1 Open Watcom F77 Diagnostic Messages

The Open Watcom FORTRAN 77 compiler checks for errors both at compile time and execution time.

Compile time errors may result from incorrect program syntax, violations of the rules of the language, underflow and overflow as a result of evaluation of expressions, etc. Three types of messages are issued:

EXTENSION *EXT* - This indicates that the programmer has used a feature which is strictly an extension of the FORTRAN 77 language definition. Such extensions may not be accepted by other FORTRAN 77 compilers.

WARNING *WRN* - This indicates that a possible problem has been detected by the compiler. For example, an unlabelled executable statement which follows an unconditional transfer of control can never be executed and so the compiler will issue a message about this condition.

ERROR *ERR* - This indicates that some error was detected which must be corrected by the programmer.

An object file will be created as long as no ERROR type messages are issued.

Execution or run time errors may result from arithmetic underflow or overflow, input/output errors, etc. An execution time error causes the program to cease execution.

Consider the following program, named "DEMO1.FOR", which contains errors.

Example:

```
* This program demonstrates the following features of
* Open Watcom's FORTRAN 77 compiler:
*
*    1. Extensions to the FORTRAN 77 standard are flagged.
*
*    2. Compile time error diagnostics are extensive.  As many
*       errors as possible are diagnosed.
*
*    3. Warning messages are displayed where potential problems
*       can arise.
*
        PROGRAM MAIN
        DIMENSION A(10)
        DO I=1,10
            A(I) = I
            I = I + 1
        ENDLOOP
        GO TO 30
        J = J + 1
30      END
```

If we compile this program with the "extensions" option, the following output appears on the screen.

```
C>wfc demo1 -exten
WATCOM FORTRAN 77/16 Optimizing Compiler Version 2.0 1997/07/16 09:22:47
Copyright (c) 2002-2026 the Open Watcom Contributors. All Rights Reserved.
Portions Copyright (c) 1984-2002 Sybase, Inc. All Rights Reserved.
Source code is available under the Sybase Open Watcom Public License.
See https://github.com/open-watcom/open-watcom-v2 for details.
demo1.for(14): *EXT* DO-05 this DO loop form is not FORTRAN 77 standard
demo1.for(16): *ERR* DO-07 column 13, DO variable cannot be redefined
    while DO loop is active
demo1.for(17): *ERR* SP-19 ENDLOOP statement does not match with DO
    statement
demo1.for(19): *WRN* ST-08 this statement will never be executed due
    to the preceding branch
demo1.for: 9 statements, 0 bytes, 1 extensions, 1 warnings, 2 errors
```

The diagnostic messages consist of the following information:

1. the name of the file being compiled,
2. the line number of the line containing the error (in parentheses),
3. a message type of either extension (*EXT*), error (*ERR*) or warning (*WRN*),
4. a message class and number (e.g., ST-08), and
5. text explaining the nature of the error.

In the above example, the first error occurred on line 16 of the file "DEMO1.FOR". Error number DO-07 was diagnosed. The second error occurred on line 17 of the file "DEMO1.FOR". Error number SP-20 was diagnosed. The other errors are informational messages that do not prevent the successful compilation of the source file.

The following is a list of all messages produced by Open Watcom F77 followed by a brief description. Run-time messages (messages displayed during execution) are also presented. The messages contain references to `%s` and `%d`. They represent strings that are substituted by Open Watcom F77 to make the error message more exact. `%d` represents a string of digits; `%s` any string, usually a symbolic name such as `AMOUNT`, a type such as `INTEGER`, or a symbol class such as `SUBROUTINE`. An error message may contain more than one reference to %d. In such a case, the description will reference them as %dn where n is the occurrence of %d in the error message. The same is true for references to %s.

1.1 Subprogram Arguments

AR-01 *invalid number of arguments to intrinsic function %s1*

The number of actual arguments specified in the argument list for the intrinsic function %s1 does not agree with the dummy argument list. Consult the Language Reference for information on intrinsic functions and their arguments.

AR-02 *dummy argument %s1 appears more than once*

The same dummy argument %s1 is named more than once in the dummy argument list.

AR-03 *dummy argument %s1 must not appear before definition of ENTRY %s2*

The dummy argument %s1 has appeared in an executable statement before its appearance in the definition of %s2 in an ENTRY statement. This is illegal.

1.2 Block Data Subprograms

BD-01 *%s1 was initialized in a block data subprogram but is not in COMMON*

The variable or array element, %s1, was initialized in a BLOCK DATA subprogram but was not specified in a named COMMON block.

BD-02 *%s1 statement is not permitted in a BLOCK DATA subprogram*

The statement, %s1, is not allowed in a BLOCK DATA subprogram. The only statements which are allowed to appear are: IMPLICIT, PARAMETER, DIMENSION, COMMON, SAVE, EQUIVALENCE, DATA, END, and type statements.

1.3 Source Format and Contents

CC-01 *invalid character encountered in source input*

The indicated statement contains an invalid character. Valid characters are: letters, digits, $, *, ., +, –, /, :, =, (,), !, %, ', and ,(comma). Any character may be used inside a character or hollerith string.

CC-02 *invalid character in statement number columns*

A column in columns 1 to 5 of the indicated statement contains a non-digit character. Columns 1 to 5 contain the statement number label. It is made up of digits from 0 to 9 and is greater than 0 and less than or equal to 99999.

CC-03 *character in continuation column, but no statement to continue*

The character in column 6 indicates that this line is a continuation of the previous statement but there is no previous statement to continue.

CC-04 *character encountered is not FORTRAN 77 standard*

A non-standard character was encountered in the source input stream. This is most likely caused by the use of lower case letters.

CC-05 *columns 1-5 in a continuation line must be blank*

When column 6 is marked as a continuation statement to the previous line, columns 1 to 5 must be left blank.

CC-06 *more than 19 continuation lines is not FORTRAN 77 standard*

More than 19 continuation lines is an extension to the FORTRAN 77 language.

CC-07 *end-of-line comment is not FORTRAN 77 standard*

End-of-line comments are an extension to the FORTRAN 77 language. End-of-line comments start with the exclamation mark (!) character.

CC-08 *D in column 1 is not FORTRAN 77 standard*

A "D" in column 1 signifies a debug statement that is compiled when the "__debug__" macro symbol is defined. If the "__debug__" macro symbol is not defined, the statement is ignored. The "c$define" compiler directive or the "define" compiler option can be used to define the "__debug__" macro symbol.

CC-09 *too many continuation lines*

The limit on the number of continuation lines has been reached. This limit depends on the size of each continuation line. A minimum of 61 continuation lines is permitted. If the "xline" option is used, a minimum of 31 continuation lines is permitted.

1.4 COMMON Blocks

CM-01 *%s1 already in COMMON*

The variable or array name, %s1, has already been specified in this or another COMMON block.

CM-02 *initializing %s1 in COMMON outside of block data subprogram is not FORTRAN 77 standard*

The symbol %s1, in a named COMMON block, has been initialized outside of a block data subprogram. This is an extension to the FORTRAN 77 language.

CM-03 *character and non-character data in COMMON is not FORTRAN 77 standard*

The FORTRAN 77 standard specifies that a COMMON block cannot contain both numeric and character data. Allowing COMMON blocks to contain both numeric and character data is an extension to the FORTRAN 77 standard.

CM-04 *COMMON block %s1 has been defined with a different size*

The COMMON block %s1 has been defined with a different size in another subprogram. A named COMMON block must define the same amount of storage units where ever named.

CM-05 *named COMMON block %s1 appears in more than one BLOCK DATA subprogram*

The named COMMON block, %s1, may not appear in more than one BLOCK DATA subprogram.

CM-06 *blank COMMON block has been defined with a different size*

The blank COMMON block has been defined with a different size in another subprogram. This is legal but a warning message is issued.

1.5 Constants

CN-01 *DOUBLE PRECISION COMPLEX constants are not FORTRAN 77 standard*

Double precision complex numbers are an extension to the FORTRAN 77 language. The indicated number is a complex number and at least one of the parts, real or imaginary, is a double precision constant. Both real and imaginary parts will be double precision.

CN-02 *invalid floating-point constant %s1*

The floating-point constant %s1 is invalid. Refer to the chapter entitled "Names, Data Types and Constants" in the Language Reference.

CN-03 *zero length character constants are not allowed*

FORTRAN 77 does not allow character constants of length 0 (i.e., an empty string).

CN-04 *invalid hexadecimal/octal constant*

An invalid hexadecimal or octal constant was specified. Hexadecimal constants can only contain digits or the letters ’a’ through ’f’ and ’A’ through ’F’. Octal constants can only contain the digits ’0’ through ’7’.

CN-05 *hexadecimal/octal constant is not FORTRAN 77 standard*

Hexadecimal and octal constants are extensions to the FORTRAN 77 standard.

1.6 Compiler Options

CO-01 *%s1 is already being included*

An attempt has been made to include a file that is currently being included in the program.

CO-02 *'%s1' option cannot take a NO prefix*

The compiler option %s1, cannot have the NO prefix specified. The NO prefix is used to negate an option. Certain options, including all options that require a value cannot have a NO prefix.

CO-03 *expecting an equals sign following the %s1 option*

The compiler option %s1, requires an equal sign to be between the option keyword and its associated value.

CO-04 *the '%s1' option requires a number*

The compiler option %s1 and an equal sign has been detected but the required associated value is missing.

CO-05 *option '%s1' not recognized - ignored*

The option %s1 is not a recognized compiler option and has been ignored. Consult the User's Guide for a complete list of compiler options.

CO-06 *'%s1' option not allowed in source input stream*

The option %s1 can only be specified on the command line. Consult the User's Guide for a description of which options are allowed in the source input stream.

