated) in the program, resulting in overflow of the label table.

Programmer Response: Attempt to simplify the program by reducing the number of labels used.

906 ERROR NUMBER 906 DURING PHASE SK.

Explanation: An invalid operation code has been produced by one of the code generation phases.

907 RESTRICTION NUMBER 907 DURING PHASE SK.

Explanation: Too many blocks (BEGIN, PROC, and ON) in the program.

Programmer Response: Rerun with larger SIZE parameter.

921 ERROR NUMBER 921 DURING PHASE SI.

Explanation: Instructions selected from a code skeleton include a local branch without a corresponding local label.

Programmer Response: Rewrite the statement referred to in the error message.

922 ERROR NUMBER 922 DURING PHASE SI.

Explanation: The number of ADCONS requested by phase SK exceeds the number allocated by storage allocation. (The value in XSAADCS exceeds the value in XADCS.)

941 ERROR NUMBER 941 DURING PHASE SM.

Explanation: An invalid entry has been found in the pseudo constants pool.

942 ERROR NUMBER 942 DURING PHASE SM.

Explanation: An inline constant has been found with an invalid type flag.

Programmer Response: Rewrite the statement referred to in the error message.

943 ERROR NUMBER 943 DURING PHASE SM.

Explanation: A marker in the text has an invalid type byte.

Programmer Response: Rewrite the statement referred to in the error message.

944 ERROR NUMBER 944 DURING PHASE SM.

Explanation: An invalid dictionary reference has been found in the input text stream.

Programmer Response: Rewrite the statement referred to in the error message.

945 ERROR NUMBER 945 DURING PHASE SM.

Explanation: An invalid dictionary reference has been found in one of the input text streams.

946 **ERROR NUMBER 946 DURING PHASE SM.**

Explanation: An invalid dictionary reference has been found, derived indirectly from text or dictionary.

Compiler return codes

For every compilation job or job step, the compiler generates a return code that indicates to the operating system the degree of success or failure achieved. This code appears in the "end of step" message that follows the listing of the job control statements and job scheduler messages for each step. Table 3 gives the meanings of the codes.

Table 3. Return codes from compilation of PL/I program

Return code	Meaning
0000	No error detected; compilation completed; successful program run anticipated.
0004	Possible error (warning) detected; compilation completed; successful program run probable.
0008	Error detected; compilation completed; successful program run probable.
0012	Severe error detected; compilation might have been completed; successful program run improbable.
0016	Unrecoverable error detected; compilation terminated abnormally; successful program run impossible.

Batched compilation return codes

The return code generated by a batched compilation is the highest code that would be returned if the procedures were compiled separately.

Interlanguage communication return codes

As part of the interlanguage facilities of PL/I, diagnostic messages are produced, and the return code is set appropriately, if you specify arguments or parameters whose attributes are such that errors might occur at run time. In general, the compiler does not prevent data being passed, nor does it attempt to correct errors. Although it produces messages to indicate likely sources of error, it allows you to attempt to pass any type of data you specify.

Table 4 on page 114 shows the return codes generated by various types of PL/I data.

Table 4. Return codes produced by PL/I data types

PL/I attribute	COBOL argument	COBOL parameter	FORTRAN argument	FORTRAN parameter
ALIGNED	0000	0000	0000	0000
AREA	Note 1	Note 1	Note 1	Note 1
BINARY	0000	0000	0000	0000
BIT	Note 1	Note 1	Note 2	Note 2
CHARACTER	0000	0000	0000	0000
COMPLEX	0004	0004	Note 4	Note 4
CONNECTED	0000	0000	0000	0000
CONTROLLED	0000	0012	0000	0012
DECIMAL	0000	0000	Note 3	Note 3
DEFINED	0000	-	0000	-
Dimension	Note 8	Note 8	0000	0000
ENTRY	0004	0004	0004	0004
EVENT	0004	0004	0004	0004
FILE	0004	0004	0004	0004
FIXED	0000	0000	0000	0000
FLOAT	0000	0000	0000	0000
GRAPHIC	0000	0000	0004	0004
LABEL	0004	0004	0004	0004
OFFSET	0004	0004	0004	0004
PICTURE	0000	0000	0004	0004
POINTER	0004	0004	0004	0004
Precision	Note 6	Note 6	Note 7	Note 7
REAL	0000	0000	0000	0000
Structure	0000	0000	Note 1	Note 1
TASK	0004	0004	0004	0004
UNALIGNED	Note 9	0000	Note 9	0000
Unconnected	Note 5	0000	Note 5	0000
VARYING	0004	0004	0004	0004

Notes:

1 Return code: 0008
Creation of a dummy argument is suppressed.
2 BIT(8) or BIT(32): 0000
Any other length: 0008
In latter case, creation of a dummy argument is suppressed.
3 FLOAT DECIMAL: 0000
FIXED DECIMAL: 0004
4 FLOAT COMPLEX: 0000
FIXED COMPLEX: 0004
5 If creation of a temporary is suppressed by NOMAP option: 0012
If no NOMAP option: 0000
6 Variable is FIXED (p,0) or is short or long FLOAT: 0000
Variable is BINARY FIXED (p,q) with q¬=0 or is extended FLOAT: 0004
7 Variable is float, or is FIXED BINARY with precision (p,0): 0000
Variable is FIXED DECIMAL, or is BINARY (p,q) with q¬=0: 0004
8 If item is element of a structure or is a minor structure: 0000
All other cases: 0008
9 If the argument is an aggregate and creation of a temporary is suppressed by NOMAP, or if argument is scalar: 0012
If argument is an aggregate and no NOMAP: 0000

Bibliography

IBM PL/I for VSE/ESA publications

Fact Sheet, GC26-8052

Installation and Customization Guide, SC26-8057

Licensed Program Specifications, GC26-8055

Language Reference, SC26-8054

Compile-Time Messages and Codes, SC26-8059

Diagnosis Guide, SC26-8058

Migration Guide, SC26-8056

Programming Guide, SC26-8053

Reference Summary, SX26-3836

IBM Language Environment for VSE/ESA publications

Concepts Guide, GC26-8063

Fact Sheet, GC26-8062

Debugging Guide and Run-Time Messages, SC26-8066

Diagnosis Guide, SC26-8060

Installation and Customization Guide, SC26-8064

Licensed Program Specifications, GC26-8061

Programming Guide, SC26-8065

Reference Summary, SX26-3835

VSE/ESA publications

VSE/ESA Version 1

Administration, SC33-6505

Messages and Codes, SC33-6507

System Control Statements, SC33-6513

System Utilities, SC33-6517

System Macros Reference, SC33-6516

Guide to System Functions, SC33-6511

VSE/VSAM Commands and Macros, SC33-6532

VSE/VSAM User's Guide, SC33-6535

VSE/ESA Version 2

Administration, SC33-6605

Messages and Codes, SC33-6607

System Control Statements, SC33-6613

System Utilities, SC33-6617

System Macros Reference, SC33-6616

Guide to System Functions, SC33-6611

VSE/VSAM Commands and Macros, SC33-6631

VSE/VSAM User's Guide, SC33-6632

Related publications

CICS/VSE

Application Programming Guide, SC33-0712

Application Programming Reference, SC33-0713

System Definition and Operations Guide, SC33-0706

Resource Definition, SC33-0708

DFSORT for VSE/ESA

Application Programming Guide, SC26-7040

Sort/Merge II

DOS/VS VM/SP Sort/Merge Version 2 Application Programming Guide, SC33-4044

DL/I DOS/VS

Application Programming: High Level Programming Interface, SH24-5009

Application Programming: CALL and RQDLI Interfaces, SH12-5411

SQL/DS

SQL/Data System Application Programming Guide for VSE, SH09-8098

Softcopy publications

These collections contain the LE/VSE and LE/VSE-conforming language product publications:

VSE Collection, SK2T-0060

Application Development Collection, SK2T-1237

Index

We'd Like to Hear from You

IBM PL/I for VSE/ESA
Compile-Time Messages and Codes
Release 1

Publication No. SC26-8059-00

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
  - Internet: COMMENTS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

IBM PL/I for VSE/ESA
Compile-Time Messages and Codes
Release 1

Publication No. SC26-8059-00

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name	Address
Company or Organization	
Phone No.	

Readers' Comments
SC26-8059-00

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

SC26-8059-00

Readers' Comments

IBM PL/I for VSE/ESA
Compile-Time Messages and Codes
Release 1

Publication No. SC26-8059-00

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Company or Organization

Phone No.

Address

Readers' Comments
SC26-8059-00

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945

Fold and Tape | **Please do not staple** | Fold and Tape

IBM®

Program Number: 5686-069

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

IBM PL/I for VSE/ESA Publications

GC26-8052	Fact Sheet
GC26-8055	Licensed Program Specifications
SC26-8056	Migration Guide
SC26-8057	Installation and Customization Guide
SC26-8053	Programming Guide
SC26-8054	Language Reference
SX26-3836	Reference Summary
SC26-8058	Diagnosis Guide
SC26-8059	Compile-Time Messages and Codes

SC26-8059-00