CO-07 *nesting level exceeded for compiler directives*

Use of the C$IFDEF or C$IFNDEF compiler directives has caused the maximum nesting level to be exceeded. The maximum nesting level is 16.

CO-08 *mismatching compiler directives*

This error message is issued if, for example, a C$ENDIF directive is used and no matching C$IFDEF or C$IFNDEF precedes it. Incorrect nesting of C$IFDEF, C$IFNDEF, C$ELSE and C$ENDIF directives will also cause this message to be issued.

CO-09 *DATA option not allowed*

A source file has been included into the current program through the use of the INCLUDE compiler option. This included source file cannot contain the DATA compiler option.

CO-10 *maximum limit exceeded in the '%s1' option - option ignored*

The user has specified a value on an option which exceeds the maximum allowed value.

CO-11 *DATA option not allowed with OBJECT option*

The DATA compiler option can not appear a file that is compiled with the OBJECT option.

1.7 Compiler Errors

CP-01 *program abnormally terminated*

This message is issued during the execution of the program. If you are running FORTRAN 77, this message indicates that an internal error has occurred in the compiler. Please report this error and any other helpful information about the program being compiled to Watcom so that the problem can be fixed. .pc If you are running an application compiled by the Watcom FORTRAN 77 optimizing compiler, this message may indicate a problem with the compiler or a problem with your program. Try compiling your application with the "debug" option. This causes the generation of run-time checking code to validate, for example, array subscripts and will help ensure that your program is not in error.

CP-02 *argument %d1 incompatible with register*

The register specified in an auxiliary pragma for argument number %d1 is invalid.

CP-03 *subprogram %s1 has invalid return register*

The register specified in an auxiliary pragma for the return value of function %s1 is invalid. This error is issued when, for example, an auxiliary pragma is used to specify EAX as the return register for a double precision function.

CP-04 *low on memory - unable to fully optimize %s1*

There is not enough memory for the code generator to fully optimize subprogram %s1.

CP-05 *internal compiler error %d1*

This error is an internal code generation error. Please report the specified internal compiler error number and any other helpful information about the program being compiled to Watcom so that the problem can be fixed.

CP-06 *illegal register modified by %s1*

An illegal register was said to be modified by %s1 in the auxiliary pragma for %s1. In a 32-bit flat memory model, the base pointer register EBP and segment registers CS, DS, ES, and SS cannot be modified. In small data models, the base pointer register (32-bit EBP or 16-bit BP) and segment registers CS, DS, and SS cannot be modified. In large data models, the base pointer register (32-bit EBP or 16-bit BP) and segment registers CS, and SS cannot be modified.

CP-07 *%s1*

The message specified by %s1 indicates an error during the code generation phase. The most probable cause is an invalid instruction in the in-line assembly code specified in an auxiliary pragma.

CP-08 *fatal: %s1*

The specified error indicates that the code generator has been abnormally terminated. This message will be issued if any internal limit is reached or a keyboard interrupt sequence is pressed during the code generation phase.

CP-09 *dynamic memory not freed*

This message indicates an internal compiler error. Please report this error and any other helpful information about the program being compiled to Watcom so that the problem can be fixed.

CP-10 *freeing unowned dynamic memory*

This message indicates an internal compiler error. Please report this error and any other helpful information about the program being compiled to Watcom so that the problem can be fixed.

CP-11 *The automatic equivalence containing %s1 exceeds 32K limit*

In 16-bit environments, the size of an equivalence on the stack must not exceed 32767 bytes.

CP-12 *The return value of %s1 exceeds 32K limit*

In 16-bit environments, the size of the return value of a function must not exceed 32767 bytes.

CP-13 *The automatic variable %s1 exceeds 32K limit*

In 16-bit environments, the size of any variable on the stack must not exceed 32767 bytes.

1.8 Character Variables

CV-01 *CHARACTER variable %s1 with length (*) not allowed in this expression*

The length of the result of evaluating the expression is indeterminate. One of the operands has an indeterminate length and the result is being assigned to a temporary.

CV-02 *character variable %s1 with length (*) must be a subprogram argument*

The character variable %s1 with a length specification (*) can only be used to declare dummy arguments in the subprogram. The length of a dummy argument assumes the length of the corresponding actual argument.

CV-03 *left and right hand sides overlap in a character assignment statement*

The expression on the right hand side defines a substring of a character variable and tries to assign it to an overlapping part of the same character variable.

1.9 Data Initialization

DA-01 *implied DO variable %s1 must be an integer variable*

The implied DO variable %s1 must be declared as a variable of type INTEGER or must have an implicit INTEGER type.

DA-02 *repeat specification must be a positive integer*

The repeat specification in the constant list of the DATA statement must be an unsigned positive integer.

DA-03 *%s1 appears in an expression but is not an implied DO variable*

The variable %s1 is used to express the array elements in the DATA statement but the variable is not used as an implied DO variable.

DA-04 *%s1 in blank COMMON block cannot be initialized*

A blank or unnamed COMMON block is a COMMON statement with the block name omitted. The entries in blank COMMON blocks cannot be initialized using DATA statements.

DA-05 *data initialization with hexadecimal constant is not FORTRAN 77 standard*

Data initialization with hexadecimal constants is an extension to the FORTRAN 77 language.

DA-06 *cannot initialize %s1 %s2*

Symbol %s2 was used as a %s1. It is illegal for such a symbol to be initialized in a DATA statement. The DATA statement can only be used to initialize variables, arrays, array elements, and substrings.

DA-07 *data initialization in %s1 statement is not FORTRAN 77 standard*

Data initialization in type specification statements is an extension to the FORTRAN 77 language. These include: CHARACTER, COMPLEX, DOUBLE PRECISION, INTEGER, LOGICAL, and REAL.

DA-08 *not enough constants for list of variables*

There are not enough constants specified to initialize all of the names listed in the DATA statement.

DA-09 *too many constants for list of variables*

There are too many constants specified to initialize the names listed in the DATA statement.

DA-10 *cannot initialize %s1 variable %s2 with %s3 constant*

The constant of type %s3 cannot be used to initialize the variable %s2 of type %s1.

DA-11 *entity can only be initialized once during data initialization*

An attempt has been made to initialize an entity more than once in DATA statements.

1.10 Dimensioned Variables

DM-01 *using %s1 incorrectly in dimension expression*

The name used as a dimension declarator has been previously declared as type %s1 and cannot be used as a dimension declarator. A dimension declarator must be an integer expression.

DM-02 *array or array element (possibly substring) associated with %s1 too small*

The dummy argument, array %s1, is defined to be larger than the size of the actual argument.

1.11 DO-loops

DO-01 *statement number %i1 already defined in line %d2 - DO loop is backwards*

The statement number to indicate the end of the DO control structure has been used previously in the program unit and cannot be used to terminate the DO loop. The terminal statement named in the DO statement must follow the DO statement.

DO-02 *%s1 statement not allowed at termination of DO range*

A non-executable statement cannot be used as the terminal statement of a DO loop. These statements include: all declarative statements, ADMIT, AT END, BLOCK DATA, CASE, DO, ELSE, ELSE IF, END, END AT END, END BLOCK, END GUESS, END IF, END LOOP, END SELECT, END WHILE, ENTRY, FORMAT, FUNCTION, assigned GO TO, unconditional GO TO, GUESS, arithmetic and block IF, LOOP, OTHERWISE, PROGRAM, RETURN, SAVE, SELECT, STOP, SUBROUTINE, UNTIL, and WHILE.

DO-03 *improper nesting of DO loop*

A nested DO loop has not been properly terminated before the termination of the outer DO loop.

DO-04 *ENDDO cannot terminate DO loop with statement label*

The ENDDO statement can only terminate a DO loop in which no statement label was specified in the defining DO statement.

DO-05 *this DO loop form is not FORTRAN 77 standard*

As an extension to FORTRAN 77, the following forms of the DO loop are also supported. .autonote .note A DO loop with no statement label specified in the defining DO statement. .note The DO WHILE form of the DO statement. .endnote

DO-06 *expecting comma or DO variable*

The item following the DO keyword and the terminal statement-label (if present) must be either a comma or a DO variable. A DO variable is an integer, real or double precision variable name. The DO statement syntax is as follows: .millust begin DO <tsl> <,> DO-var = ex, ex <, ex> .millust end

DO-07 *DO variable cannot be redefined while DO loop is active*

The DO variable named in the DO statement cannot have its value altered by a statement in the DO loop structure.

DO-08 *incrementation parameter for DO-loop cannot be zero*

The third expression in the DO statement cannot be zero. This expression indicates the increment to the DO variable each iteration of the DO loop. If the increment expression is not specified a value of 1 is assumed.

1.12 Equivalence and/or Common

EC-01 *equivalencing %s1 has caused extension of COMMON block %s2 to the left*

The name %s1 has been equivalenced to a name in the COMMON block %s2. This relationship has caused the storage of the COMMON block to be extended to the left. FORTRAN 77 does not allow a COMMON block to be extended in this way.

EC-02 *%s1 and %s2 in COMMON are equivalenced to each other*

The names %s1 and %s2 appear in different COMMON blocks and each occupies its own piece of storage and therefore cannot be equivalenced.

1.13 END Statement

EN-01 *missing END statement*

The END statement for a PROGRAM, SUBROUTINE, FUNCTION or BLOCK DATA subprogram was not found before the next subprogram or the end of the source input stream.

1.14 Equal Sign

EQ-01 *target of assignment is illegal*

The target of an assignment statement, an input/output status specifier in an input/output statement, or an inquiry specifier in an INQUIRE statement, is illegal. The target in any of the above cases must be a variable name, array element, or a substring name.

EQ-02 *cannot assign value to %s1*

An attempt has been made to assign a value to a symbol with class %s1. For example, an array name cannot be the target of an assignment statement. This error may also be issued when an illegal target is used for the input/output status specifier in an input/output statement or an inquiry specifier in an INQUIRE statement.

EQ-03 *illegal use of equal sign*

An equal sign has been found in the statement but the statement is not an assignment statement.

EQ-04 *multiple assignment is not FORTRAN 77 standard*

More than one equal sign has been found in the assignment statement.

EQ-05 *expecting equals sign*

The equal sign is missing or misplaced. The PARAMETER statement uses an equal sign to equate a symbolic name to the value of a constant expression. The I/O statements use an equal sign to equate the appropriate values to the various specifiers. The DO statement uses an equal sign to assign the initial value to the DO variable.

1.15 Equivalenced Variables

EV-01 *%s1 has been equivalenced to 2 different relative positions*

The storage unit referenced by %s1 has been equivalenced to two different storage units starting in two different places. One name cannot be associated to two different values at the same time.

EV-02 *EQUIVALENCE list must contain at least 2 names*

The list of names to make a storage unit equivalent to several names must contain at least two names.

EV-03 *%s1 incorrectly subscripted in %s2 statement*

The name %s1 has been incorrectly subscripted in a %s2 statement.

EV-04 *incorrect substring of %s1 in %s2 statement*

An attempt has been made to incorrectly substring %s1 in a %s2 statement. For example, if a CHARACTER variable was declared to be of length 4 then (2:5) would be an invalid substring expression.

EV-05 *equivalencing CHARACTER and non-CHARACTER data is not FORTRAN 77 standard*

Equivalencing numeric and character data is an extension to the FORTRAN 77 language.

EV-06 *attempt to substring %s1 in EQUIVALENCE statement but type is %s2*

An attempt has been made to substring the symbolic name %s1 in an EQUIVALENCE statement but the type of the name is %s2 and should be of type CHARACTER.

1.16 Exponentiation

EX-01 *zero**J where J <= 0 is not allowed*

Zero cannot be raised to a power less than or equal to zero.

EX-02 *X**Y where X < 0.0, Y is not of type INTEGER, is not allowed*

When X is less than zero, Y may only be of type INTEGER.

EX-03 *(0,0)**Y where Y is not real is not allowed*

In complex exponentiation, when the base is zero, the exponent may only be a real number or a complex number whose imaginary part is zero.

1.17 ENTRY Statement

EY-01 *type of entry %s1 does not match type of function %s2*

If the type of a function is CHARACTER or a user-defined STRUCTURE, then the type of all entry names must match the type of the function name.

EY-02 *ENTRY statement not allowed within structured control blocks*

FORTRAN 77 does not allow an ENTRY statement to appear between the start and end of a control structure.

EY-03 *size of entry %s1 does not match size of function %s2*

The name %s1 found in an ENTRY statement must be declared to be the same size as that of the function name. If the name of the function or the name of any entry point has a length specification of (*), then all such entries must have a length specification of (*) otherwise they must all have a length specification of the same integer value.

1.18 Format

FM-01 *missing delimiter in format string, comma assumed*

The omission of a comma between the descriptors listed in a format string is an extension to the FORTRAN 77 language. Care should be taken when omitting the comma since the assumed separation may not occur in the intended place.

FM-02 *missing or invalid constant*

An unsigned integer constant was expected with the indicated edit descriptor but was not correctly placed or was missing.

FM-03 *Ew.dDe format code is not FORTRAN 77 standard*

The edit descriptor Ew.dDe is an extension to the FORTRAN 77 language.

FM-04 *missing decimal point*

The indicated edit descriptor must have a decimal point and an integer to indicate the number of decimal positions. These edit descriptors include: F, E, D and G.

FM-05 *missing or invalid edit descriptor in format string*

In the format string, two delimiters were found in succession with no valid descriptor in between.

FM-06 *unrecognizable edit descriptor in format string*

An edit descriptor has been found in the format string that is not a valid code. Valid codes are: apostrophe (’), I, F, E, D, G, L, A, Z, H, T, TL, TR, X, /, :, S, SP, SS, P, BN, B, $, and \.

FM-07 *invalid repeat specification*

The indicated repeatable edit descriptor is invalid. The forms of repeatable edit descriptors are: Iw, Iw.m, Fw.d, Ew.d, Ew.dEe, Dw.d, Gw.d, Gw.dEe, Lw, A, Aw, Ew.dDe, and Zw where w and e are positive unsigned integer constants, and d and m are unsigned integer constants.

FM-08 *$ or \ format code is not FORTRAN 77 standard*

The non-repeatable edit descriptors $ and \ are extensions to the FORTRAN 77 language.

FM-09 *invalid field modifier*

The indicated edit descriptor for a field is incorrect. Consult the Language Reference for the correct form of the edit descriptor.

FM-10 *expecting end of FORMAT statement but found more text*

The right parenthesis was encountered in the FORMAT statement to terminate the statement and more text was found on the line.

FM-11 *repeat specification not allowed for this format code*

A repeat specification was found in front of a format code that is a nonrepeatable edit descriptor. These include: apostrophe, H, T, TL, TR, X, /, :, S, SP, SS, P, BN, BZ, $,and \.

FM-12 *no statement number on FORMAT statement*

The FORMAT statement must have a statement label. This statement number is used by I/O statements to reference the FORMAT statement.

FM-13 *no closing quote on apostrophe edit descriptor*

The closing quote of an apostrophe edit descriptor was not found.

FM-14 *field count greater than 256 is invalid*

The repeat specification of the indicated edit descriptor is greater than the maximum allowed of 256.

FM-15 *invalid field width specification*

The width specifier on the indicated edit descriptor is invalid.

FM-16 *Z format code is not FORTRAN 77 standard*

The Z (hexadecimal format) repeatable edit descriptor is an extension to the FORTRAN 77 language.

FM-17 *FORMAT statement exceeds allotted storage size*

The maximum allowable size of a FORMAT statement has exceeded. The statement must be split into two or more FORMAT statements.

FM-18 *format specification not allowed on input*

A format specification, in the FORMAT statement, is not allowed to be used as an input specification. Valid specifications include: T, TL,TR, X, /, :, P, BN, BZ, I, F, E, D, G, L, A, and Z.

FM-19 *FORMAT missing repeatable edit descriptor*

An attempt has been made to read or write a piece of data without a valid repeatable edit descriptor. All data requires a repeatable edit descriptor in the format. The forms of repeatable edit descriptors are: Iw, Iw.m, Fw.d, Ew.d, Ew.dEe, Dw.d, Gw.d, Gw.dEe, Lw, A, Aw, Ew.dDe, and Zw where w and e are positive unsigned integer constants, and d and m are unsigned integer constants.

FM-20 *missing constant before X edit descriptor, 1 assumed*

The omission of the constant before an X edit descriptor in a format specification is an extension to the FORTRAN 77 language.

FM-21 *Ew.dQe format code is not FORTRAN 77 standard*

The edit descriptor Ew.dQe is an extension to the FORTRAN 77 language.

FM-22 *Qw.d format code is not FORTRAN 77 standard*

The edit descriptor Qw.d is an extension to the FORTRAN 77 language.

1.19 GOTO and ASSIGN Statements

GO-01 *%s1 statement label may not appear in ASSIGN statement but did in line %d2*

The statement label in the ASSIGN statement in line %d2 references a %s1 statement. The statement label in the ASSIGN statement must appear in the same program unit and must be that of an executable statement or a FORMAT statement.

GO-02 *ASSIGN of statement number %i1 in line %d2 not allowed*

The statement label %d1 in the ASSIGN statement is used in the line %d2 which references a non-executable statement. A statement label must appear in the same program unit as the ASSIGN statement and must be that of an executable statement or a FORMAT statement.

GO-03 *expecting TO*

The keyword TO is missing or misplaced in the ASSIGN statement.

1.20 Hollerith Constants

HO-01 *hollerith constant is not FORTRAN 77 standard*

Hollerith constants are an extension to the FORTRAN 77 language.

HO-02 *not enough characters for hollerith constant*

The number of characters following the H or h is not equal to the constant preceding the H or h. A hollerith constant consists of a positive unsigned integer constant n followed by the letter H or h followed by a string of exactly n characters.

1.21 IF Statements

IF-01 *ELSE block must be the last block in block IF*

Another ELSE IF block has been found after the ELSE block. The ELSE block must be the last block in an IF block. The form of the block IF is as follows: .millust begin IF (logical expression) THEN [:block-label] {statement} { ELSE IF {statement} } [ELSE {statement}] ENDIF .millust end

IF-02 *expecting THEN*

The keyword THEN is missing or misplaced in the block IF statement. The form of the block IF is as follows: .millust begin IF (logical expression) THEN [:block-label] {statement} { ELSE IF {statement} } [ELSE {statement}] ENDIF .millust end

1.22 I/O Lists

IL-01 *missing or invalid format/FMT specification*

A valid format specification is required on all READ and WRITE statements. The format specification is specified by: .millust begin [FMT=] <format identifier> .millust end .pc .sy <format identifier> is one of the following: statement label, integer variable-name, character array-name, character expression, or *.

IL-02 *the UNIT may not be an internal file for this statement*

An internal file may only be referenced in a READ or WRITE statement. An internal file may not be referenced in a BACKSPACE, CLOSE, ENDFILE, INQUIRE, OPEN, or REWIND statement.

IL-03 *%s1 statement cannot have %s2 specification*

The I/O statement %s1 may not have the control information %s2 specified.

IL-04 *variable must have a size of 4*

The variable used as a specifier in an I/O statement must be of size 4 but another size was specified. These include the EXIST, OPENED, RECL, IOSTAT, NEXTREC, and NUMBER. The name used in the ASSIGN statement must also be of size 4 but a different size was specified.

IL-05 *missing or unrecognizable control list item %s1*

A control list item %s1 was encountered in an I/O statement and is not a valid control list item for that statement, or a control list item was expected and was not found.

IL-06 *attempt to specify control list item %s1 more than once*

The control list item %s1 in the indicated I/O statement, has been named more than once.

IL-07 *implied DO loop has no input/output list*

The implied DO loop specified in the I/O statement does not correspond with a variable or expression in the input/output list.

IL-08 *list-directed input/output with internal files is not FORTRAN 77 standard*

List-directed input/output with internal files is an extension to the FORTRAN 77 language.

IL-09 *FORTRAN 77 standard requires an asterisk for list-directed formatting*

An optional asterisk for list-directed formatting is an extension to the FORTRAN 77 language. The standard FORTRAN 77 language specifies that an asterisk is required.

IL-10 *missing or improper unit identification*

The control specifier, UNIT, in the I/O statement is either missing or identifies an improper unit. The unit specifier specifies an external unit or internal file. The external unit identifier is a non-negative integer expression or an asterisk. The internal file identifier is character variable, character array, character array element, or character substring.

IL-11 *missing unit identification or file specification*

An identifier to specifically identify the required file is missing. The UNIT specifier is used to identify the external unit or internal file. The FILE specifier in the INQUIRE and OPEN statements is used to identify the file name.

IL-12 *asterisk unit identifier not allowed in %s1 statement*

The BACKSPACE, CLOSE, ENDFILE, INQUIRE, OPEN, and REWIND statements require the external unit identifier be an unsigned positive integer from 0 to 999.

IL-13 *cannot have both UNIT and FILE specifier*

There are two valid forms of the INQUIRE statement; INQUIRE by FILE and INQUIRE by UNIT. Both of these specifiers cannot be specified in the same statement.

IL-14 *internal files require sequential access*

An attempt has been made to randomly access an internal file. Internal files may only be accessed sequentially.

IL-15 *END specifier with REC specifier is not FORTRAN 77 standard*

The FORTRAN 77 standard specifies that an end-of-file condition can only occur with a file connected for sequential access or an internal file. The REC specifier indicates that the file is connected for direct access. This extension allows the programmer to detect an end-of-file condition when reading the records sequentially from a file connected for direct access.

IL-16 *%s1 specifier in i/o list is not FORTRAN 77 standard*

The specified i/o list item is provided as an extension to the FORTRAN 77 language.

IL-17 *i/o list is not allowed with NAMELIST-directed format*

An i/o list is not allowed when the format specification is a NAMELIST.

IL-18 *non-character array as format specifier is not FORTRAN 77 standard*

A format specifier must be of type character unless it is an array name. Allowing a non-character array name is an extension to the FORTRAN 77 standard.

1.23 IMPLICIT Statements

IM-01 *illegal range of characters*

In the IMPLICIT statement, the first letter in the range of characters must be smaller in the collating sequence than the second letter in the range.

IM-02 *letter can only be implicitly declared once*

The indicated letter has been named more than once in this or a previous IMPLICIT statement. A letter may only be named once.

IM-03 *unrecognizable type*

The type declared in the IMPLICIT statement is not one of INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL or CHARACTER.

IM-04 *(*) length specifier in an IMPLICIT statement is not FORTRAN 77 standard*

A character length specified of (*) in an IMPLICIT statement is an extension to the FORTRAN 77 language.

IM-05 *IMPLICIT NONE allowed once or not allowed with other IMPLICIT statements*

The IMPLICIT NONE statement must be the only IMPLICIT statement in the program unit in which it appears. Only one IMPLICIT NONE statement is allowed in a program unit.

1.24 Input/Output

IO-01 *BACKSPACE statement requires sequential access mode*

The file connected to the unit specified in the BACKSPACE statement has not been opened for sequential access.

IO-02 *input/output is already active*

An attempt has been made to read or write a record when there is an already active read or write in progress. The execution of a READ or WRITE statement has caused transfer to a function that contains a READ or WRITE statement.

IO-03 *ENDFILE statement requires sequential access mode*

The specified external unit identifier must be connected for sequential access but was connected for direct access.

IO-04 *formatted connection requires formatted input/output statements*

The FORM specifier in the OPEN statement specifies FORMATTED and the subsequent READ and/or WRITE statement does not use formatted I/O. If the FORM specifier has been omitted and access is SEQUENTIAL then FORMATTED is assumed. If the access is DIRECT then UNFORMATTED is assumed.

IO-05 *unformatted connection requires unformatted input/output statements*

The FORM specifier in the OPEN statement specifies UNFORMATTED and the subsequent READ and/or WRITE statement uses formatted I/O. If the FORM specifier has been omitted and access is SEQUENTIAL then FORMATTED is assumed. If the access is DIRECT then UNFORMATTED is assumed.

IO-06 *REWIND statement requires sequential access*

The external unit identifier is not connected to a sequential file. The REWIND statement positions to the first record in the file.

IO-07 *bad character in input field*

The data received from the record in a file does not match the type of the input list item.

IO-08 *BLANK specifier requires FORM specifier to be 'FORMATTED'*

In the OPEN statement, the BLANK specifier may only be used when the FORM specifier has the value of FORMATTED. The BLANK specifier indicates whether blanks are treated as zeroes or ignored.

IO-09 *system file error - %s1*

A system error has occurred while attempting to access a file. The I/O system error message is displayed.

IO-10 *format specification does not match data type*

A format specification in the FMT specifier or FORMAT statement specifies data of one type and the variable list specifies data of a different type.

IO-11 *input item does not match the data type of list variable*

In the READ statement, the data type of a variable listed is not of the same data type in the data file. For example, non-digit character data being read into an integer item.

IO-12 *internal file is full*

The internal file is full of data. If a file is a variable then the file may only contain one record. If the file is a character array then there can be one record for each array element.

IO-13 *RECL specifier is invalid*

In the OPEN statement, the record length specifier must be a positive integer expression.

IO-14 *invalid STATUS specifier in CLOSE statement*

The STATUS specifier can only have a value of KEEP or DELETE. If the STATUS in the OPEN statement is SCRATCH then the KEEP status on the CLOSE statement cannot be used.

IO-15 *unit specified is not connected*

The unit number specified in the I/O statement has not been previously connected.

IO-16 *attempt to perform data transfer past end of file*

An attempt has been made to read or write data after the end of file has been read or written.

IO-17 *invalid RECL specifier/ACCESS specifier combination*

In the OPEN statement, if the ACCESS specifier is DIRECT then the RECL specifier must be given.

IO-18 *REC specifier required in direct access input/output statements*

In the OPEN statement, the ACCESS specified was DIRECT. All subsequent input/output statements for that file must use the REC specifier to indicate which record to access.

IO-19 *REC specifier not allowed in sequential access input/output statements*

In the OPEN statement, the ACCESS specified was SEQUENTIAL. The REC specifier may not be used in subsequent I/O statements for that file. The REC specifier is used to indicate which record to access when access is DIRECT.

IO-20 *%s1 specifier may not change in a subsequent OPEN statement*

The %s1 specifier may not be changed on a subsequent OPEN statement for the same file, in the same program. Only the BLANK specifier may be changed.

IO-21 *invalid STATUS specifier for given file*

In the OPEN statement, the STATUS specifier does not match with the actual file status: OLD means the file must exist, NEW means the file must not exist. If the STATUS specifier is omitted, UNKNOWN is assumed.

IO-22 *invalid STATUS specifier/FILE specifier combination*

In the OPEN statement, if the STATUS is SCRATCH, the FILE specifier cannot be used. If the STATUS is NEW or OLD, the FILE specifier must be given.

IO-23 *record size exceeded during unformatted input/output*

This error is issued when the size of an i/o list item exceeds the maximum record size of the file. The record size can be specified using the RECL= specified in the OPEN statement.

IO-24 *unit specified does not exist*

The external unit identifier specified in the input/output statement has not yet been connected. Use preconnection or the OPEN statement to connect a file to the external unit identifier.

IO-25 *REC specifier is invalid*

The REC specifier must be an unsigned positive integer.

IO-26 *UNIT specifier is invalid*

The UNIT specifier must be an unsigned integer between 0 and 999 inclusive.

IO-27 *formatted record or format edit descriptor is too large for record size*

This error is issued when the amount of formatted data in a READ, WRITE or PRINT statement exceeds the maximum record size of the file. The record size can be specified using the RECL= specified in the OPEN statement.

IO-28 *illegal '%s1=' specifier*

In the OPEN or CLOSE statement the value associated with the %s1 specifier is not a valid value. In the OPEN statement, STATUS may only be one of OLD, NEW, SCRATCH or UNKNOWN; ACCESS may only be one of SEQUENTIAL, APPEND or DIRECT; FORM may only be one of FORMATTED or UNFORMATTED; CARRIAGECONTROL may only be one of YES or NO; RECORDTYPE may only be one of FIXED, TEXT or VARIABLE; ACTION may only be one of READ, WRITE or READ/WRITE; and BLANK may only be one of NULL, or ZERO. In the CLOSE statement the STATUS may only be one of KEEP or DELETE.

IO-29 *invalid CARRIAGECONTROL specifier/FORM specifier combination*

The CARRIAGECONTROL specifier is only allowed with formatted i/o statements.

IO-30 *i/o operation not consistent with file attributes*

An attempt was made to read from a file that was opened with ACTION=WRITE or write to a file that was opened with ACTION=READ. This message is also issued if you attempt to write to a read-only file or read from a write-only file.

IO-31 *symbol %s1 not found in NAMELIST*

During NAMELIST-directed input, a symbol was specified that does not belong to the NAMELIST group specified in the i/o statement.

IO-32 *syntax error during NAMELIST-directed input*

Bad input was encountered during NAMELIST-directed input. Data must be in a special form during NAMELIST-directed input.

IO-33 *subscripting error during NAMELIST-directed i/o*

An array was incorrectly subscripted during NAMELIST-directed input.

IO-34 *substring error during NAMELIST-directed i/o*

An character array element or variable was incorrectly substring during NAMELIST-directed input.

IO-35 *BLOCKSIZE specifier is invalid*

In the OPEN statement, the block size specifier must be a positive integer expression.

IO-36 *invalid operation for files with no record structure*

An attempt has been made to perform an i/o operation on a file that requires a record structure. For example, it is illegal to use a BACKSPACE statement for a file that has no record structure.

IO-37 *integer overflow converting character data to integer*

An overflow has occurred while converting the character data read to its internal representation as an integer.

IO-38 *range exceeded converting character data to floating-point*

An overflow or underflow has occurred while converting the character data read to its internal representation as a floating-point number.

1.25 Program Termination

KO-01 *floating-point divide by zero*

An attempt has been made to divide a number by zero in a floating-point expression.

KO-02 *floating-point overflow*

The floating-point expression result has exceeded the maximum floating-point number.

KO-03 *floating-point underflow*

The floating-point expression result has exceeded the minimum floating-point number.

KO-04 *integer divide by zero*

An attempt has been made to divide a number by zero in an integer expression.

KO-05 *program interrupted from keyboard*

The user has interrupted the compilation or execution of a program through use of the keyboard.

KO-06 *integer overflow*

The integer expression result has exceeded the maximum integer number.

KO-07 *maximum pages of output exceeded*

The specified maximum number of output pages has been exceeded. The maximum number of output pages can be increased by using the "pages=n" option in the command line or specifying C$PAGES=n in the source file.

KO-08 *statement count has been exceeded*

The maximum number of source statements has been executed. The maximum number of source statements that can be executed can be increased by using the "statements=n" option in the command line or specifying C$STATEMENTS=n in the source file.

KO-09 *time limit exceeded*

The maximum amount of time for program execution has been exceeded. The maximum amount of time can be increased by using the "time=t" option in the command line or specifying C$TIME=t in the source file.

1.26 Library Routines

LI-01 *argument must be greater than zero*

The argument to the intrinsic function must be greater than zero (i.e., a positive number).

LI-02 *absolute value of argument to arcsine, arccosine must not exceed one*

The absolute value of the argument to the intrinsic function ASIN or ACOS cannot be greater than or equal to the value 1.0.

LI-03 *argument must not be negative*

The argument to the intrinsic function must be greater than or equal to zero.

LI-04 *argument(s) must not be zero*

The argument(s) to the intrinsic function must not be zero.

LI-05 *argument of CHAR must be in the range zero to 255*

The argument to the intrinsic function CHAR must be in the range 0 to 255 inclusive. CHAR returns the character represented by an 8-bit pattern.

LI-06 *%s1 intrinsic function cannot be passed 2 complex arguments*

The second argument to the intrinsic function CMPLX and DCMPLX cannot be a complex number.

LI-07 *argument types must be the same for the %s1 intrinsic function*

The second argument to the intrinsic function CMPLX or DCMPLX must be of the same type as the first argument. The second argument may only be used when the first argument is of type INTEGER, REAL or DOUBLE PRECISION.

LI-08 *expecting numeric argument, but %s1 argument was found*

The argument to the intrinsic function, INT, REAL, DBLE, CMPLX, or DCMPLX was of type %s1 and a numeric argument was expected.

LI-09 *length of ICHAR argument greater than one*

The length of the argument to the intrinsic function ICHAR must be of type CHARACTER and length of 1. ICHAR converts a character to its integer representation.

LI-10 *cannot pass %s1 as argument to intrinsic function*

The item %s1 cannot be used as an argument to an intrinsic function. Only constants, simple variables, array elements, and substring array elements may be used as arguments.

LI-11 *intrinsic function requires argument(s)*

An attempt has been made to invoke an intrinsic function and no actual arguments were listed.

LI-12 *%s1 argument type is invalid for this generic function*

The type of the argument used in the generic intrinsic function is not correct.

LI-13 *this intrinsic function cannot be passed as an argument*

Only the specific name of the intrinsic function can be used as an actual argument. The generic name may not be used. When the generic and intrinsic names are the same, use the INTRINSIC statement.

LI-14 *expecting %s1 argument, but %s2 argument was found*

An argument of type %s2 was passed to a function but an argument of type %s1 was expected.

LI-15 *intrinsic function was assigned wrong type*

The declared type of an intrinsic function does not agree with the actual type.

LI-16 *intrinsic function %s1 is not FORTRAN 77 standard*

The specified intrinsic function is provided as an extension to the FORTRAN 77 language.

LI-17 *argument to ALLOCATED intrinsic function must be an allocatable array or character*(*) variable*

The argument to the intrinsic function ALLOCATED must be an allocatable array or character*(*) variable.

LI-18 *invalid argument to ISIZEOF intrinsic function*

The argument to the intrinsic function ISIZEOF must be a user-defined structure name, a symbol name, or a constant.

1.27 Mixed Mode

MD-01 *relational operator has a logical operand*

The operands of a relational expression must either be both arithmetic or both character expressions. The operand indicated is a logical expression.

MD-02 *mixing DOUBLE PRECISION and COMPLEX types is not FORTRAN 77 standard*

The mixing of items of type DOUBLE PRECISION and COMPLEX in an expression is an extension to the FORTRAN 77 language.

MD-03 *operator not expecting %s1 operands*

Operands of type %s1 cannot be used with the indicated operator. The operators **, /, *, +, and – may only have numeric type data. The operator // may only have character type data.

MD-04 *operator not expecting %s1 and %s2 operands*

Operands of conflicting type have been encountered. For example, in a relational expression, it is not possible to compare a character expression to an arithmetic expression. Also, the type of the left hand operand of the field selection operator must be a user-defined structure.

MD-05 *complex quantities can only be compared using .EQ. or .NE.*

Complex operands cannot be compared using less than (.LT.), less than or equal (.LE.), greater than (.GT.), or greater than or equal (.GE.) operators.

MD-06 *unary operator not expecting %s1 type*

The unary operators, + and –, may only be used with numeric types. The unary operator .NOT. may be used only with a logical or integer operand. The indicated operand was of type %s1 which is not one of the valid types.

MD-07 *logical operator with integer operands is not FORTRAN 77 standard*

Integer operands are permitted with the logical operators .AND., .OR., .EQV., .NEQV., .NOT. and .XOR. as an extension to the FORTRAN 77 language.

MD-08 *logical operator %s1 is not FORTRAN 77 standard*

The specified logical operator is an extension to the FORTRAN 77 standard.

1.28 Memory Overflow

MO-01 *%s1 exceeds compiler limit of %u2 bytes*

An internal compiler limit has been reached. %s1 describes the limit and %d2 specifies the limit.

MO-02 *out of memory*

All available memory has been used up. During the compilation phase, memory is primarily used for the symbol table. During execution, memory is used for file descriptors and buffers, and dynamically allocatable arrays and character*(*) variables.

MO-03 *dynamic memory exhausted due to length of this statement - statement ignored*

There was not enough memory to encode the specified statement. This message is usually issued when the compiler is low on memory or if the statement is a very large statement that spans many continuation lines. This error does not terminate the compiler since it may have been caused by a very large statement. The compiler attempts to compile the remaining statements.

MO-04 *attempt to deallocate an unallocated array or character*(*) variable*

An attempt has been made to deallocate an array that has not been previously allocated. An array or character*(*) variable must be allocated using an ALLOCATE statement.

MO-05 *attempt to allocate an already allocated array or character*(*) variable*

An attempt has been made to allocate an array or character*(*) variable that has been previously allocated in an ALLOCATE statement.

MO-06 *object memory exhausted*

The amount of object code generated for the program has exceeded the amount of memory allocated to store the object code. The "/codesize" option can be used to increase the amount of memory allocated for object code.

1.29 Parentheses

PC-01 *missing or misplaced closing parenthesis*

An opening parenthesis '(' was found but no matching closing parenthesis ')' was found before the end of the statement.

PC-02 *missing or misplaced opening parenthesis*

A closing parenthesis ')' was found before the matching opening parenthesis '('.

PC-03 *unexpected parenthesis*

A parenthesis was found in a statement where parentheses are not expected.

PC-04 *unmatched parentheses*

The parentheses in the expression are not balanced.

1.30 PRAGMA Compiler Directive

PR-01 *expecting symbolic name*

Every auxiliary pragma must refer to a symbol. This error is issued when the symbolic name is illegal or missing. Valid symbolic names are formed from the following characters: a dollar sign, an underscore, digits and any letter of the alphabet. The first character of a symbolic name must be alphabetic, a dollar sign, or an underscore.

PR-02 *illegal size specified for VALUE attribute*

The VALUE argument attribute of an auxiliary pragma contains in illegal length specification. Valid length specifications are 1, 2, 4 and 8.

PR-03 *illegal argument attribute*

An illegal argument attribute was specified. Valid argument attributes are VALUE, REFERENCE, or DATA_REFERENCE.

PR-04 *continuation line must contain a comment character in column 1*

When continuing a line of an auxiliary pragma directive, the continued line must end with a back-slash ('\') character and the continuation line must begin with a comment character ('c', 'C' or '*') in column 1.

PR-05 *expecting '%s1' near '%s2'*

A syntax error was found while processing a PRAGMA directive. %s1 identifies the expected information and %s2 identifies where in the pragma the error occurred.

PR-06 *in-line byte sequence limit exceeded*

The limit on the number of bytes of code that can be generated in-line using a an auxiliary pragma has been exceeded. The limit is 127 bytes.

PR-07 *illegal hexadecimal data in byte sequence*

An illegal hexadecimal constant was encountered while processing a in-line byte sequence of an auxiliary pragma. Valid hexadecimal constants in an in-line byte sequence must begin with the letter Z or z and followed by a string of hexadecimal digits.

PR-08 *symbol '%s1' in in-line assembly code cannot be resolved*

The symbol %s1, referenced in an assembly language instruction in an auxiliary pragma, could not be resolved.

1.31 RETURN Statement

RE-01 *alternate return specifier only allowed in subroutine*

An alternate return specifier, in the RETURN statement, may only be specified when returning from a subroutine.

RE-02 *RETURN statement in main program is not FORTRAN 77 standard*

A RETURN statement in the main program is allowed as an extension to the FORTRAN 77 standard.

1.32 SAVE Statement

SA-01 *COMMON block %s1 saved but not properly defined*

The named COMMON block %s1 was listed in a SAVE statement but there is no named COMMON block defined by that name.

SA-02 *COMMON block %s1 must be saved in every subprogram in which it appears*

The named COMMON block %s1 appears in a SAVE statement in another subprogram and is not in a SAVE statement in this subprogram. If a named COMMON block is specified in a SAVE statement in a subprogram, it must be specified in a SAVE statement in every subprogram in which that COMMON block appears.

SA-03 *name already appeared in a previous SAVE statement*

The indicated name has already been referenced in another SAVE statement in this subprogram.

1.33 Statement Functions

SF-01 *statement function definition contains duplicate dummy arguments*

A dummy argument is repeated in the argument list of the statement function.

SF-02 *character length of statement function name must not be (*)*

If the type of a character function is character, its length specification must not be (*); it must be a constant integer expression.

SF-03 *statement function definition contains illegal dummy argument*

A dummy argument of type CHARACTER must have a length specification of an integer constant expression that is not (*).

SF-04 *cannot pass %s1 %s2 to statement function*

The actual arguments to a statement function can be any expression except character expressions involving the concatenation of an operand whose length specification is (*) unless the operand is a symbolic constant.

SF-05 *%s1 actual argument was passed to %s2 dummy argument*

The indicated actual argument is of type %s1 which is not the same type as that of the dummy argument of type %s2.

SF-06 *incorrect number of arguments passed to statement function %s1*

The number of arguments passed to statement function %s1 does not agree with the number of dummy arguments specified in its definition.

SF-07 *type of statement function name must not be a user-defined structure*

The type of a statement function cannot be a user-defined structure. Valid types for statement functions are: LOGICAL*1, LOGICAL, INTEGER*1, INTEGER*2, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, DOUBLE COMPLEX, and CHARACTER. If the statement function is of type CHARACTER, its length specification must not be (*); it must be an integer constant expression.

1.34 Source Management

SM-01 *system file error reading %s1 - %s2*

An I/O error, described by %s2, has occurred while reading the FORTRAN source file %s1.

SM-02 *error opening file %s1 - %s2*

The FORTRAN source file %s1 could not be opened. The error is described by %s2.

SM-03 *system file error writing %s1 - %s2*

An I/O error, described by %s2, has occurred while writing to the file %s1.

SM-04 *error spawning %s1 - %s2*

An error, described by %s2, occurred while trying to spawn the external program named %s1.

SM-05 *error while linking*

An error occurred while trying to create the executable file. See the WLINK documentation for a description of the error.

SM-06 *error opening %s1 - too many temporary files exist*

The compiler was not able to open a temporary file for intermediate storage during code generation. Temporary files are created in the directory specified by the TMP environment variable. If the TMP environment variable is not set, the temporary file is created in the current directory. This error is issued if an non-existent directory is specified in the TMP environment variable, or more than 26 concurrent compiles are taking place in a multi-tasking environment and the directory in which the temporary files are created is the same for all compilation processes.

SM-07 *generation of browsing information failed*

An error occurred during the generation of browsing information. For example, a disk full condition encountered during the creation of the browser module file will cause this message to be issued. Browsing information is generated when the /db switch is specified.

1.35 Structured Programming Features

SP-01 *cannot have both ATEND and the END= specifier*

It is not valid to use the AT END control statement and the END= option on the READ statement. Only one method can be used to control the end-of-file condition.

SP-02 *ATEND must immediately follow a READ statement*

The indicated AT END control statement or block does not immediately follow a READ statement. The AT END control statement or block is executed when an end-of-file condition is encountered during the read.

SP-03 *block label must be a symbolic name*

The indicated block label must be a symbolic name. A symbolic name must start with a letter and contain no more than 32 letters and digits. A letter is an upper or lower case letter of the alphabet, a dollar sign ($), or an underscore (_). A digit is a character in the range ’0’ to ’9’.

SP-04 *could not find a structure to %s1 from*

This message is issued in the following cases. .autonote .note There is no control structure to QUIT from. The QUIT statement will transfer control to the statement following the currently active control structure or return from a REMOTE BLOCK if no other control structures are active within the REMOTE BLOCK. .note There is no control structure to EXIT from. The EXIT statement is used to exit a loop-processing structure such as DO, DO WHILE, WHILE and LOOP, to return from a REMOTE BLOCK regardless of the number of active control structures within the REMOTE BLOCK, or to transfer control from a GUESS or ADMIT block to the statement following the ENDGUESS statement. .note There is no active looping control structure from which a CYCLE statement can be used. A CYCLE statement can only be used within a DO, DO WHILE, WHILE and LOOP control structure. .endnote

SP-05 *REMOTE BLOCK is not allowed in the range of any control structure*

An attempt has been made to define a REMOTE BLOCK inside a control structure. Control structures include IF, LOOP, WHILE, DO, SELECT and GUESS. When a REMOTE BLOCK definition is encountered during execution, control is transferred to the statement following the corresponding END BLOCK statement.

SP-06 *the SELECT statement must be followed immediately by a CASE statement*

The statement immediately after the SELECT statement must be a CASE statement. The SELECT statement allows one of a number of blocks of code (case blocks) to be selected for execution by means of an integer expression in the SELECT statement.

SP-07 *cases are overlapping*

The case lists specified in the CASE statements in the SELECT control structure are in conflict. Each case list must specify a unique integer constant expression or range.

SP-08 *select structure requires at least one CASE statement*

In the SELECT control structure, there must be at least one CASE statement.

SP-09 *cannot branch to %i1 from outside control structure in line %d2*

The statement in line %d2 passes control to the statement %d1 in a control structure. Control may only be passed out of a control structure or to another place in that control structure. Control structures include DO, GUESS, IF, LOOP, SELECT, and WHILE.

SP-10 *cannot branch to %i1 inside structure on line %d2*

The statement attempts to pass control to statement %d1 in line %d2 which is in a control structure. Control may only be passed out of a control structure or to another place in that control structure. Control structures include DO, GUESS, IF, LOOP, SELECT, and WHILE.

SP-11 *low end of range exceeds the high end*

The first number, the low end of the range, is greater than the second number, the high end of the range.

SP-12 *default case block must follow all case blocks*

The default case block in the SELECT control structure must be the last case block. A case block may not follow the default case block.

SP-13 *attempt to branch out of a REMOTE BLOCK*

An attempt has been made to transfer execution control out of a REMOTE BLOCK. A REMOTE BLOCK may only be terminated with the END BLOCK statement. Execution of a REMOTE BLOCK is similar in concept to execution of a subroutine.

SP-14 *attempt to EXECUTE undefined REMOTE BLOCK %s1*

The REMOTE BLOCK %s1 referenced in the EXECUTE statement does not exist in the current program unit. A REMOTE BLOCK is local to the program unit in which it is defined and may not be referenced from another program unit.

SP-15 *attempted to use REMOTE BLOCK recursively*

An attempt was made to execute a REMOTE BLOCK which was already active.

SP-16 *cannot RETURN from subprogram within a REMOTE BLOCK*

An illegal attempt has been made to execute a RETURN statement within a REMOTE BLOCK in a subprogram.

SP-17 *%s1 statement is not FORTRAN 77 standard*

The statement %s1 is an extension to the FORTRAN 77 language.

SP-18 *%s1 block is unfinished*

The block starting with the statement %s1 does not have the ending block statement. For example: ATENDDO-ENDATEND, DO-ENDDO, GUESS-ENDGUESS, IF-ENDIF, LOOP-ENDLOOP, SELECT-ENDSELECT, STRUCTURE-ENDSTRUCTURE and WHILE-ENDWHILE.

SP-19 *%s1 statement does not match with %s2 statement*

The statement %s1, which ends a control structure, cannot be used with statement %s2 to form a control structure. Valid control structures are: LOOP - ENDLOOP, LOOP - UNTIL, WHILE - ENDWHILE, and WHILE - UNTIL.

SP-20 *incomplete control structure found at %s1 statement*

The ending control structure statement %s1 was found and there was no preceding matching beginning statement. Valid control structures include: ATENDDO - ENDATEND, GUESS - ENDGUESS, IF - ENDIF, LOOP - ENDLOOP, REMOTE BLOCK - ENDBLOCK, and SELECT - ENDSELECT.

SP-21 *%s1 statement is not allowed in %s2 definition*

Statement %s1 is not allowed between a %s2 statement and the corresponding END %s2 statement. For example, an EXTERNAL statement is not allowed between a STRUCTURE and END STRUCTURE statement, a UNION and END UNION statement, or a MAP and END MAP statement.

SP-22 *no such field name found in structure %s1*

A structure reference contained a field name that does not belong to the specified structure.

SP-23 *multiple definition of field name %s1*

The field name %s1 has already been defined in a structure.

SP-24 *structure %s1 has not been defined*

An attempt has been made to declare a symbol of user-defined type %s1. No structure definition for %s1 has occurred.

SP-25 *structure %s1 has already been defined*

The specified structure has already been defined as a structure.

SP-26 *structure %s1 must contain at least one field*

Structures must contain at least one field definition.

SP-27 *recursion detected in definition of structure %s1*

Structure %s1 has been defined recursively. For example, it is illegal for structure X to contain a field that is itself a structure named X.

SP-28 *illegal use of structure %s1 containing union*

Structures containing unions cannot be used in formatted I/O statements or data initialized.

SP-29 *allocatable arrays cannot be fields within structures*

An allocatable array cannot appear as a field name within a structure definition.

SP-30 *an integer conditional expression is not FORTRAN 77 standard*

A conditional expression is the expression that is evaluated and checked to determine a path of execution. A conditional expression can be found in an IF or WHILE statement. FORTRAN 77 requires that the conditional expression be a logical expression. As an extension, an integer expression is also allowed. When an integer expression is used, it is converted to a logical expression by comparing the value of the integer expression to zero.

SP-31 *%s1 statement must be used within %s2 definition*

The statement identified by %s1 must appear within a definition identified by %s2.

1.36 Subprograms

SR-01 *name can only appear in an EXTERNAL statement once*

A function/subroutine name appears more than once in an EXTERNAL statement.

SR-02 *character function %s1 may not be called since size was declared as (*)*

In the declaration of the character function name, the length was defined to be (*). The (*) length specification is only allowed for external functions, dummy arguments or symbolic character constants.

SR-03 *%s1 can only be used an an argument to a subroutine*

The specified class of an argument must only be passed to a subroutine. For example, an alternate return specifier is illegal as a subscript or an argument to a function.

SR-04 *name cannot appear in both an INTRINSIC and EXTERNAL statement*

The same name appears in an INTRINSIC statement and in an EXTERNAL statement.

SR-05 *expecting a subroutine name*

The subroutine named in the CALL statement does not define a subroutine. A subroutine is declared in a SUBROUTINE statement.

SR-06 *%s1 statement not allowed in main program*

The main program can contain any statements except a FUNCTION, SUBROUTINE, BLOCK DATA, or ENTRY statement. A SAVE statement is allowed but has no effect in the main program. A RETURN statement in the main program is an extension to the FORTRAN 77 language.

SR-07 *not an intrinsic FUNCTION name*

A name in the INTRINSIC statement is not an intrinsic function name. Refer to the Language Reference for a complete list of the intrinsic functions.

SR-08 *name can only appear in an INTRINSIC statement once*

An intrinsic function name appears more than once in the intrinsic function list.

SR-09 *subprogram recursion detected*

An attempt has been made to recursively invoke a subprogram, that is, to invoke an already active subprogram.

SR-10 *two main program units in the same file*

There are two places in the program that signify the start of a main program. The PROGRAM statement or the first statement that is not enclosed by a PROGRAM, FUNCTION, SUBROUTINE or BLOCK DATA statement specifies the main program start.

SR-11 *only one unnamed %s1 is allowed in an executable program*

There may only be one unnamed BLOCK DATA subprogram or main program in an executable program.

SR-12 *function referenced as a subroutine*

An attempt has been made to invoke a function using the CALL statement.

SR-13 *attempt to invoke active function/subroutine*

An attempt has been made to invoke the current function/subroutine or a function/subroutine that was used to invoke current function/subroutine. The traceback produced when the error occurred lists all currently active functions/subroutines.

SR-14 *dummy argument %s1 is not in dummy argument list of entered subprogram*

The named dummy argument found in the ENTRY statement does not appear in the subroutine's dummy argument list in the subprogram statement.

SR-15 *function referenced as %s1 but defined to be %s2*

An attempt has been made to invoke a function of the type %s1 but the function was defined as %s2 in the FUNCTION or ENTRY statement. The function name's type must be correctly declared in the main program.

SR-16 *function referenced as CHARACTER*%u1 but defined to be CHARACTER*%u2*

The character length of the function in the calling subprogram is %d1 but the length used to define the function is %d2. These two lengths must match.

SR-17 *missing main program*

The program file is either empty or contains only subroutines and functions. Each program require a main program. A main program starts with an optional PROGRAM statement and ends with an END statement.

SR-18 *subroutine referenced as a function*

An attempt has been made to invoke a name as a function and has been defined as a subroutine in a SUBROUTINE or ENTRY statement.

SR-19 *attempt to invoke a block data subprogram*

An attempt has been made to invoke a block data subprogram. Block data subprograms are used to initialize variables before program execution commences.

SR-20 *structure type of function %s1 does not match expected structure type*

The function returns a structure that is not equivalent to the structure expected. Two structures are equivalent if the types and orders of each field are the same. Unions are considered equivalent if their sizes are the same. Field names, and the structure name itself, do not have to be the same.

1.37 Subscripts and Substrings

SS-01 *substringing of function or statement function return value is not FORTRAN 77 standard*

The character value returned from a CHARACTER function or statement function cannot be substring. Only character variable names and array element names may be substring.

SS-02 *substringing valid only for character variables and array elements*

An attempt has been made to substring a name that is not defined to be of type CHARACTER and is neither a variable nor an array element.

SS-03 *subscript expression out of range; %s1 does not exist*

An attempt has been made to reference an element in an array that is out of bounds of the declared array size. The array element %s1 does not exist.

SS-04 *substring expression (%i1:%i2) is out of range*

An expression in the substring is larger than the string length or less than the value 1. The substring expression must be one in which .millust begin 1 <= %d1 <= %d2 <= len .millust end

1.38 Statements and Statement Numbers

ST-01 *statement number %i1 has already been defined in line %d2*

The two statements, in line %d2 and the current line, in the current program unit have the same statement label number, namely %d1.

ST-02 *statement function definition appears after first executable statement*

There is a statement function definition after the first executable statement in the program unit. Statement function definitions must follow specification statements and precede executable statements. Check that the statement function name is not an undeclared array name.

ST-03 *%s1 statement must not be branched to but was in line %d2*

Line %d2 passed execution control down to the statement %s1. The specification statements, ADMIT, AT END, BLOCK DATA, CASE, ELSE, ELSE IF, END AT END, END BLOCK, END DO, END LOOP, END SELECT, END WHILE, ENTRY, FORMAT, FUNCTION, OTHERWISE, PROGRAM, QUIT, REMOTE BLOCK, SAVE, SUBROUTINE, and UNTIL statements may not have control of execution transferred to it.

ST-04 *branch to statement %i1 in line %d2 not allowed*

An attempt has been made to pass execution control up to the statement labelled %d1 in line %d2. The specification statements, ADMIT, AT END, BLOCK DATA, CASE, ELSE, ELSE IF, END AT END, END BLOCK, END DO, END LOOP, END SELECT, END WHILE, ENTRY, FORMAT, FUNCTION, OTHERWISE, PROGRAM, QUIT, REMOTE BLOCK, SAVE, SUBROUTINE, and UNTIL statements may not have control of execution transferred to it.

ST-05 *specification statement must appear before %s1 is initialized*

The variable %s1 has been initialized in a specification statement. A COMMON or EQUIVALENCE statement then references the variable. The COMMON or EQUIVALENCE statement must appear before the item can be initialized. Use the DATA statement to initialize data in this case.

ST-06 *statement %i1 was referenced as a FORMAT statement in line %d2*

The statement in line %d2 references statement label %d1 as a FORMAT statement. The statement at that label is not a FORMAT statement.

ST-07 *IMPLICIT statement appears too late*

The current IMPLICIT statement is out of order. The IMPLICIT statement may be interspersed with the PARAMETER statement but must appear before other specification statements.

ST-08 *this statement will never be executed due to the preceding branch*

Because execution control will always be passed around the indicated statement, the statement will never be executed.

ST-09 *expecting statement number*

The keyword GOTO or ASSIGN has been detected and the next part of the statement was not a statement number as was expected.

ST-10 *statement number %i1 was not used as a FORMAT statement in line %d2*

The statement at line %d2 with statement number %d1 is not a FORMAT statement but the current statement uses statement number %d1 as if it labelled a FORMAT statement.

ST-11 *specification statement appears too late*

The indicated specification statement appears after a statement function definition or an executable statement. All specification statements must appear before these types of statements.

ST-12 *%s1 statement not allowed after %s2 statement*

The statement %s1 cannot be the object of a %s2 statement. %s2 represents a logical IF or WHILE statement. These statements include: specification statements, ADMIT, AT END, CASE, DO, ELSE, ELSE IF END, END AT END, END BLOCK, END DO, END GUESS, ENDIF, END LOOP, END SELECT, END WHILE, ENTRY, FORMAT, FUNCTION, GUESS, logical IF, block IF, LOOP, OTHERWISE, PROGRAM, REMOTE BLOCK, SAVE, SELECT, SUBROUTINE, UNTIL, and WHILE.

ST-13 *statement number must be 99999 or less*

The statement label number specified in the indicated statement has more than 5 digits.

ST-14 *statement number cannot be zero*

The statement label number specified in the indicated statement is zero. Statement label numbers must be greater than 0 and less than or equal to 99999.

ST-15 *this statement could branch to itself*

The indicated statement refers to a statement label number which appears on the statement itself and therefore could branch to itself, creating an endless loop.

ST-16 *missing statement number %i1 - used in line %d2*

A statement with the statement label number %d1 does not exist in the current program unit. The statement label number is referenced in line %d2 of the program unit.

ST-17 *undecodeable statement or misspelled word %s1*

The statement cannot be identified as an assignment statement or any other type of FORTRAN statement. The first word of a FORTRAN statement must be a statement keyword or the statement must be an assignment statement.

ST-18 *statement %i1 will never be executed due to the preceding branch*

The statement with the statement label number of %d1 will never be executed because the preceding statement will always pass execution control around the statement and no other reference is made to the statement label.

ST-19 *expecting keyword or symbolic name*

The first character of a statement is not an alphabetic. The first word of a statement must be a statement keyword or a symbolic name. Symbolic names must start with a letter (upper case or lower case), a dollar sign ($) or an underscore (_).

ST-20 *number in %s1 statement is longer than 5 digits*

The number in the PAUSE or STOP statement is longer than 5 digits.

ST-21 *position of DATA statement is not FORTRAN 77 standard*

The FORTRAN 77 standard requires DATA statements to appear after all specification statements. As an extension to the standard, Watcom FORTRAN 77 allows DATA statements to appear before specification statements. Note that in the latter case, the type of the symbol must be established before data initialization occurs.

ST-22 *no FORMAT statement with given label*

The current statement refers to the label of a FORMAT statement but the label appears on some other statement that is not a FORMAT statement.

ST-23 *statement number not in list or not the label of an executable statement*

The specified statement number in the indicated statement is not in the list of statement numbers or it is not the statement label number of an executable statement.

ST-24 *attempt to branch into a control structure*

An attempt has been made to pass execution control into a control structure. A statement uses a computed statement label number to transfer control. This value references a statement inside a control structure.

1.39 Subscripted Variables

SV-01 *variable %s1 in array declarator must be in COMMON or a dummy argument*

The variable %s1 was used as an array declarator in a subroutine or function but the variable was not in a COMMON block nor was it a dummy argument in the FUNCTION, SUBROUTINE or ENTRY statement.

SV-02 *adjustable/assumed size array %s1 must be a dummy argument*

The array %s1 used in the current subroutine or function must be a dummy argument. When the array declarator is adjustable or assumed-size, the array name must be a dummy argument.

SV-03 *invalid subscript expression*

The indicated subscript expression is not a valid integer expression or the high bound of the array is less than the low bound of the array when declaring the size of the array.

SV-04 *invalid number of subscripts*

The number of subscripts used to describe an array element does not match the number of subscripts in the array declaration. The maximum number of subscripts allowed is 7.

SV-05 *using %s1 name incorrectly without list*

An attempt has been made to assign a value to the declared array %s1. Values may only be assigned to elements in the array. An array element is the array name followed by integer expressions enclosed in parentheses and separated by commas.

SV-06 *cannot substring array name %s1*

An attempt has been made to substring the array %s1. Only an array element may be substring.

SV-07 *%s1 treated as an assumed size array*

A dummy array argument has been declared with 1 in the last dimension. The array is treated as if an '*' had been specified in place of the 1. This is done to support a feature called "pseudo-variable dimensioning" which was supported by some FORTRAN IV compilers and is identical in concept to FORTRAN 77 assumed-size arrays.

SV-08 *assumed size array %s1 cannot be used as an i/o list item or a format/unit identifier*

Assumed size arrays (arrays whose last dimension is '*') must not appear as an i/o list item (i.e. in a PRINT statement), a format identifier or an internal file specifier.

SV-09 *limit of 65535 elements per dimension has been exceeded*

On the IBM PC, for 16-bit real mode applications, the number of elements in a dimension must not exceed 65535.

1.40 Syntax Errors

SX-01 *unexpected number or name %s1*

The number or name %s1 is in an unexpected place in the statement.

SX-02 *bad sequence of operators*

The indicated arithmetic operator is out of order. An arithmetic operator is one of the following: **, *, /, +, and −. All arithmetic operators must be followed by at least a primary. A primary is an array element, constant, (expression), function name, or variable name.

SX-03 *invalid operator*

The indicated operator between the two arithmetic primaries is not a valid operator. Valid arithmetic operators include: **, *, /, +, and −. A primary is an array element, constant, (expression), function name, or variable name.

SX-04 *expecting end of statement after right parenthesis*

The end of the statement is indicated by the closing right parenthesis but more characters were found on the line. Multiple statements per line are not allowed in FORTRAN 77.

SX-05 *expecting an asterisk*

The next character of the statement should be an asterisk but another character was found instead.

SX-06 *expecting colon*

A colon (:) was expecting but not found. For example, the colon separating the low and high bounds of a character substring was not found.

SX-07 *expecting colon or end of statement*

On a control statement, a word was found at the end of the statement that cannot be related to the statement. The last word on several of the control statements may be a block label. All block labels must be preceded by a colon (:).

SX-08 *missing comma*

A comma was expected and is missing. There must be a comma after the statement keyword AT END when a statement follows. A comma must occur between the two statement labels in the GO TO statement. A comma must occur between the expressions in the DO statement. A comma must occur between the names listed in the DATA statement and specification statements. A comma must occur between the specifiers in I/O statements.

SX-09 *expecting end of statement*

The end of the statement was expected but more words were found on the line and cannot be associated to the statement. FORTRAN 77 only allows for one statement per line.

SX-10 *expecting integer variable*

The name indicated in the statement must be of type INTEGER but is not.

SX-11 *expecting %s1 name*

A name with the characteristic %s1 was expected at the indicated place in the statement but is missing.

SX-12 *expecting an integer*

The length specifier, as in the IMPLICIT statement, must be an integer constant or an integer constant expression. The repeat specifier of the value to be assigned to the variables, as in the DATA statement, must be an integer constant or an integer constant expression.

SX-13 *expecting INTEGER, REAL, or DOUBLE PRECISION variable*

The indicated DO variable is not one of the types INTEGER, REAL, or DOUBLE PRECISION.

SX-14 *missing operator*

Two primaries were found in an expression and an operator was not found in between. A primary is an array element, constant, (expression), function name, or variable name.

SX-15 *expecting a slash*

A slash is expected in the indicated place in the statement. Slashes must be balanced as parentheses. Slashes are used to enclose the initial data values in specification statements or to enclose names of COMMON blocks.

SX-16 *expecting %s1 expression*

An expression of type %s1 is required.

SX-17 *expecting a constant expression*

A constant expression is required.

SX-18 *expecting INTEGER, REAL, or DOUBLE PRECISION expression*

The indicated expression is not one of type INTEGER, REAL, or DOUBLE PRECISION. Each expression following the DO variable must be an expression of one of these types.

SX-19 *expecting INTEGER or CHARACTER constant*

In the PAUSE and STOP statement, the name following the keyword must be a constant of type INTEGER or of type CHARACTER. This constant will be printed on the console when the statement is executed.

SX-20 *unexpected operator*

An operand was expected but none was found. For example, in an I/O statement, the comma is used to separate I/O list items. Two consecutive commas without an I/O list item between them would result in this error.

SX-21 *no closing quote on literal string*

The closing quote of a literal string was not found before the end of the statement.

SX-22 *missing or invalid constant*

In a DATA statement, the constant required to initialize a variable was not found or incorrectly specified.

SX-23 *expecting character constant*

A character constant is required.

1.41 Type Statements

TY-01 *length specification before array declarator is not FORTRAN 77 standard*

An array declarator specified immediately after the length specification of the array is an extension to the FORTRAN 77 language.

TY-02 *%i1 is an illegal length for %s2 type*

The length specifier %d1 is not valid for the type %s2. For type LOGICAL, valid lengths are 1 and 4. For the type INTEGER, valid lengths are 1, 2, and 4. For the type REAL, valid lengths are 4 and 8. For the type COMPLEX, valid lengths are 8 and 16. On the IBM PC, the length specifier for items of type CHARACTER must be greater than 0 and not exceed 65535.

TY-03 *length specifier in %s1 statement is not FORTRAN 77 standard*

A length specifier in certain type specification statements is an extension to the FORTRAN 77 language. These include: LOGICAL*1, LOGICAL*4, INTEGER*1, INTEGER*2, INTEGER*4, REAL*4, REAL*8, COMPLEX*8, and COMPLEX*16.

TY-04 *length specification not allowed with type %s1*

A length specification is not allowed in a DOUBLE PRECISION or DOUBLE COMPLEX statement.

TY-05 *type of %s1 has already been established as %s2*

The indicated name %s1 has already been declared to have a different type, namely %s2. The name %s1 cannot be used in this specification statement.

TY-06 *type of %s1 has not been declared*

The indicated name %s1 has not been declared. This message is only issued when the IMPLICIT NONE specification statement is used.

TY-07 *%s1 of type %s2 is illegal in %s3 statement*

The symbol %s1 with type %s2 cannot be used in statement %s3. For example, a symbol of type STRUCTURE cannot be used in a PARAMETER statement.

1.42 Variable Names

VA-01 *illegal use of %s1 name %s2 in %s3 statement*

The name %s2 has been defined as %s1 and cannot be used as a name in the statement %s3.

VA-02 *symbolic name %s1 is longer than 6 characters*

Symbolic names greater than 6 characters is an extension to the FORTRAN 77 language. The maximum length is 32 characters.

VA-03 *%s1 has already been defined as a %s2*

The name %s1 has been previously defined as a %s2 in another statement and cannot be redefined as specified in the indicated statement.

VA-04 *%s1 %s2 has not been defined*

The name %s2 has been referenced to be a %s1 but has not been defined as such in the program unit.

VA-05 *%s1 is an unreferenced symbol*

The name %s1 has been defined but not referenced.

VA-06 *%s1 already belongs to this NAMELIST group*

The name %s1 can only appear in a NAMELIST group once. However, a name can belong to multiple NAMELIST groups.

VA-07 *%s1 has been used but not defined*

%s1 has not been defined before using it in a way that requires its definition. Note that symbols that are equivalenced, belong to a common block, are dummy arguments, or passed as an argument to a subprogram, will not be checked to ensure that they have been defined before requiring a value.

VA-08 *dynamically allocating %s1 is not FORTRAN 77 standard*

Allocatable storage are extensions to the FORTRAN 77 standard.

VA-09 *%s1 in NAMELIST %s2 is illegal*

Symbol %s1 appearing in NAMELIST %s2 is illegal. Symbols appearing in a NAMELIST cannot be dummy arguments, allocatable, or of a user-defined type.